Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 9615463, 17 pages
https://doi.org/10.1155/2021/9615463

Research Article

Modeling and Solving Scheduling in Overloaded Situations with Weighted Partial MaxSAT

Xiaojuan Liao,[1,2] **Hui Zhang**,[2] **Miyuki Koshimura**,[3] **Rong Huang**,[4] **Wenxin Yu**,[5] **and Fagen Li**[1]

[1]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China*
[2]*College of Computer Science and Cyber Security, Chengdu University of Technology, Chengdu, Sichuan 610059, China*
[3]*Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan*
[4]*College of Information Science and Technology, Donghua University, Shanghai 201620, China*
[5]*School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang, Sichuan 621010, China*

Correspondence should be addressed to Hui Zhang; zhanghui18@cdut.edu.cn

Received 24 May 2021; Revised 6 July 2021; Accepted 8 July 2021; Published 16 July 2021

Academic Editor: Mohammad Yazdi

In real-time systems, where tasks have timing requirements, once the workload exceeds the system's capacity, missed due dates may cause system overload. In this situation, finding an optimal scheduling that minimizes the cumulative values of late tasks is critical in both theory and practice. Recently, formalizing scheduling problems as a class of generalized problems, such as Satisfiability Modulo Theory (SMT) and Maximum Satisfiability (MaxSAT), has been receiving immense concern. Enlightened by the high efficiency of these satisfiability-based methods, this paper formulates the single-machine scheduling problem of minimizing the total weight of late tasks as a Weighted Partial Maximum (WPM) Satisfiability problem. In the formulation, scheduling features are encoded as rigidly enforced hard clauses and the scheduling objective is treated as a set of weighted soft ones. Then an off-the-shelf WPM solver is exploited to maximize the total weight of the satisfied soft clauses, provided that all the hard clauses are satisfied. Experimental results demonstrate that, compared with the existing satisfiability-based methods, the proposed method significantly improves the efficiency of identifying the optimal schedule. Moreover, we make minor changes to apply the WPM formulation to parallel-machine scheduling, showing that the proposed method is sufficiently flexible and well scalable.

1. Introduction

Real-time systems, which are designed to handle tasks with completion due dates, play an important role in a variety of modern applications, such as robotics [1], pacemakers [2], chemical plants [3], telecommunications [4], and multimedia systems [5]. Under ideal circumstances, a real-time system completes all tasks before their due dates expire. However, in reality, the workload may exceed the system's capacity, leading to missed deadlines [6]. Such a phenomenon is called overload. A classic example is a switch in a communication network which polls its incoming links to forward packets that have arrived since its previous servicing of the link [7]. On each link, different packets may have different processing times, deadlines, and importance values. When packages flood the switch, overload happens. In this situation, designing a suitable scheduling strategy to maximize the total value of forwarded packages is critical to maintain a service's stability.

Generally, scheduling algorithms can be classified as online scheduling and offline scheduling, depending on whether tasks' information is known a priori. In online scheduling, the scheduler receives tasks that arrive over time and must schedule tasks without any knowledge of the future. On the contrary, offline scheduling algorithms aim at solving the problem optimally, provided that all data are known beforehand [1](note that an offline algorithm does not contradict with a real-time system. An offline scheduling algorithm allows a scheduler to make decision based on the total knowledge of the problem, while a real-time system assigns each task with a specific due date). Although online scheduling is more flexible, in many situations, it is necessary to obtain optimal schedules by offline algorithms, especially in time-critical systems or for the evaluation of heuristics [8]. In this paper, we take an interest in designing an offline scheduling method on a single machine to minimize the total weight of tasks that miss their due dates.

To date, there has been an immense amount of work devoted to characterizing scheduling problems and analyzing the complexity of problems with specific characteristics. When preemption is prohibited, Michael Moore [9] presented an optimal algorithm in polynomial time to minimize the number of late tasks on a single machine, under the assumption that all tasks are released simultaneously without dependency relations. In the standard three-field notation [10], this problem is $1\|\sum U_j$. Adding weights on the criteria and setting precedence constraints may complicate the problem. Richard [11] showed that $1\|\sum w_j U_j$ is binary NP-hard but can be solved by dynamic programming in $O(n\sum p_j)$ time [12], where n is the number of tasks and p_j is the processing time of task τ_j. Garey and Johnson [13] showed that the problem of unit-time tasks subject to precedence constraints, that is, $1|\text{prec}, p_j = 1|\sum U_j$, is NP-hard. Furthermore, Lenstra and Rinnooy Kan [14] proved that, even for chain-like precedence constraints, where each task has at most one immediate predecessor and at most one immediate successor, the problem $1|\text{chains}, p_j = 1|\sum U_j$ is also NP-hard. Another prevailing common knowledge is that usually preemptive problems are not harder than their nonpreemptive counterparts. For example, the problem with release dates $1|r_j|\sum U_j$ is NP-hard [10], while the preemptive version $1|r_j, \text{pmtn}|\sum U_j$ can be solved in $O(n^3k^2)$ [15], where k is the number of distinct release dates. Further complexity results for single-machine scheduling problems are listed by [16] and the scheduling problems with the late work criteria are surveyed by [17].

Algorithms for solving scheduling are generally classified as online scheduling and offline scheduling, depending on whether the full information about tasks is known a priori or not. Online algorithms aim to return high-quality results within reasonable CPU time [18–21], whereas offline algorithms are devoted to optimally solving scheduling problems, given that all data are known beforehand. For large-scale problems that are computationally intractable, finding optimal solutions requires a significantly long computation time and heuristics are proposed to seek for suboptimal solutions within a short computation time [22–24]. Nevertheless, it is still of great significance to design optimization algorithms as a testbed for suboptimal solutions and reap huge benefit when the scheduled application is executed many times [8]. Previous attempts at finding optimal solutions to single-machine scheduling problems are mainly based on dynamic programming and branch-and-bound algorithms, with promising results. For a comprehensive survey, see the work by [25]. In the last decade, formalizing scheduling problems as a class of generalized problems, such as Mathematical Programming (MP) [26–31], Satisfiability Modulo Theory (SMT) [8, 32–34], Boolean Satisfiability (SAT) [35–37], and Partial Maximum (PM) Satisfiability [38], has received considerable attention. Motivated by the significant progress in solving these generalized problems, the formalized scheduling problem can be efficiently addressed with the corresponding solving algorithms.

As a pioneering work in satisfiability formalization, Crawford and Baker [35] first encoded scheduling problems into a SAT problem, paving the way for subsequent work [36] that solved six types of open job-shop scheduling problems. Venugopalan and Oliver [27], Liu et al. [37], and Malik et al. [8] presented optimization frameworks to address task graph scheduling with communication costs based on MIP, SAT, and SMT, respectively. Qi et al. [34] utilized task duplication strategy-based SMT formulation to mitigate the negative impact of the interprocessor communication delay in the task graph scheduling problem. Qamhan et al. [31] presented a new MILP model to schedule a set of tasks on a single-machine subject to nonzero release date, sequence-dependent setup time, and periodic maintenance. The objective of all the above formulations is to minimize the maximum completion time makespan.

To achieve the goal of minimizing the number of late jobs, Ourari et al. [26] designed a mathematical integer programming formulation for single-machine scheduling without preemption, and Hung et al. [29] developed a nonstandard MIP formulation to address the arbitrary preemptive version on parallel machines. The restricted preemptive counterpart on a single machine was solved by Cheng et al. [32], which encoded the problem as a set of first-order formulas that are tackled by an SMT solver called Z3. By running the Z3 solver repeatedly to identify the maximum number of on-time tasks, the optimal schedule could be finally determined. The SMT-based scheduling is sufficiently flexible because it handles various task properties and objectives with very few changes in adaption procedure. For example, when tasks have different importance values and the objective turns to minimizing the total weight of late tasks, only the target constraints need to be modified [33]. Later on, Wang et al. [39] enhanced Cheng's SMT formulation by removing redundant constraints and eliminating successive calls of the Z3 solver. Experiments illustrated that the updated formulation improved the efficiency by more than two orders of magnitude. Recently, Liao et al. [38] encoded the unweighted version of the same scheduling problem into Boolean propositional logic and showed that PM solvers are a competitive alternative to SMT solvers. However, we notice that when encoding scheduling features, PM formulation generates redundant variables and clauses. Such redundancy may create extra calculations and decrease the overall performance. Furthermore, PM formulation presented in [38] is incapable of handling weighted problems where the scheduling goal is to minimize the total weight of late tasks.

In this paper, we present a Weighted Partial MaxSAT (WPM) formulation to optimally solve scheduling in overloaded situations, with the aim of minimizing the total weight value of late tasks. The minimization objective is equivalent to maximizing the total weight of the tasks meeting their due dates. Confronted with a weighted scheduling problem, we first identify scheduling features that uniquely characterize the problem, facilitating the WPM formulation in the encoding phase. Then, enlightened by the WPM characteristics that satisfy all hard clauses and maximize the total weight of the satisfied soft clauses, we recast the weighted scheduling problem as a WPM problem. Finally, in the problem-solving phase, we exploit the off-the-shelf WPM solver to satisfy all the scheduling features and maximize the total weight of tasks meeting their due dates, thus deriving the optimal schedule from the output of the WPM solver. Specifically, we make the following contributions:

(i) We extend the PM formulation in [38], which was originally designed for scheduling tasks without weights, to adapt to the weighted cases.

(ii) Having noticed that redundant Boolean variables and clauses exist in the previous formulation [38], we develop a more compact encoding to characterize the scheduling problem. Particularly, to denote a task's completion time, the number of Boolean variables generated by [38] is proportional to the number of possible preemptions of the task. In contrast, the novel compact encoding generates only one Boolean variable to represent the task's completion time, no matter how many times the task may be preempted. Theoretical analysis shows the correctness of the compact encoding and experiments demonstrate the substantial advantages over the previous PM encoding [38] and SMT formulation [39], which was enhanced from [33].

(iii) In the WPM encoding, task features are encoded by several separate rules. This means if some of the task features happen to change, only partial rules need to be modified. To demonstrate the flexibility of our formulation, we extend the current WPM encoding to adapt to parallel-machine scheduling with little modification. We believe that the proposed WPM encoding can help users readily and effectively design scheduling for practical systems with low design cost.

Confronted with a weighted scheduling problem, we first identify scheduling features that uniquely characterize the problem, facilitating the WPM formulation in the encoding phase. Then, enlightened by the WPM characteristics that satisfy all hard clauses and maximize the total weight of the satisfied soft clauses, we recast the weighted scheduling problem as a WPM problem. In the WPM formulation, tasks' features are encoded as a set of hard clauses, and the goal of completing tasks before their due dates is transformed into a set of weighted soft clauses. Finally, in the problem-solving phase, we exploit the off-the-shelf WPM solver to satisfy all the scheduling features and maximize the total weight of tasks meeting their due dates, thus deriving the optimal schedule from the output of the WPM solver.

To evaluate the performance of the proposed WPM formulation, we compare it with the state-of-the-art SMT formulation [39]. Our evaluation shows that WPM formulation has a dominant advantage over the SMT-based method in finding out the optimal schedule. We also compare WPM with the latest PM formulation [38] on a special case, where all the tasks have equal weights. We reveal redundancies in PM and discuss how they can be avoided in WPM. Experiments show that our WPM encoding is more compact and more time-efficient than PM for solving the same set of problem instances. Furthermore, we show that the presented approach is sufficiently flexible to adapt to parallel-machine scheduling problems with minor changes.

The remainder of this paper is organized as follows. The scheduling model is described in Section 2, followed by the WPM-based optimization framework in Sections 3. We make theoretical comparisons between WPM and PM formulation [38] in Section 4 and provide experimental results to show the superiority of WPM in Section 5. Section 6 exhibits how to extend the WPM formulation of single-machine scheduling to parallel-machine scheduling. Finally, we conclude this paper in Section 7.

2. Scheduling Model

We adhere to the definition of scheduling problems in previous works [33, 39]. For convenience, the notations used in the model are summarized in Table 1.

The problem involves n tasks $\Gamma = \{\tau_1, \dots, \tau_n\}$ to be processed. All the tasks request a uniprocessor for execution when they arrive in the system. Each task $\tau_\ell \in \Gamma$ is represented by a 4-tuple $\tau_\ell = (r_\ell, c_\ell, d_\ell, w_\ell)$, where r_ℓ, c_ℓ, d_ℓ, and w_ℓ are all nonnegative integers [2](assuming all parameters to be integers does not make the problem less general since real numbers can be scaled to integers with a few orders of magnitude) representing the release date, the execution time, the due date, and the weight of τ_ℓ, respectively. Naturally, $r_\ell + c_\ell \le d_\ell$. Weight w_ℓ reflects the importance of task τ_ℓ. The larger w_ℓ is, the more important τ_ℓ is. Given due date d_ℓ, if task τ_ℓ is completed at or before d_ℓ, τ_ℓ is on time and weight w_ℓ is obtained by the system. Otherwise, τ_ℓ is late and worthless to the system.

To allow preemption, which indicates that a running task may be interrupted and resumed later, each task $\tau_\ell \in \Gamma$ is split into q_ℓ nonpreemptive subtasks (fragments) and is defined as a chain of indivisible fragments $\langle f_1^\ell, \dots, f_{q_\ell}^\ell \rangle$. Symbol c_i^ℓ stands for the required execution time of f_i^ℓ. Clearly, $\sum_{i=1}^{q_\ell} c_i^\ell = c_\ell$. For $2 \le i \le q_\ell$, f_i^ℓ can only start to run after f_{i-1}^ℓ is completed.

In practical systems, tasks usually have dependency relations. For example, if task τ_ℓ requires the computed result of τ_k, τ_ℓ cannot start until τ_k is finished. Such a dependency relation between tasks is written as $\tau_k \prec \tau_\ell$, where τ_k is the immediate predecessor of τ_ℓ, and τ_ℓ is the immediate successor of τ_k. Obviously, if $\tau_k \prec \tau_\ell$, the constraint $r_k + c_k \le d_\ell - c_\ell$ should be satisfied; otherwise, τ_ℓ can never

Table 1: Notations and descriptions in scheduling model.

Notation	Description
Γ	Finite set of real-time tasks
n	Number of real-time tasks. $n = \|\Gamma\|$
τ_ℓ	Task in Γ, where ℓ is its index
r_ℓ	Release date of τ_ℓ
c_ℓ	Execution time of τ_ℓ
d_ℓ	Due date of τ_ℓ
f_i^ℓ	ith fragment of τ_ℓ
q_ℓ	Number of fragments in τ_ℓ
c_i^ℓ	Execution time of f_i^ℓ
$\tau_k \prec \tau_\ell$	τ_ℓ succeeds τ_k
R_{dp}	Set of task pairs that have dependency relations

be completed no matter when its predecessor finishes. The set of task pairs that have dependency relations over Γ is denoted by $R_{\text{dp}} = \{(\tau_k, \tau_\ell): \tau_k, \tau_\ell \in \Gamma, \tau_k \prec \tau_\ell\}$.

A system is defined as overloaded if no scheduling algorithm can meet the due dates of all the tasks that have been submitted to it. This paper focuses on designing an exact method to tackle overloaded single-machine scheduling problems. The scheduling objective is to maximize the total weight of the on-time tasks. In particular, if all tasks have equal weights, then a schedule that maximizes the total weight will be one that maximizes the number of on-time tasks.

3. WPM-Based Optimization Framework

In this section, we provide the WPM formulation for solving the task scheduling problem on a single machine. The overview of the WPM formulation is illustrated in Figure 1.

Feature preprocessing identifies the scheduling problem with two types of constraints, that is, constraints on scheduling features and those on objective. Given a set of tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$, scheduling features uniquely characterize the problem over Γ and the objective is to seek for a schedule that completes all tasks on time subject to constraints on the scheduling features. In overloaded situations, making all the tasks complete by their due dates is impossible, and the scheduling problem is then treated as an optimization problem, which aims to maximize the total weight of tasks that are completed by their due dates. After feature preprocessing, WPM encoding can be implemented separately on the scheduling features and objective. Specifically, scheduling features, which are intrinsic and determined when tasks are released, are encoded into a set of hard clauses that should be satisfied without exception. On the other hand, the objective in overloaded situations is encoded as a set of weighted soft clauses that are allowed to be unsatisfied. By conjunction of hard clauses with weighted soft clauses, the problem turns to a WPM problem that can be addressed by any off-the-shelf WPM solver. In the problem-solving phase, a WPM solver tries to satisfy all the hard clauses and maximize the total weight of satisfied soft clauses. The output of the WPM solver includes the assignment of all Boolean variables, from which the optimal schedule can be derived.

Section 3.1 shows how to identify fragments' critical time points to characterize the scheduling problem, paving the way for the WPM encoding described in Sections 3.2. Section 3.3 exhibits an example to show how the scheduling problem is addressed with the presented WPM encoding. Details on the problem-solving phase are also exhibited in Section 3.3.

3.1. Feature Preprocessing

3.1. Feature Preprocessing. Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$, where each task $\tau_\ell \in \Gamma$ is characterized by $(r_\ell, c_\ell, d_\ell, w_\ell)$, $\forall \tau_\ell \in \Gamma$, τ_ℓ cannot start before its release date r_ℓ. Thus, the possible Earliest Start Time (EST) of τ_ℓ, denoted by EST_ℓ, is equal to its release date r_ℓ, and the Earliest Completion Time (ECT) of τ_ℓ, denoted by ECT_ℓ, is $\text{EST}_i + c_i$. $\forall (\tau_k, \tau_\ell) \in R_{\text{dp}}$, EST_ℓ is jointly restricted by r_ℓ and ECT_k. Specifically, if $\text{ECT}_k \le r_\ell$, then $\text{EST}_\ell = r_\ell$; otherwise, $\text{EST}_\ell = \text{ECT}_k$, suggesting that τ_ℓ should wait until τ_k is finished. Typically, let P_ℓ be the set of predecessors of τ_ℓ; then $\text{EST}_\ell = \max\{r_\ell, \max_{\tau_k \in P_\ell}\{\text{ECT}_k\}\}$.

After EST_ℓ for each τ_ℓ in Γ has been determined, EST and ECT of each fragment in τ_ℓ can be calculated as follows:

(i) Earliest Start Time (EST) of f_i^ℓ is denoted by EST_i^ℓ, where $\text{EST}_i^\ell = \text{EST}_\ell + \sum_{u=1}^{i-1} c_u^\ell$. $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, f_i^ℓ should start at or after EST_i^ℓ, indicating that f_i^ℓ cannot be started before all its previous fragments are finished. Typically, $\text{EST}_1^\ell = \text{EST}_\ell$.

(ii) Earliest Completion Time (ECT) of f_i^ℓ is denoted by ECT_i^ℓ, where $\text{ECT}_i^\ell = \text{EST}_\ell + \sum_{u=1}^{i} c_u^\ell$. No fragment f_i^ℓ can be completed before ECT_i^ℓ. Particularly, $\text{ECT}_i^\ell = \text{EST}_i^\ell + c_i^\ell$.

In addition to EST and ECT, $\forall \tau_\ell \in \Gamma$, each fragment $f_i^\ell \in \tau_\ell$ is characterized by the two following types of time:

(i) Latest Start Time (LST) of f_i^ℓ is denoted by LST_i^ℓ, where $\text{LST}_i^\ell = d_\ell - \sum_{u=i}^{q_\ell} c_u^\ell$. If f_i^ℓ fails to start before LST_i^ℓ, τ_ℓ cannot end by its due date d_ℓ, and thus this task becomes worthless to the system.

(ii) Latest Completion Time (LCT) of f_i^ℓ is denoted by LCT_i^ℓ, where $\text{LCT}_i^\ell = d_\ell - \sum_{u=i+1}^{q_\ell} c_u^\ell$. Particularly, $\text{LCT}_i^\ell = \text{LST}_i^\ell + c_i^\ell$.

Note that, in the above four critical time points, ECT is unrelated to our WPM encoding. Nevertheless, it is indispensable in the previous work [38], which will be discussed in Section 4..

3.2. WPM Formulation. This section introduces how to encode the scheduling problem as a WPM problem. A WPM instance consists of a number of clauses that need to be managed by the WPM solver. To formulate all the necessary constraints that characterize the scheduling model, we introduce the three following Boolean variables:

(i) $\text{sa}_{i,t}^\ell$, which is true if f_i^ℓ starts at time t or later

(ii) $\text{pr}_{i,j}^{\ell,k}$, which is true if f_i^ℓ precedes f_j^k

(iii) eb_ℓ, which is true if τ_ℓ ends by its due date d_ℓ

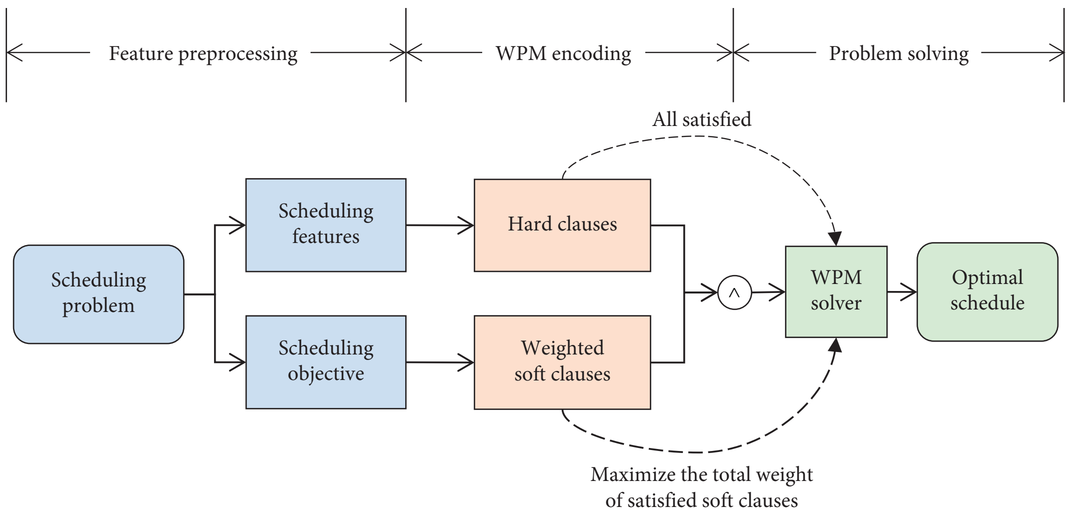


FIGURE 1: Overview of the WPM-based scheduling method.

Based on the generated Boolean variables, the WPM encoding can be implemented. In what follows, several rules are presented to encode the features of fragments into a set of hard clauses. The main encoding is derived and enhanced from previous work [35, 36, 38]. Logical implication $a \longrightarrow b$ is equivalent to $\bar{a} \vee b$ in classical logic.

(i) (C1) $\forall \tau_\ell \in \Gamma$, the first fragment of τ_ℓ, that is, f_1^ℓ, starts at or after EST_1^ℓ:

$$\mathrm{sa}_{1,\mathrm{EST}_1^\ell}^\ell. \tag{1}$$

(ii) (C2) $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell, f_{i+1}^\ell \in \tau_\ell$, f_i^ℓ precedes f_{i+1}^ℓ:

$$\mathrm{pr}_{i,i+1}^{\ell,\ell}. \tag{2}$$

(iii) (C3) $\forall \tau_k, \tau_\ell \in \Gamma$, $\forall f_i^k \in \tau_k$, and $\forall f_j^\ell \in \tau_\ell$, if $k \neq l$, $\tau_k \nprec \tau_\ell$, $\tau_\ell \nprec \tau_k$, $\mathrm{EST}_i^k < \mathrm{LCT}_j^\ell$ and $\mathrm{EST}_j^\ell < \mathrm{LCT}_i^k$, and then f_i^k and f_j^ℓ may require the processor at the same time. In this condition, f_i^k precedes f_j^ℓ or f_j^ℓ precedes f_i^k:

$$\mathrm{pr}_{i,j}^{k,\ell} \vee \mathrm{pr}_{j,i}^{\ell,k}. \tag{3}$$

(iv) (C4) $\forall (\tau_k, \tau_\ell) \in R_{dp}$, if τ_k fails to be completed by its due date, then τ_ℓ cannot even start at LST_1^ℓ:

$$\mathrm{eb}_k \longrightarrow \overline{\mathrm{sa}_{1,\mathrm{LST}_1^\ell+1}^\ell}. \tag{4}$$

(v) Furthermore, if $d_k > \mathrm{EST}_1^\ell$, the first fragment of τ_ℓ cannot start until the last fragment of τ_k finishes. That is, if $d_k > \mathrm{EST}_1^\ell$, then the constraint that $f_{q_k}^k$ precedes f_1^ℓ is enforced:

$$\mathrm{pr}_{q_k,1}^{k,\ell}. \tag{5}$$

(vi) (C5)$\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, if f_i^ℓ starts at or after time t, then it starts at or after time $t-1$:

$$\mathrm{sa}_{i,t}^\ell \longrightarrow \mathrm{sa}_{i,t-1}^\ell, \quad \left(\mathrm{EST}_i^\ell + 1 \leq t \leq \mathrm{LST}_i^\ell + 1\right). \tag{6}$$

(vii) (C6) $\forall \tau_\ell \in \Gamma$, if τ_ℓ ends before its due date d_ℓ, then last fragment $f_{q_\ell}^\ell$ must start at or before time $\mathrm{LST}_{q_\ell}^\ell$. In other words, $f_{q_\ell}^\ell$ cannot start at or after time $\mathrm{LST}_{q_\ell}^\ell + 1$:

$$\overline{\mathrm{eb}_\ell} \longrightarrow \overline{\mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell}. \tag{7}$$

(viii) (C7) $\forall \tau_k, \tau_\ell \in \Gamma$, $\forall f_i^k \in \tau_k$, and $\forall f_j^\ell \in \tau_\ell$, if f_i^k starts at or after time t and f_j^ℓ follows f_i^k, then f_j^ℓ cannot start until f_i^k is finished. That is, for each $\mathrm{pr}_{i,j}^{k,\ell}$ asserted by (C2) ~ (C4), one clause is generated:

$$\mathrm{sa}_{i,t}^k \wedge \mathrm{pr}_{i,j}^{k,\ell} \longrightarrow \mathrm{sa}_{j,t'}^\ell, \tag{8}$$

where t varies in $[\mathrm{EST}_i^k, \mathrm{LST}_i^k + 1]$ and

$$t' = \begin{cases} \mathrm{LST}_j^\ell + 1, & \text{if } t + c_i^k > \mathrm{LST}_j^\ell, \\ \mathrm{EST}_j^\ell, & \text{if } t + c_i^k < \mathrm{EST}_j^\ell, \\ t + c_i^k, & \text{Otherwise.} \end{cases} \tag{9}$$

This formula reveals the following facts: First, if f_i^k ends after LST_j^ℓ (i.e., $t + c_i^k > \mathrm{LST}_j^\ell$), then f_j^ℓ cannot start at or before LST_j^ℓ. Second, if f_i^k finishes before EST_j^ℓ (i.e., $t + c_i^k < \mathrm{EST}_j^\ell$), then f_j^ℓ starts at or after EST_j^ℓ. Otherwise, f_j^ℓ must start at or after time f_i^k finishes, that is, $t + c_i^k$.

Up to this point, we have encoded the scheduling features as Boolean formulas that can be converted to a set of clauses. Since the scheduling features are intrinsic properties inherent in the tasks and their fragments, such clauses are specified as hard, indicating that all of them must absolutely be satisfied. For convenience, we refer to the set of hard clauses introduced in (C1) ~ (C7) as $\mathscr{C}$.

A task is said to be on time if and only if it is completed before its due date. Thus, the scheduling objective can be directly encoded by the following rule:

(O) Maximizing the sum of the weights of on-time tasks:

$$(\mathrm{eb}_\ell, w_\ell), \quad (1 \le \ell \le n). \tag{10}$$

Equation (10) indicates that if clause eb_ℓ is satisfied (i.e., evaluating to true), then weight w_ℓ is gained; otherwise, the gain is zero. To simplify the discussion, we introduce $\mathcal{O}$ to denote the set of clauses in equation (10). In an overloaded system, not all tasks can be completed before their due dates. To handle such situations, we declare the clauses in $\mathcal{O}$ to be soft, indicating that completing all the tasks by their due dates is a soft constraint. Conjunct with $\mathcal{C}$, the problem is then $\{\mathcal{C}, \mathcal{O}\}$. This leads to a WPM problem, which tries to find an assignment of variables to satisfy all the hard clauses in $\mathcal{C}$ and to maximize the sum of the weights of the satisfied soft clauses in $\mathcal{O}$, that is, to maximize $\sum_{\ell=1}^{n} w_\ell \cdot \mathrm{eb}_\ell$.

3.3. A Pedagogical Example. Let us consider a simple scheduling problem to describe how WPM formulation works. As shown in Figure 2(a), there are a set of real-time tasks $\Gamma = \{\tau_1, \tau_2, \tau_3\}$, where both τ_1 and τ_3 rely on the computed result of τ_2, represented as $\tau_2 \prec \tau_1$ and $\tau_2 \prec \tau_3$; that is, $R_{\mathrm{dp}} = \{(\tau_2, \tau_1), (\tau_2, \tau_3)\}$. The release dates, the execution times, the due dates, and the weights of these tasks are, respectively, defined as $\tau_1 = (0, 2, 3, 1)$, $\tau_2 = (0, 1, 1, 2)$, and $\tau_3 = (0, 1, 2, 3)$. Suppose that τ_1 has two fragments, that is, $\langle f_1^1, f_2^1 \rangle$. τ_2 and τ_3 each have one, denoted by f_1^2 and f_1^3, respectively. The execution time of each fragment is 1.

The refined problem exhibition and the critical time points after feature preprocessing are summarized in Figure 2(b) and Table 2, respectively. Since ECT has nothing to do with the WPM encoding, we omit this entry in Table 2 for conciseness.

The MaxSAT formulation applied to the scheduling problem is shown in Figure 3. Constraint (C1) states that each task starts at or after its EST. Constraints (C2) and (C3) work together to specify the execution sequence of the fragments. (C2) forces all the fragments in a single task to be executed sequentially, and (C3) guarantees no overlap of the execution times of any two fragments in different tasks. In the three tasks, only τ_1 has more than one fragment, and hence constraint (C2) only applies to τ_1, ensuring that f_1^1 precedes f_2^1. Constraint (C3) applies to pairs of fragments f_i^k and $f_j^\ell\,(k \ne l)$ satisfying $\mathrm{EST}_i^k < \mathrm{LCT}_j^\ell$ and $\mathrm{EST}_j^\ell < \mathrm{LCT}_i^k$. If both conditions are met, then f_i^k and f_j^ℓ may simultaneously occupy the processor, and thus we need to decide in what order to execute them. Constraint (C3) tackles this ordering dilemma, which states that either one can precede the other. Consider f_1^1 and f_1^3. As seen in Table 2, $\mathrm{EST}_1^1 < \mathrm{LCT}_1^3$ and $\mathrm{EST}_1^3 < \mathrm{LCT}_1^1$; hence, we must explicitly specify that f_1^1 precedes f_1^3 or f_1^3 precedes f_1^1; otherwise, the execution time of these two fragments may overlap. Constraint (C4) applies to a situation where tasks have dependency relations. In the example, τ_2 is the predecessor of both τ_1 and τ_3; thus we should give up processing τ_1 and τ_3 if τ_2 misses its due date. In addition, given $d_2 = \mathrm{EST}_1^1$ and $d_2 = \mathrm{EST}_1^3$, there is no need to explicitly specify the execution sequence of the predecessor and successors since τ_2 must have finished before τ_1 and τ_3 start. Constraints (C5) ~ (C7) are partially extracted from a collection of coherence conditions [35] on the introduced variables for all the fragments of all the tasks. Finally, constraint (O) gives the problem's objective, that is, completing the last fragment of each task by its due date.

All the clauses are conjunct with $\wedge$ to form a WPM problem in CNF, where clauses (C1) ~ (C7) are declared hard and those in (O) are soft. Then the CNF formula is input to a WPM solver. The solver's output includes the maximum sum of the weights of the satisfied soft clauses as well as the corresponding assignment of all the Boolean variables.

In the exemplified problem, the assigned truth values of all the Boolean variables are listed in Figure 4, from which the exact start time of each fragment can be derived. Consider f_1^2. $\mathrm{sa}_{1,0}^2 = 1$ and $\mathrm{sa}_{1,1}^2 = 0$ indicate that f_1^2 starts at or after time 0, but it does not start at or after time 1. Then we can readily determine that f_1^2 starts at time 0. A similar reference can be made on f_1^3, which starts at time 1. Both τ_2 and τ_3 can be completed by their due dates. This is ensured by $\mathrm{eb}_2 = 1$ and $\mathrm{eb}_3 = 1$. Now, consider f_1^1. Figure 4 shows that $\mathrm{sa}_{1,1}^1 = 1$ and $\mathrm{sa}_{1,2}^1 = 1$. Since $\mathrm{LST}_1^1 = 1$, that $\mathrm{sa}_{1,2}^1 = 1$ means that f_1^1 cannot start at or before LST_1^1. Similarly, as indicated by the fact that $\mathrm{sa}_{2,3}^1 = 1$, f_2^1 cannot start at or before LST_2^1 either. Thus, we infer that τ_1 cannot finish on time. This is confirmed by $\mathrm{eb}_1 = 0$. As a result, the maximized total weight of the completed on-time tasks is $2 + 3 = 5$, achieved by completing τ_2 and τ_3 by their due dates.

4. Theoretical Discussion

Recall that, in the scheduling model described in Section 2, each task τ_ℓ is represented as a 4-tuple $\tau_\ell = (r_\ell, c_\ell, d_\ell, w_\ell)$. Consider a special case where the weights of all the tasks are equal to a constant w, that is, $\forall \tau_\ell \in \Gamma$, $w_\ell = w$. According to Section 3.2, to encode this problem, a set of weighted soft clauses $\mathcal{O} = \{(\mathrm{eb}_\ell, w)\}$ are introduced, where $1 \le \ell \le n$. Conjunct with hard clauses $\mathcal{C}$, problem $\{\mathcal{C}, \mathcal{O}\}$ becomes a special WPM problem with equal weights, which tries to find an assignment of variables to satisfy all hard clauses in $\mathcal{C}$ and to maximize $w \cdot \sum_{\ell=1}^{n} \mathrm{eb}_\ell$. Obviously, the essence of maximizing $w \cdot \sum_{\ell=1}^{n} \mathrm{eb}_\ell$ is equivalent to maximizing $\sum_{\ell=1}^{n} \mathrm{eb}_\ell$. This is a PM problem encoded by [38], where each task τ_ℓ is modeled as a 3-tuple $\tau_\ell = (r_\ell, c_\ell, d_\ell)$, and the scheduling goal is to maximize the total number of on-time tasks. In this section, we make theoretical comparisons between PM-based [38] and our WPM-based optimization frameworks when all the tasks have equal weights.

4.1. Similarities of PM and WPM Formulations. Essentially, the purpose of solving the scheduling problem is to determine the start time of each fragment to construct an optimal scheduling solution. To describe such crucial information, $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, a set of Boolean variables $\mathrm{sa}_{i,t}^\ell\ (\mathrm{EST}_i^\ell \le t \le \mathrm{LST}_i^\ell + 1)$ are introduced to indicate whether f_i^ℓ starts at or after time t. As long as the truth values of sa variables are all assigned, we can readily determine the exact start time of each fragment. To be specific, if we find a certain time $t\prime \in [\mathrm{EST}_i^\ell, \mathrm{LST}_i^\ell]$ such that $\mathrm{sa}_{i,t}^\ell = 1$ for $t \le t'$ and $\mathrm{sa}_{i,t}^\ell =$

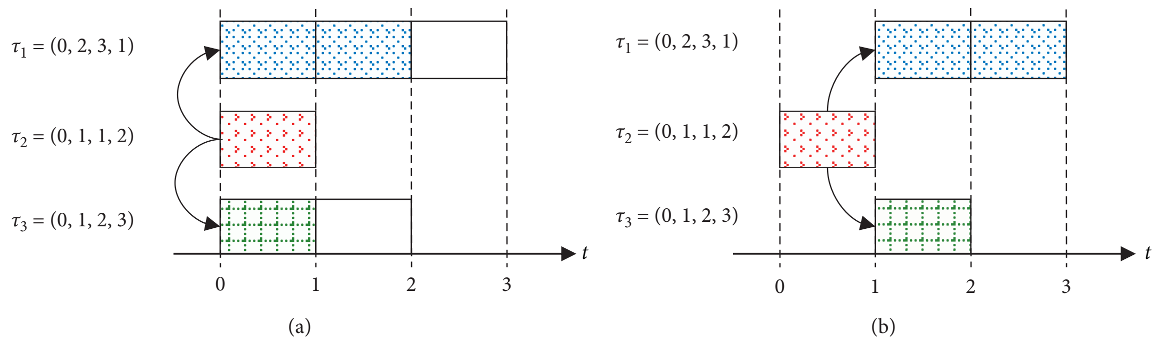


Figure 2: A simple scheduling example. (a) Original problem. (b) Problem after feature preprocessing.

Table 2: Critical time points of each fragment in three tasks.

Task	Fragment	EST	LST	LCT
τ_1	f_1^1	1	1	2
	f_2^1	2	2	3
τ_2	f_1^2	0	0	1
τ_3	f_1^3	1	1	2

(C1)	$sa_{1,1}^1$	$\wedge$		(C7)	$\neg sa_{1,1}^1 \vee \neg pr_{1,2}^{1,1} \vee sa_{2,2}^1$
$\wedge$	$sa_{1,0}^2$	(C5)	$\neg sa_{1,2}^1 \vee sa_{1,1}^1$	$\wedge$	$\neg sa_{1,2}^1 \vee \neg pr_{1,2}^{1,1} \vee sa_{2,3}^1$
$\wedge$	$sa_{1,1}^3$	$\wedge$	$\neg sa_{2,3}^1 \vee sa_{2,2}^1$	$\wedge$	$\neg sa_{1,1}^1 \vee \neg pr_{1,1}^{1,3} \vee sa_{1,2}^3$
$\wedge$		$\wedge$	$\neg sa_{1,1}^2 \vee sa_{1,0}^2$	$\wedge$	$\neg sa_{1,2}^1 \vee \neg pr_{1,1}^{1,3} \vee sa_{1,2}^3$
(C2)	$pr_{1,2}^{1,1}$	$\wedge$	$\neg sa_{1,2}^3 \vee sa_{1,1}^3$	$\wedge$	$\neg sa_{1,1}^3 \vee \neg pr_{1,1}^{3,1} \vee sa_{1,2}^1$
$\wedge$		$\wedge$		$\wedge$	$\neg sa_{1,2}^3 \vee \neg pr_{1,1}^{3,1} \vee sa_{1,2}^1$
(C3)	$pr_{1,1}^{1,3} \vee pr_{1,1}^{3,1}$	(C6)	$\neg sa_{2,3}^1 \vee \neg eb_1$	$\wedge$	
$\wedge$		$\wedge$	$\neg sa_{1,1}^2 \vee \neg eb_2$	(O)	$(eb_1, 1)$
(C4)	$eb_2 \vee sa_{1,2}^1$	$\wedge$	$\neg sa_{1,2}^3 \vee \neg eb_3$	$\wedge$	$(eb_2, 2)$
$\wedge$	$eb_2 \vee sa_{1,2}^3$	$\wedge$		$\wedge$	$(eb_3, 3)$

Figure 3: WPM formulation for exemplified problem.

	$sa_{1,1}^1 = 1$	$sa_{1,0}^2 = 1$	
$eb_1 = 0$	$sa_{1,2}^1 = 1$	$sa_{1,1}^2 = 0$	$pr_{1,2}^{1,1} = 1$
$eb_2 = 1$	$sa_{2,2}^1 = 1$	$sa_{1,1}^3 = 1$	$pr_{1,1}^{1,3} = 0$
$eb_3 = 1$	$sa_{2,3}^1 = 1$	$sa_{1,2}^3 = 0$	$pr_{1,1}^{3,1} = 1$

Figure 4: Assignment of all Boolean variables in the exemplified problem.

0 for $t > t'$, then we can determine that the start time of f_i^ℓ is $t\prime$. If $\mathrm{sa}_{i,t}^\ell$ is constantly true when t varies in $[\mathrm{EST}_i^\ell, \mathrm{LST}_i^\ell + 1]$, we can conclude that τ_ℓ is late. Note that it is impossible to designate $\mathrm{sa}_{i,t}^\ell$ as true while making $\mathrm{sa}_{i,t-1}^\ell$ false, since if f_i^ℓ starts at or after time t, f_i^ℓ must start at or after $t-1$. This constraint is interpreted by rule (C5) in our encoding. In addition, $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, the start time of f_i^ℓ should be at or after EST_i^ℓ. Compared with PM, WPM encodes this constraint in a more concise way by (C1). We postpone the discussion later in Section 4.2.

Aside from the inherent properties of a fragment's start time specified in (C1) and (C5), whether a fragment can start at a certain time is constrained by two other factors. The first is the execution order of the fragments, which should be explicitly pointed out in the following two situations: if two fragments belong to the same task, then they should be executed sequentially; if two fragments from different tasks may simultaneously occupy the processor, then either one can precede the other. The execution order of two fragments, f_i^k and f_j^ℓ, is characterized by Boolean variable $\mathrm{pr}_{i,j}^{k,\ell}$, and the constraints specified in the above two cases are interpreted by rules (C2) and (C3), respectively. If the execution order of the two fragments is determined, the relation of their start times should be specified. To be specific, if f_i^k precedes f_j^ℓ, then f_j^ℓ must start after f_i^k finishes. This constraint is guaranteed by rule (C7). Up to this point, all the Boolean variables and clauses introduced by PM and WPM are identical. That is, rules (C2), (C3), (C5), and (C7) are all consistent with those introduced in PM.

4.2. WPM Improvement. As mentioned in Section 4.1, $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, the start time of f_i^ℓ cannot be earlier than EST_i^ℓ. PM interprets this constraint by introducing one hard clause $\mathrm{sa}_{i,\mathrm{EST}_i^\ell}^\ell$ for $1 \le i \le q_\ell$; thus, a total of q_ℓ clauses are generated for τ_ℓ. In comparison, as described by (C1) in Section 3.2, WPM generates only one clause $\mathrm{sa}_{1,\mathrm{EST}_1^\ell}^\ell$ for each task $\tau_\ell \in \Gamma$ no matter how many fragments τ_ℓ contains. By combining (C1) with constraints (C2) and (C7), we can easily obtain $\mathrm{sa}_{i,\mathrm{EST}_i^\ell}^\ell = 1$ for $1 \le i \le q_\ell$, which is the same as that declared by PM. Therefore, the updated rule (C1) reduces the number of generated clauses while maintaining the correctness of the encoding.

For tasks with dependency relations, their execution sequence may be explicitly or implicitly specified. In particular, $\forall (\tau_k, \tau_\ell) \in R_{\mathrm{dp}}$, PM indistinguishably generates $\mathrm{pr}_{q_k,1}^{k,\ell}$ for the dependency relation $\tau_k \prec \tau_\ell$. In fact, if the due date of τ_k is no later than the EST of τ_ℓ, that is, $d_k \le \mathrm{EST}_1^\ell$, τ_ℓ naturally starts after τ_k is completed; thus the constraint that is explicitly imposed on the execution sequence is unnecessary and can be omitted safely. WPM formulation refined the encoding in (C4) by introducing $\mathrm{pr}_{q_k,1}^{k,\ell}$ to interpret the constraint that τ_k precedes τ_ℓ only when $d_k > \mathrm{EST}_1^\ell$; thus, redundant variables and clauses are eliminated.

Another factor that affects a fragment's start time is the completion time of a task. In general, if a task is expected to be completed by a particular time, then each fragment of the task should not start later than its LST. The relation of completion time and LST is encoded by PM and WPM in different ways. In PM, for each fragment $f_i^\ell (1 \le \ell \le n, 1 \le i \le q_\ell)$, a set of Boolean variables $\mathrm{eb}_{i,t}^\ell (\mathrm{ECT}_i^\ell - 1 \le t \le \mathrm{LCT}_i^\ell)$ are introduced to indicate whether f_i^ℓ finishes at or before time t. Then a set of hard clauses are generated to restrict the values of the eb variables.

(i) (H1) If f_i^ℓ ends by t, then it ends by time $t+1$:
$$\mathrm{eb}_{i,t}^\ell \longrightarrow \mathrm{eb}_{i,t+1}^\ell, \quad \left(\mathrm{ECT}_i^\ell - 1 \le t \le \mathrm{LCT}_i^\ell - 1\right), \tag{11}$$

(ii) (H2) if τ_ℓ is completed by its due date, then, $\forall f_i^\ell \in \tau_\ell$, f_i^ℓ should finish by LCT_i^ℓ:
$$\mathrm{eb}_{q_\ell,d_\ell}^\ell \longrightarrow \mathrm{eb}_{i,\mathrm{LCT}_i^\ell}^\ell, \quad (1 \le i \le q_\ell - 1), \tag{12}$$

(iii) (H3) if f_i^ℓ starts at or after time t, then it cannot end before time $t + c_i^\ell - 1$:
$$\mathrm{sa}_{i,t}^\ell \longrightarrow \mathrm{eb}_{i,t+c_i^\ell-1}^\ell, \quad \left(\mathrm{EST}_i^\ell \le t \le \mathrm{LST}_i^\ell + 1\right), \tag{13}$$

In fact, not all the variables and clauses in (H1) ~ (H3) are requisite. Given a task, instead of generating variables to characterize each fragment's completion time, it suffices to create only one variable to declare the task's due date and introduce one clause to clarify how the task's due date both restricts and is restricted by its start time. In particular, $\forall \tau_\ell \in \Gamma$, WPM introduces Boolean variable eb_ℓ to describe whether τ_ℓ can finish by its due date. Then a constraint is set to correlate the due date and the start time of τ_ℓ as follows: if τ_ℓ ends by d_ℓ, then $f_{q_\ell}^\ell$ must start at or before $\mathrm{LST}_{q_\ell}^\ell$. In other words, if $f_{q_\ell}^\ell$ fails to start at or before $\mathrm{LST}_{q_\ell}^\ell$, then it cannot end by d_ℓ. This constraint is encoded by just one clause described in rule (C6), as declared in Section 3.2:

$$\mathrm{eb}_\ell \longrightarrow \mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell. \tag{14}$$

Theorem 1. *Given task $\tau_\ell \in \Gamma$, if $\mathrm{eb}_{q_\ell,d_\ell}^\ell$ in PM (resp., eb_ℓ in WPM) is evaluated as true, then $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell$ for $1 \le i \le q_\ell$ in both PM and WPM are false.*

Proof. In PM, according to (H2), $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 1$ makes $\mathrm{eb}_{i,\mathrm{LCT}_i^\ell}^\ell = 1$ for $1 \le i \le q_\ell - 1$. This indicates that, $\forall f_i^\ell \in \tau_\ell$, f_i^ℓ should finish by LCT_i^ℓ. Combined with (H3), it can be inferred that $\mathrm{sa}_{i,\mathrm{LST}_{i+1}^\ell}^\ell = 0$ for $1 \le i \le q_\ell$.

In WPM, that $\mathrm{eb}_\ell = 1$ leads to $\mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell = 0$ by (C6). In addition, based on (C2) and (C7), it is clear that $\mathrm{sa}_{q_\ell-1,\mathrm{LST}_{q_\ell-1}^\ell+1}^\ell \longrightarrow \mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell$ holds. Under the condition where $\mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell = 0$, it can be immediately inferred that $\mathrm{sa}_{q_\ell-1,\mathrm{LST}_{q_\ell-1}^\ell+1}^\ell = 0$. By repeatedly substituting (C2) into (C7), we finally obtain $\mathrm{sa}_{i,\mathrm{LST}_{i+1}^\ell}^\ell = 0$ for $1 \le i \le q_\ell$. □

Theorem 2. *Given task $\tau_\ell \in \Gamma$, if $\exists f_i^\ell \in \tau_\ell$ that makes $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell$ be evaluated as true, then $\mathrm{eb}_{q_\ell,d_\ell}^\ell$ in PM (resp., eb_ℓ in WPM) is false.*

Proof. In PM, by (H3), $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell = 1$ makes $\mathrm{eb}_{i,\mathrm{LCT}_i^\ell}^\ell = 0$. Combined with (H2), it is obvious that $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 0$.

In WPM, we discuss the two following cases:

(1) If critical f_i^ℓ, which makes $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell = 1$, is the last fragment in τ_ℓ, that is, $i = q_\ell$, it is obvious that $\mathrm{eb}_\ell = 0$ by (C6).

(2) If $1 \le i \le q_\ell - 1$, according to (C2) and (C7), we have $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell \longrightarrow \mathrm{sa}_{i+1,\mathrm{LST}_{i+1}^\ell+1}^\ell$. Since $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell = 1$, it is clear that $\mathrm{sa}_{i+1,\mathrm{LST}_{i+1}^\ell+1}^\ell = 1$. The step of substituting (C2) into (C7) can be repeated until $i + 1 = q_\ell$. Finally, we have $\mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell = 1$. Then, by (C6), $\mathrm{eb}_\ell = 0$ is obtained.

Theorem 1 shows that if $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 1$ in PM (resp., $\mathrm{eb}_\ell = 1$), both PM and WPM guarantee that $\mathrm{sa}_{i,\mathrm{LST}_{i+1}^\ell}^\ell = 0$ for $1 \le i \le q_\ell$. Theorem 2 shows that when there exists a fragment $f_i^\ell \in \tau_\ell$ that leads to $\mathrm{sa}_{i,\mathrm{LST}_i^\ell+1}^\ell = 1$, both PM and WPM come to the same conclusion that τ_ℓ cannot finish by τ_ℓ, pointed out by $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 0$ and $\mathrm{eb}_\ell = 0$ in PM and WPM, respectively. Therefore, the eb variables introduced in PM and WPM have the same impact on the *sa* variables and vice versa. Note that, $\forall f_{q_\ell}^\ell \in \tau_\ell$, if $\mathrm{sa}_{q_\ell,\mathrm{LST}_{q_\ell}^\ell+1}^\ell = 0$, a WPM solver always prefers $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 1$ in PM (resp., $\mathrm{eb}_\ell = 1$ in WPM) to satisfy soft clause $\mathrm{eb}_{q_\ell,d_\ell}^\ell$ (resp., (eb_ℓ, w)). In other words, a WPM solver never prefers $\mathrm{eb}_{q_\ell,d_\ell}^\ell = 0$ in PM (resp., $\mathrm{eb}_\ell = 0$ in WPM) as long as all the hard clauses input to the solver are satisfied. □

Example 1. This example shows the differences between the PM and WPM formulations when dealing with the completion time. Assume that $\tau_1 = (0, 2, 3, 1)$ is a task in Γ, which contains two fragments $\langle f_1^1, f_2^1 \rangle$. Each fragment has execution time 1. Table 3 shows the Boolean variables and the hard clauses introduced by PM and WPM to characterize the fragments' completion times.

First, we can determine that $\mathrm{ECT}_1^1 = 1$, $\mathrm{LCT}_1^1 = 2$, $\mathrm{ECT}_2^1 = 2$, and $\mathrm{LCT}_2^1 = 3$. To encode τ_1, PM introduces a series of Boolean variables, $\{\mathrm{eb}_{1,0}^1, \mathrm{eb}_{1,1}^1, \mathrm{eb}_{1,2}^1\}$ and $\{\mathrm{eb}_{2,1}^1, \mathrm{eb}_{2,2}^1, \mathrm{eb}_{2,3}^1\}$ for f_1^1 and f_2^1, respectively, representing whether each fragment is completed by time point t, where $\mathrm{ECT}_i^1 - 1 \le t \le \mathrm{LCT}_i^1$ for $1 \le i \le 2$. The relation of $\mathrm{eb}_{i,t}^1$ and $\mathrm{eb}_{i,t+1}^1$ is specified by rule (H1). To make τ_1 finish by its due date, f_1^1 should be completed no later than time 2. This is guaranteed by rule (H2). Finally, the relation between a fragment's start and completion times is indicated by rule (H3). Therefore, to constrain the completion time of each fragment, six Boolean variables and eleven hard clauses are generated by PM. By contrast, WPM introduces only one variable eb_1 to indicate whether τ_1 is completed by its due date. Then a hard clause is generated to constrain τ_1's due date and its last fragment's start time. In this way, both the Boolean variables and clauses generated by the WPM formulation are reduced.

The experimental comparison of these two encodings is demonstrated in Subsection 5.3.

Table 3: Encodings related to *eb* variables in PM and WPM formulations.

		PM [38]	WPM
Variables		$\mathrm{eb}_{1,0}^1, \mathrm{eb}_{1,1}^1, eb_{1,2}^1, eb_{2,1}^1, eb_{2,2}^1, eb_{2,3}^1$	eb_1
Clauses	(H1)	$\mathrm{eb}_{1,0}^1 \vee \mathrm{eb}_{1,1}^1$, $\mathrm{eb}_{1,1}^1 \vee \mathrm{eb}_{1,2}^1$, $\mathrm{eb}_{2,1}^1 \vee \mathrm{eb}_{2,2}^1$, $\mathrm{eb}_{2,2}^1 \vee \mathrm{eb}_{2,3}^1$	
	(H2)	$\mathrm{eb}_{2,3}^1 \vee \mathrm{eb}_{1,2}^1$	$\mathrm{eb}_1 \vee \mathrm{sa}_{2,3}^1$
	(H3)	$\mathrm{sa}_{1,0}^1 \vee \mathrm{eb}_{1,0}^1$, $\mathrm{sa}_{1,1}^1 \vee \mathrm{eb}_{1,1}^1$, $\mathrm{sa}_{1,2}^1 \vee \mathrm{eb}_{1,2}^1$ $\mathrm{sa}_{2,1}^1 \vee \mathrm{eb}_{2,1}^1$, $\mathrm{sa}_{2,2}^1 \vee \mathrm{eb}_{2,2}^1$, $\mathrm{sa}_{2,3}^1 \vee \mathrm{eb}_{2,3}^1$	

5. Experiments

In this section, we scrutinize the performance comparisons of the presented WPM formulation and two satisfiability-based formulations on a set of randomly generated problems. The experimental design is described in Section 5.1, followed by the evaluation of WPM, SMT [39], and PM formulations [38] in Section 5.2 and Section 5.3. All tests were conducted on a 3.4 GHz Intel E3-1230 processor with 8 GB RAM. The selected solver to evaluate WPM and PM is QMaxSAT [40], which is a SAT-based solver using the CNF encodings of cardinality constraints. The solver for SMT is Z3 [41], which is a high-performance theorem prover chosen by the previous SMT formulations [33, 39].

5.1. Experimental Design. The procedure for generating test instances is similar to [33, 38, 39], with a variety of parameter values. Tasks' release times are randomly generated following a discrete uniform distribution with an arriving rate λ, which represents the number of tasks that arrive during 100 time units. Clearly, a larger λ indicates that more tasks arrive in the system during a specific period of time, thus causing heavier overload. For each task τ_ℓ, the execution time c_ℓ and the number of fragments in τ_ℓ (denoted by q_ℓ) are also randomly generated according to a discrete uniform distribution. The value of deadline d_ℓ is calculated by the formula $d_\ell = r_\ell + sf_\ell \cdot c_\ell$, where sf_ℓ is a slack factor that reflects the tightness of the due date. Weight w_ℓ of task τ_ℓ is randomly chosen from 1 to n in Section 5.2 and is taken as constant 1 in Section 5.3. The number of rule pairs with dependency relations is set 10% to the total number of tasks.

We study the factors that may affect the performance of the satisfiability-based formulations. These factors include the number of tasks n, the task density (determined by λ), the execution time c_ℓ, the number of fragments q_ℓ, and the slack factor sf_ℓ. In the rest of this section, we run different types of experiments to test the possible changes on these parameters. Table 4 provides a summary of the experimental design, where DU (i, j) means the discrete uniform distribution over integer interval $[i, j]$. For each individual scenario, 100 problem instances are generated. For each instance and solver, we set a time limit of 300 seconds. If the solver fails to output the optimal result of an instance within the time limit, we terminate the procedure and move to the next problem instance.

5.2. Comparison on Weighted Case. This subsection evaluates the behavior of WPM and SMT formulations [39] for maximizing the total weight value of on-time tasks. The main metrics for comparison include (1) the proportion of instances solved within the given time limit and (2) the average computation time spent on solved instances. The performances of these two formulations in different types of scenarios are shown in Figure 5. Each data point is the average computation time of the solved problem instances. A number with an arrow in the figures denotes the proportion of instances solved within the time limit and is omitted if the solver addresses all the 100 instances. When the proportion of the solved instances drops to zero, the corresponding curve is omitted.

As shown in Figure 5, both formulations perform worse with the increasing values of all parameters. Nevertheless, in all the varied scenarios, WPM substantially outperforms SMT in terms of both the average computation time and the proportion of successfully solved instances.

5.3. Comparison on Unweighted Case. A Partial MaxSAT (PM) formulation for optimal scheduling on a single machine was recently presented [38]. The scheduling objective tackled by PM differs from our WPM formulation in that PM aims to maximize the total number of on-time tasks, while WPM tries to maximize the total weight of the on-time tasks. When all the tasks have equal weights, these two objectives become identical, and thus formulations on these two objectives are directly comparable. In this subsection, we compare the SMT, PM, and WPM formulations on the objective of maximizing the number of on-time tasks.

Table 4: Experimental design.

Scenario	n	λ	c_ℓ	q_ℓ	sf_ℓ
S1	$50, 100, \ldots, 300$	10	DU(1, 30)	DU(1, 3)	DU(1, 4)
S2	100	$5, 10, \ldots, 25$	DU(1, 30)	DU(1, 3)	DU(1, 4)
S3	100	10	DU(1, α) $\alpha = 10, 20, \ldots, 100$	DU(1, 3)	DU(1, 4)
S4	100	10	DU(1, 30)	DU(1, β) $\beta = 2, 4, \ldots, 12$	DU(1, 4)
S5	100	10	DU(1, 30)	DU(1, 3)	DU(1, γ) $\gamma = 1, 2, \ldots, 7$

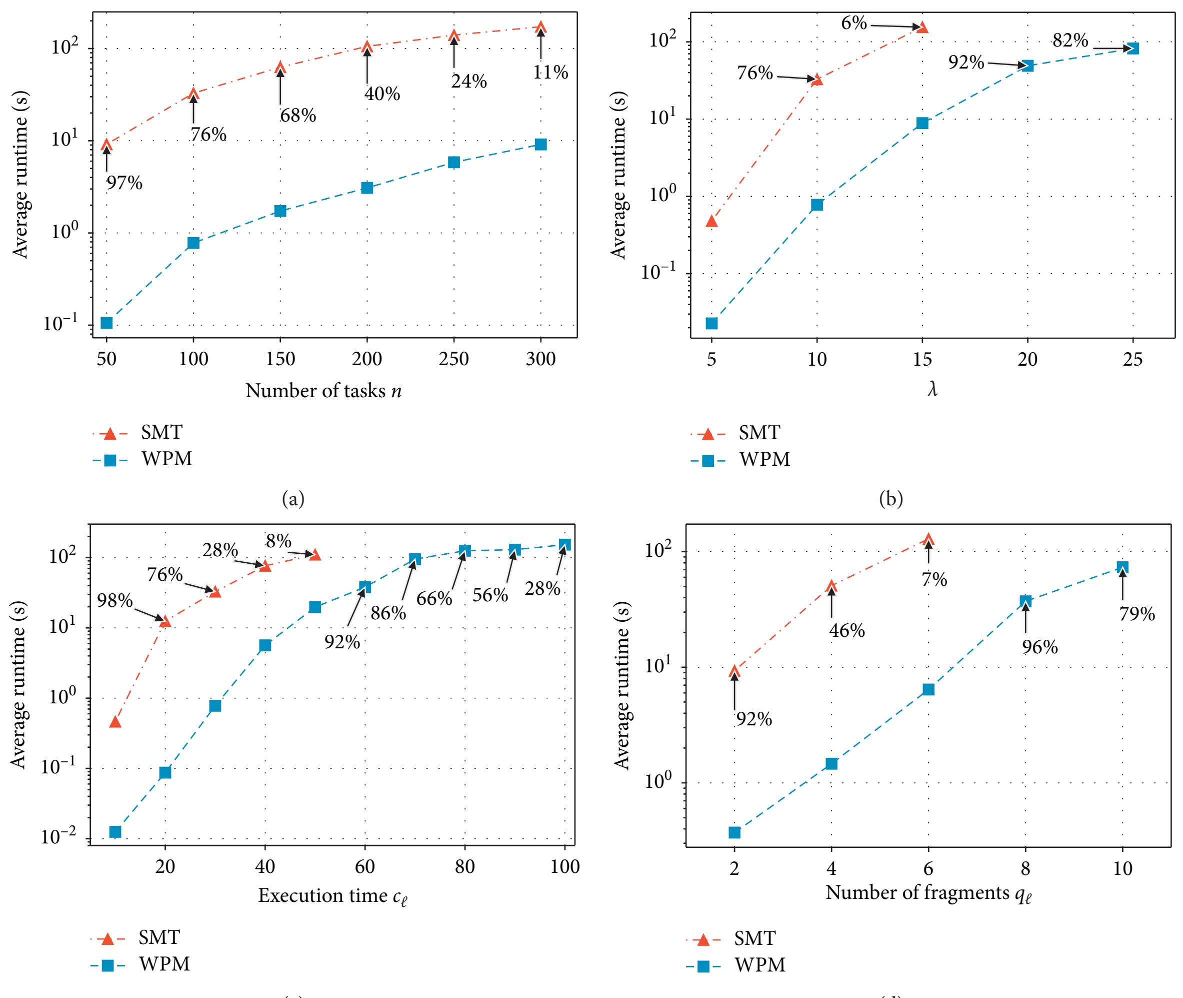
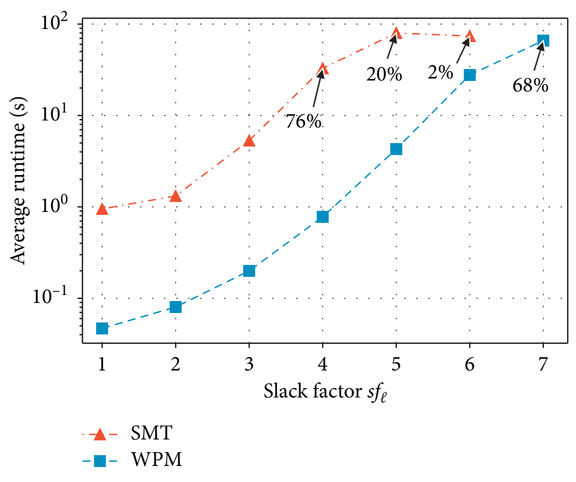


Figure 5: Continued.



(e)

FIGURE 5: Performance comparison of WPM and SMT for weighted cases in scenarios 1–5. (a) S1: varied number of tasks n; (b) S2: varied task arriving rate λ; (c) S3: varied execution time c_ℓ; (d) S4: varied number of fragments q_ℓ; (e) S5: varied slack factor sf_ℓ.

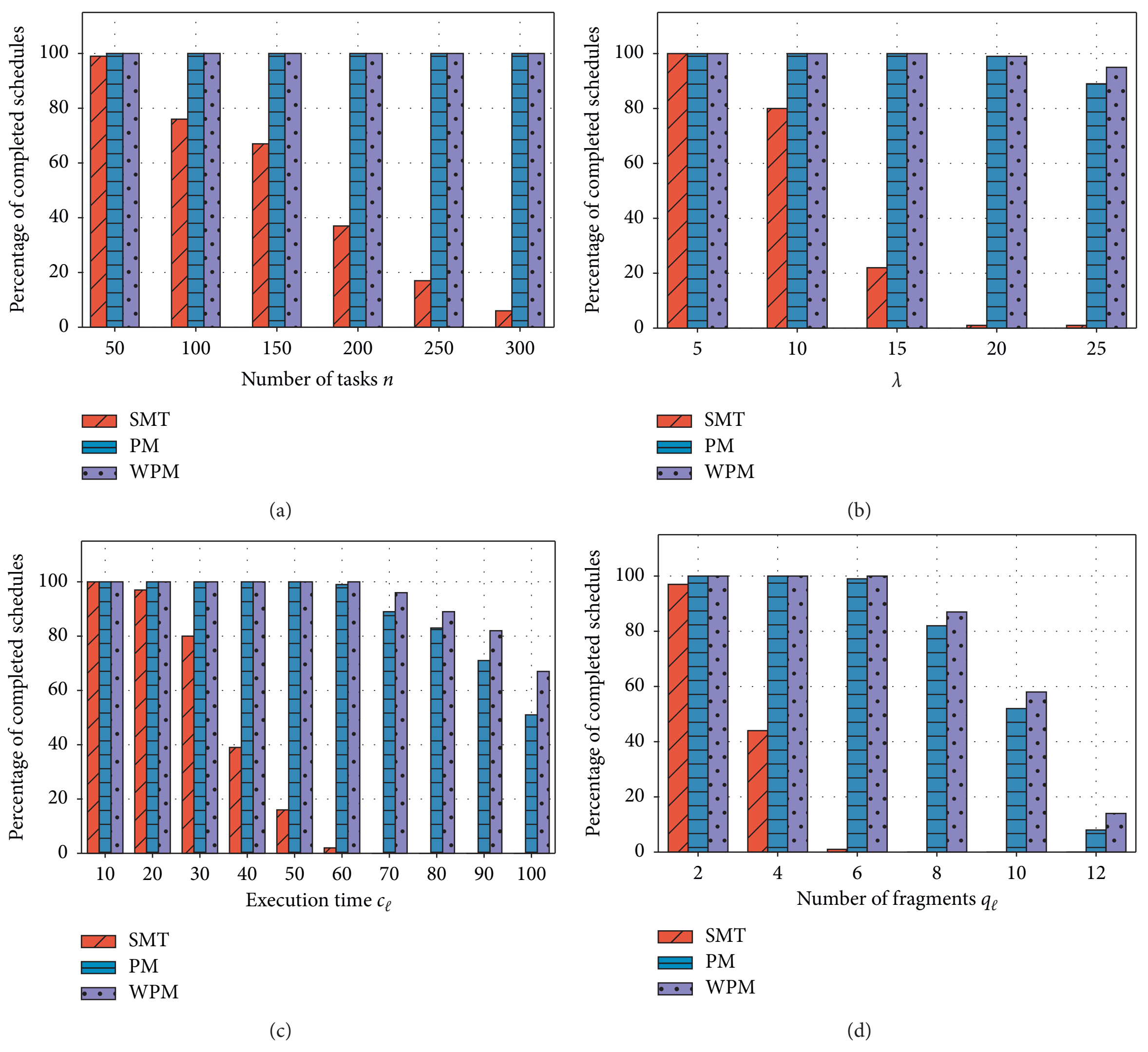
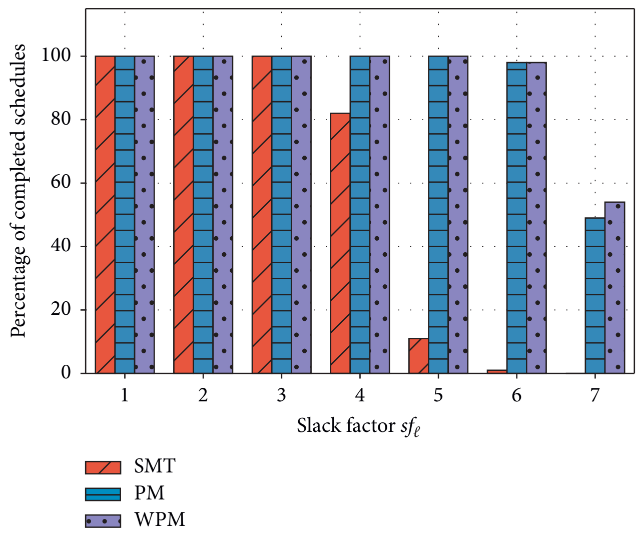


FIGURE 6: Continued.



(e)

Figure 6: Completion percentage of formulations for unweighted cases in scenarios 1–5. (a) S1: varied number of tasks n; (b) S2: varied task arriving rate λ; (c) S3: varied execution time c_ℓ; (d) S4: varied number of fragments q_ℓ; (e) Scenario 5: varied slack factor sf_ℓ.

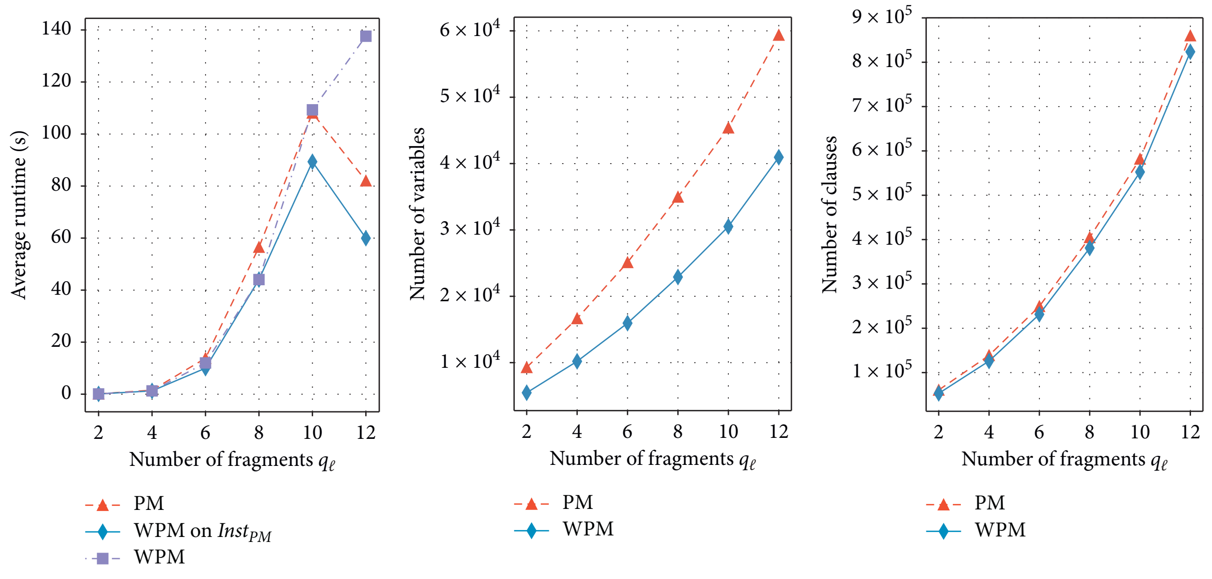


Figure 7: Comparison statistics of PM and WPM for unweighted cases in Scenario 4. (a) Average runtime (s) (b) Number of variables. (c) Number of clauses.

The overall results with varied n, λ, c_ℓ, q_ℓ, and sf_ℓ are shown in Figure 6. MaxSAT encodings (both PM and WPM) solved substantially more instances than SMT within the time limit in all cases. As to the performance comparison between PM and WPM, the completion percentage of WPM is always no less than that of PM no matter how the experimental parameters change. To be specific, with the number of tasks n increasing from 50 to 300, both PM and WPM solved all the instances within the time limit (Figure 6(a)). In comparison, as any of the parameters λ, c_ℓ, q_ℓ, and sf_ℓ increases, WPM formulation gradually surpasses PM with more completed schedules (Figures 6(b)–6(e)).

To compare the performances of PM and WPM more clearly, we provide more criteria for evaluation. The first criterion is the average runtime of PM and WPM formulations on their respective completed schedules. Note that when PM and WPM solve different numbers of instances within the time limit, the average runtime on solved instances may not clearly reflect the efficiency of these two formulations. For example, when $q_\ell \sim \mathrm{DU}(1,12)$, WPM

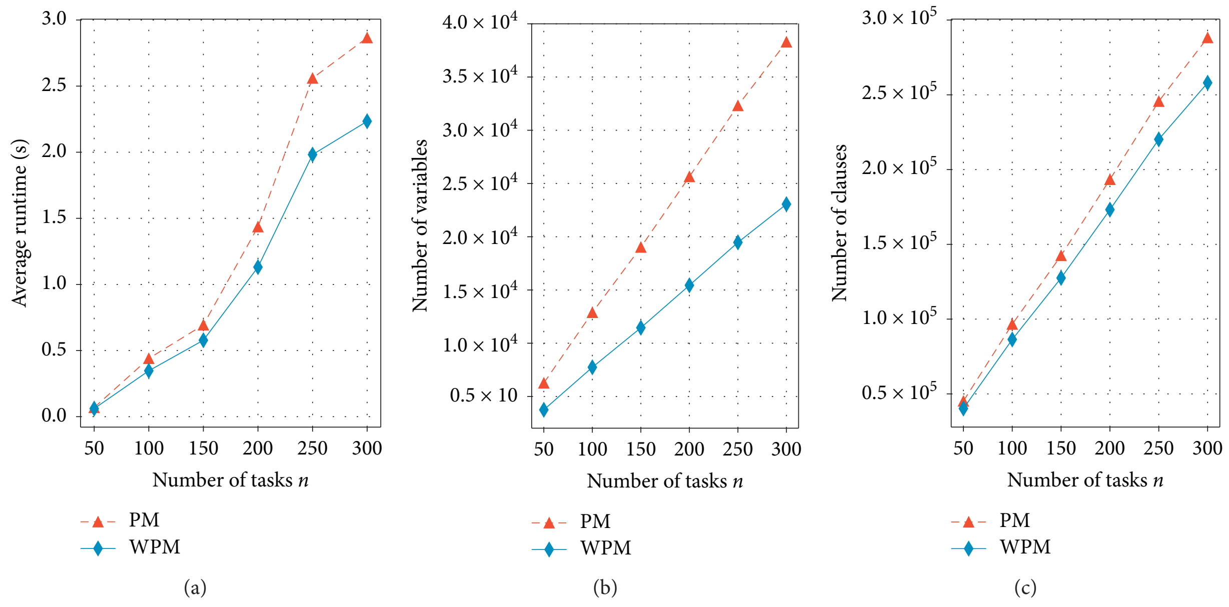


Figure 8: Comparison statistics of PM and WPM for unweighted cases in Scenario 1. (a) Average runtime (s). (b) Number of variables. (c) Number of clauses.

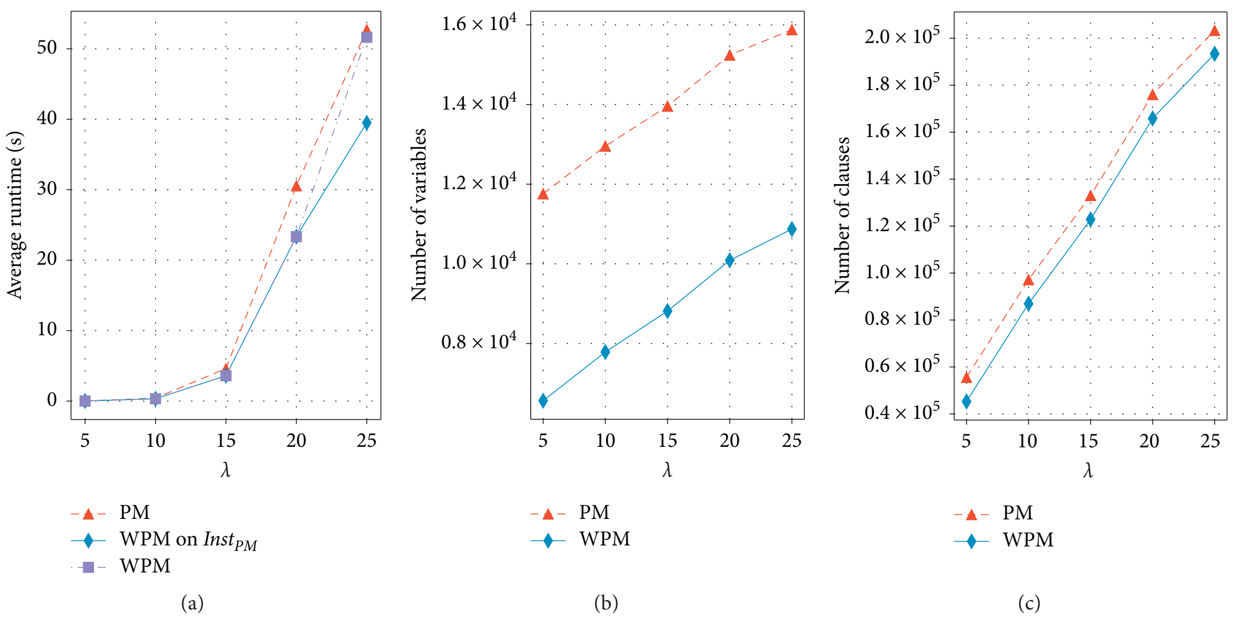


Figure 9: Comparison statistics of PM and WPM for unweighted cases in Scenario 2. (a) Average runtime (s). (b) Number of variables. (c) Number of clauses.

solved more instances than PM (as shown in Figure 6(d)) and meanwhile consumed longer time on solved instances (as shown in Figure 7(a)). In this case, the average runtime is on longer the proper metric to indicate the performance. On the other hand, we notice that, in all the tested scenarios, the instances that were solved by PM could also be addressed by WPM. Therefore, we provide a secondary criterion on runtime, that is, the average runtime of WPM, in solving the same set of instances as PM solved. Furthermore, since the computation time of MaxSAT-based methods is closely related to the numbers of variables and clauses, we collect such information for reference. The comparison results are illustrated in Figures 8–11 . Criteria for evaluating PM and WPM are summarized as follows:

(i) The average runtime of MaxSAT formulations on their respective completed schedules. The corresponding information of PM and WPM is depicted

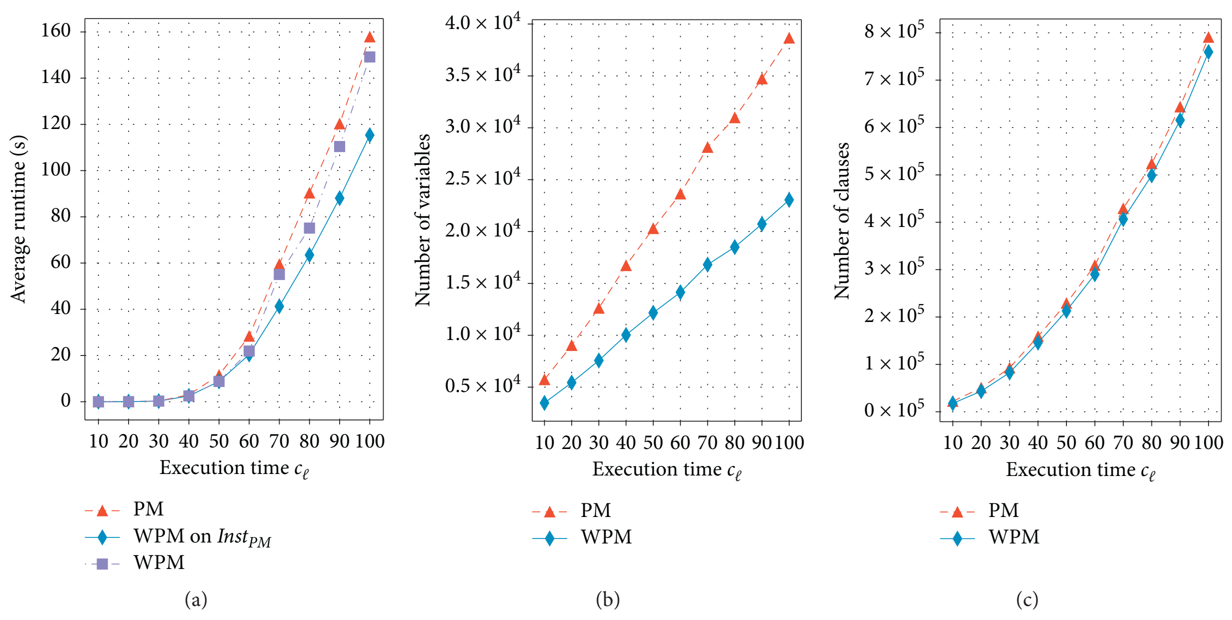


Figure 10: Comparison statistics of PM and WPM for unweighted cases in Scenario 3. (a) Average runtime (s). (b) Number of variables. (c) Number of clauses.

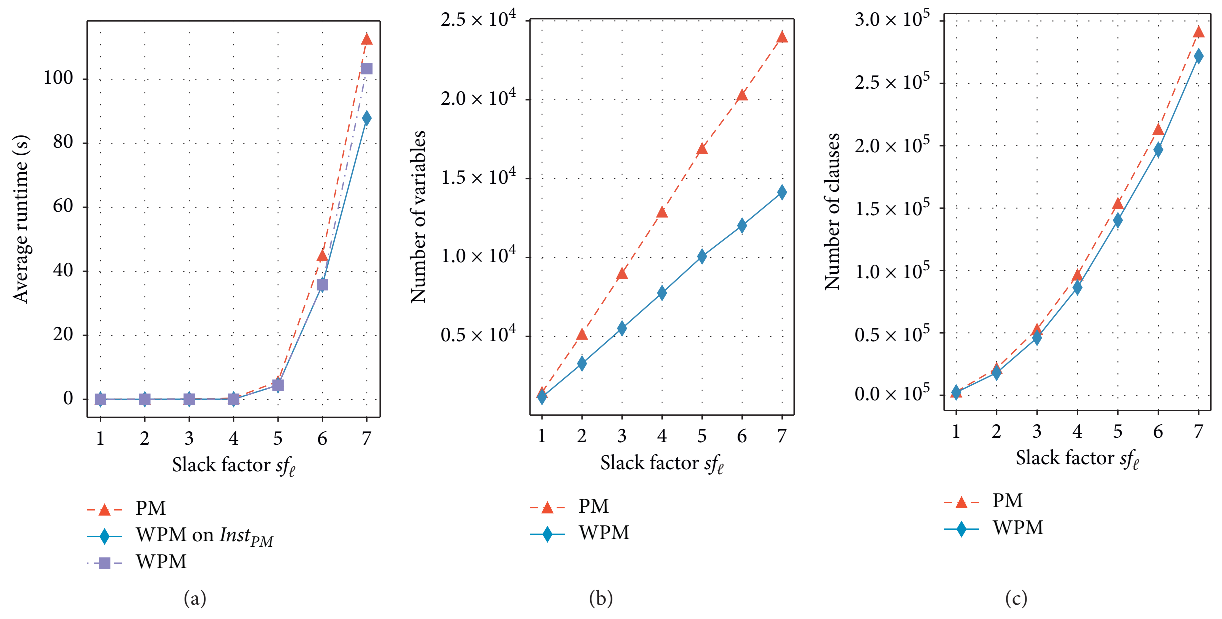


Figure 11: Comparison statistics of PM and WPM for unweighted cases in Scenario 5. (a) Average runtime (s). (b) Number of variables. (c) Number of clauses.

in Figures 8(a), 9(a), 10(a), 7(a), and 11(a) labeled "PM" and "WPM," respectively.

(ii) The average runtime of WPM in solving the same set of instances as PM solved. The corresponding information is depicted with label "WPM on Inst_{PM}." If the percentages of schedules completed by PM and WPM are the same, then "WPM on Inst_{PM}" is omitted (e.g., Figure 7(a)).

(iii) The average number of variables generated by MaxSAT formulations.

(iv) The average number of clauses generated by Max-SAT formulations.

Overall, the WPM formulation is more efficient than PM in both runtime criteria. The only exception appears in Figure 7(a), where the average runtime of WPM and PM on solved instances when $q_\ell \sim \text{DU}(1, 12)$ is around 140 and 80 seconds, respectively. Although the average computation time taken by WPM is longer than that by PM, Figure 6(d) shows that WPM managed to solve 14 instances, while PM solved only 8 within the time limit. In this case, we resort to the secondary criterion on runtime. As shown in Figure 7(a), WPM consumed merely 60 seconds to solve all the instances completed by PM. Thus, it is correct to say that WPM can make a considerable improvement to the solving time. In addition, WPM achieved more compact encodings by significantly reducing the number of Boolean variables and slightly decreasing the number of clauses. In general, the computation time increases as the numbers of variables and clauses grow. This in turn explains why WPM achieves higher efficiency in our experiment.

6. Adaption for Parallel-Machine Scheduling Problem

While the WPM formulation is designed for single-machine scheduling, the encoding can be extended to parallel identical machines with minor changes. This section shows how to apply the WPM formulation to parallel identical machine scheduling problems.

The parallel-machine scheduling involves processing n tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ on m identical machines $\mathscr{M} = \{M_1, M_2, \ldots, M_m\}$. Each machine can handle only one task at a time, and each task cannot be processed in parallel. Other settings of the problem are consistent with the single-machine scheduling problem described in Section 2. Given a set of tasks Γ and that of machines $\mathscr{M}$, the optimization problem is not only to find a schedule that gives the start execution time of each fragment $f_i^\ell \in \tau_\ell$ in Γ but also a mapping for f_i^ℓ to a machine in $\mathscr{M}$.

To indicate on which machine a fragment is executed, $\forall \tau_\ell \in \Gamma$, $\forall f_i^\ell \in \tau_\ell$, and $\forall M_u \in \mathscr{M}$, we extend the single-machine scheduling formulation by introducing one more Boolean variable $b_{i,u}^\ell$. $b_{i,u}^\ell = 1$ if f_i^ℓ is executed by M_u and $b_{i,u}^\ell = 0$ otherwise. Correspondingly, rule (C3) in Section 3.2 is replaced with the two following additional constraints:

(i) (R1) $\forall \tau_\ell \in \Gamma$ and $\forall f_i^\ell \in \tau_\ell$, f_i^ℓ is allocated and executed on a single machine:

$$\bigvee_{u=1,\ldots,m} b_{i,u}^\ell, \quad (1 \le \ell \le n, 1 \le i \le q_\ell). \tag{15}$$

(ii) (R2) $\forall \tau_k, \tau_\ell \in \Gamma$, $\forall f_i^k \in \tau_k$, and $\forall f_j^\ell \in \tau_\ell$, if $k \neq l$, $\tau_k \nprec \tau_\ell$, $\tau_\ell \nprec \tau_k$, $\mathrm{EST}_i^k < \mathrm{LCT}_j^\ell$ and $\mathrm{EST}_j^\ell < \mathrm{LCT}_i^k$, and then f_i^k and f_j^ℓ may require the processor at the same time. In this condition, f_i^k precedes f_j^ℓ or f_j^ℓ precedes f_i^k:

$$b_{i,u}^k \wedge b_{i,u}^\ell \longrightarrow \mathrm{pr}_{i,j}^{k,\ell} \vee \mathrm{pr}_{j,i}^{\ell,k}. \tag{16}$$

We refer to the set of hard clauses introduced in (R1) and (R2) as $\mathscr{R}$ and rules in (C1) ~ (C7) excluding (C3) as $\mathscr{C}'$. Conjunct with the set of weighted soft clauses $\mathscr{O}$ defined in Section 3.2, the problem is $\{\mathscr{C}', \mathscr{R}, \mathscr{O}\}$, which tries to find an assignment of variables to satisfy all the hard clauses in $\mathscr{C}' \cup \mathscr{R}$ and to maximize the sum of the weights of the satisfied soft clauses in $\mathscr{O}$. Thereby, the parallel identical machine scheduling problem of maximizing the total weight of on-time tasks can be optimally solved with any off-the-shelf weighted Partial MaxSAT solver.

7. Conclusions and Perspective

We concentrated on a Weight Partial MaxSAT (WPM) formulation for optimal scheduling in overloaded situations. The aim is to maximize the (weighted) number of on-time tasks. Motivated by the WPM feature that distinguishes between hard clauses and weighted soft clauses, we encoded the properties of tasks as hard clauses and the goal of completing tasks on time as a set of weighted soft clauses. Then an off-the-shelf WPM solver was employed to satisfy all the hard clauses and maximize the total weight of the satisfied soft clauses. From the output of the WPM solver, the optimal schedule can be obtained.

The WPM formulation's performance was compared with that of the recent SMT and Partial MaxSAT (PM) formulations. First, we compared the performance of WPM and SMT, demonstrating that WPM is significantly superior to SMT. Then, we considered a special case where WPM and PM formulations are directly comparable. Results indicate that the WPM formulation achieves more compact encoding and higher efficiency than PM. Finally, we applied the WPM formulation to parallel identical machines with very little modification, highlighting the flexibility and scalability of the encoding.

The restricted preemptive scheduling model considered in this paper is built upon earlier works [33, 38, 39], where each task is split into several nonpreemptive fragments, and preemptions can only take place at the fragments' boundaries. This preemptive model with fixed points is too rigid to apply in many practical systems. Our future work is to design exact methods to adapt to variations of preemptive models. One example is to execute tasks continuously without interruption for at least a certain portion of time [42–44]. Instead of splitting tasks into fragments in advance, this restricted preemptive model only guarantees the minimum "granularity" of preemption without predefined subtasks and thus is more suitable for real-world applications.

Data Availability

All the datasets are available from the corresponding author upon request.

Disclosure

This paper is an extended version of the authors' paper published in Pacific Rim International Conference on Artificial Intelligence, Yanuca Island, Fiji, 08, 2019 (Springer).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (Grant no. 61806171) and JSPS KAKENHI (Grants nos. JP17K00307 and JP19H04175).

References

[1] C. E. Aguero, N. Koenig, I. Chen et al., "Inside the virtual robotics challenge: simulating real-time robotic disaster response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 494–506, 2015.

[2] S. B. Er and R. E. Smith, *Method and Apparatus for Monitoring and Displaying Lead Impedance in Real-Time for an Implantable Medical Device: US*, Patent US5891179 A[P], 1999.

[3] D. P. Xenos, M. Cicciotti, G. M. Kopanos et al., "Optimization of a network of compressors in parallel: real time optimization (rto) of compressors in chemical plants - an industrial case study," *Applied Energy*, vol. 144, no. 5, pp. 51–63, 2015.

[4] G. D. Baulier, S. M. Blott, B. L. Branch et al., *Real-time Event Processing System for Telecommunications and Other Applications*, Patent US6496831 B1[P], 2002.

[5] L. Abeni, G. Buttazzo, S. Superiore, and S. Anna, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[6] S. K Baruah and J. R. Haritsa, "Scheduling for overload in real-time systems," *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1034–1039, 1997.

[7] S. K. Baruah, J. R. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions," in *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS '94)*, pp. 228–236, San Juan, PR, USA, December 1994.

[8] A. Malik, C. Walker, M. OSullivan, and S. Oliver, "Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay," *Computers and Operations Research*, vol. 89, pp. 113–126, 2018.

[9] J. Michael Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management Science*, vol. 15, no. 1, pp. 102–109, 1968.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, no. 1, pp. 287–326, 1979.

[11] M. Richard, *Karp. Reducibility Among Combinatorial Problems*, pp. 85–103, Plenum Press, New York, NY, USA, 1972.

[12] E. L. Lawler and J. M. Moore, "A functional equation and its application to the characterization of gamma distributions," *Management Science*, vol. 16, no. 1, pp. 77–84, 1969.

[13] M. R Garey and D. S. Johnson, "Scheduling tasks with nonuniform deadlines on two processors," *Journal of the Acm*, vol. 23, no. 3, pp. 461–467, 1976.

[14] J. K. Lenstra and A. H. G. Rinnooy Kan, "Complexity results for scheduling chains on a single machine," *European Journal of Operational Research*, vol. 4, no. 4, pp. 270–275, 1980.

[15] E. L. Lawler, "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs," *Annals of Operations Research*, vol. 26, no. 1, pp. 125–133, 1990.

[16] D. Nikos and M. Ross: Single Machine Problems. http://www2.informatik.uni-osnabrueck.de/knust/class/dateien/classes/ein_ma/ein_ma/.

[17] M. Sterna, *A Survey of Scheduling Problems with Late Work Criteria*, Omega, Bienne, Switzerland, 2011.

[18] X. Chu and J. Tao, "A competitive online algorithm for minimizing total weighted completion time on uniform machines," *Mathematical Problems in Engineering*, vol. 2020, Article ID 7527862, 9 pages, 2020.

[19] Y. Li, E. Fadda, D. Manerba, R. Tadei, and T. Olivier, "Reinforcement learning algorithms for online single-machine scheduling," in *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, FedCSIS 2020*, pp. 277–283, Sofia, Bulgaria, September 2020.

[20] W. Li and J. Yuan, "Single-machine online scheduling of jobs with non-delayed processing constraint," *Journal of Combinatorial Optimization*, vol. 41, no. 4, pp. 830–843, 2021.

[21] B. Goldengorin and V. Romanuke, "Online heuristic for the preemptive single machine scheduling problem to minimize the total weighted tardiness," *Computers & Industrial Engineering*, vol. 155, Article ID 107090, 2021.

[22] J. Y. Lee, "A genetic algorithm for a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times," *Mathematical Problems in Engineering*, vol. 2020, Article ID 8833645, 13 pages, 2020.

[23] Q. Jiang, X. Liao, R. Zhang, and Q. Lin, "Energy-saving production scheduling in a single-machine manufacturing system by improved particle swarm optimization," *Mathematical Problems in Engineering*, vol. 2020, Article ID 8870917, 16 pages, 2020.

[24] H. Wei, S. Li, H. Quan et al., "Unified multi-objective genetic algorithm for energy efficient job shop scheduling," *IEEE Access*, vol. 9, pp. 54542–54557, 2021.

[25] O. A. Muminu and A. O Adewumi, "A survey of single machine scheduling to minimize weighted number of tardy jobs," *Journal of Industrial & Management Optimization*, vol. 10, no. 1, pp. 219–241, 2013.

[26] S. Ourari, C. Briand, and B. Bouzouiac, "A MIP approach for the minimization of the number of late jobs in single machine scheduling," *Journal of Mathematical Modelling and Algorithms*, vol. 1, pp. 1–15, 2009.

[27] S. Venugopalan and S. Oliver, "ILP formulations for optimal task scheduling with communication delays on parallel systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 142–151, 2015.

[28] K. Ying, C. Cheng, S. Lin, and C. Hung, "Comparative analysis of mixed integer programming formulations for single-machine and parallel-machine scheduling problems," *IEEE Access*, vol. 7, pp. 152998–153011, 2019.

[29] H.-C. Hung, M. Bertrand, T. Lin, M. E. Posner, and J.-M. Wei, "Preemptive parallel-machine scheduling problem of maximizing the number of on-time jobs," *Journal of Scheduling*, vol. 22, no. 4, pp. 413–431, 2019.

[30] Z. Yang, H. S. Xiao, R. Guan, Y. Yang, and H. L. Ji, "Task scheduling for multiunit parallel test using mixed-integer linear programming," *Mathematical Problems in Engineering*, vol. 2021, Article ID 3785452, 13 pages, 2021.

[31] M. A. Qamhan, A. A. Qamhan, I. M. Al-Harkan, and Y. A. Alotaibi, "Mathematical modeling and discrete firefly algorithm to optimize scheduling problem with release date, sequence-dependent setup time, and periodic maintenance," *Mathematical Problems in Engineering*, vol. 2019, Article ID 8028759, 16 pages, 2019.

[32] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using SMT solver," in *Proceedings of the IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/distributed Computing*, pp. 189–194, IEEE, Shanghai, China, June 2016.

[33] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "SMT-based scheduling for overloaded real-time systems," *IEICE Transactions on Informations and Systems*, vol. E100-D, no. 5, pp. 1055–1066, 2017.

[34] T. Qi, L.-H. Zhu, J. Lian, L. Zhou, and J.-B. Wei, "An efficient multi-functional duplication-based scheduling framework for multiprocessor systems," *The Journal of Supercomputing*, vol. 76, no. 11, pp. 9142–9167, 2020.
[35] J. M. Crawford and A. B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," in *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, pp. 1092–1097, American Association for Artificial Intelligence, Menlo Park, CA, USA, August 1994.
[36] M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa, "Solving open job-shop scheduling problems by sat encoding," *IEICE Transactions on Informations and Systems*, vol. E93-D, no. 8, pp. 2316–2318, 2010.
[37] W. Liu, Z. Gu, X. Jiang, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1382–1389, 2011.
[38] X. Liao, H. Zhang, M. Koshimura, R. Huang, and W. Yu, "Maximum satisfiability formulation for optimal scheduling in overloaded real-time systems," in *Proceedings of the PRICAI 2019: Trends in Artificial Intelligence—16th Pacific Rim International Conference on Artificial Intelligence*, pp. 618–631, Springer, Yanuca Island, Fiji, August 2019.
[39] S. Wang, X. Liao, M. Wang, L. Chang, H. Yang, and T. Wang, "An improved smt-based scheduling for overloaded real-time systems," *Engineering Letters*, vol. 28, no. 1, pp. 112–122, 2020.
[40] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "Qmaxsat: a partial max-sat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012.
[41] L. De Moura and N. Bjørner, "Z3: an efficient SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, Berlin, Germany, 2008.
[42] K. Ecker and R. Hirschberg, "Task scheduling with restricted preemptions," in *Arndt Bode, Mike Reeve, and Gottfried Wolf*, pp. 464–475, Springer, Berlin, Germany, 1993.
[43] T. Barański, "Task scheduling with restricted preemptions," in *Proceedings of the 2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 231–238, IEEE, Szczecin, Poland, 2011.
[44] K. Pieńkosz and A. Prus, "Task scheduling with restricted preemptions on two parallel processors," in *Proceedings of the 2015 20th International Conference On Methods and Models in Automation and Robotics (MMAR)*, August 2015.