BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **25**, No 3
Sofia • 2025 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2025-0025

A Compact SAT Encoding for Non-Preemptive Task Scheduling on Multiple Identical Resources

Tuyen Van Kieu, Khanh Van To

Faculty of Information Technology, University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam
E-mails: tuyenkv@vnu.edu.vn khanhtv@vnu.edu.vn

Abstract**:** *This paper presents an efficient SAT-solving approach for addressing the NP-hard problem of non-preemptive task scheduling on multiple identical resources. This problem is relevant to various application domains, including automotive, avionics, and industrial automation where tasks compete for shared resources. The proposed approach, called CSE, incorporates several novel optimizations, including a Block encoding technique for efficient continuity constraint representation and specialized symmetry-breaking constraints to prune the search space. We evaluate the performance of CSE compared to state-of-the-art SAT encoding schemes and leading optimization solvers like Google OR-Tools, IBM CPLEX, and Gurobi through extensive experiments across diverse datasets. Our method achieves substantial reductions in solving time and exhibits superior scalability for large problem instances.*

Keywords**:** *Non-preemptive scheduling, Identical resources, SAT encoding, SAT solving, Symmetry-breaking.*

1. Introduction

Real-time systems in safety-critical applications require efficient scheduling algorithms to ensure tasks are completed within deadlines. Unlike preemptive scheduling [1], this paper addresses non-preemptive task scheduling on multiple identical resources, an NP-hard problem [2, 3] where tasks must execute continuously until completion once started.

This scheduling problem appears in diverse applications, including home appliance scheduling [4], meeting room allocation [5, 6], classroom scheduling [7], timetable generation [8], faculty-course assignment [9], smart parking systems [10, 11], and allocation of platforms to trains [12, 13]. These scenarios involve aperiodic tasks with known release times, execution durations, and deadlines.

We propose a novel Compact SAT Encoding (CSE) scheme with three key contributions:

1. CSE scheme with refined constraint representations for improved efficiency.

2. Block encoding technique (in Section 4) that achieves linear clause complexity for task execution continuity constraints, improving scalability over traditional quadratic approaches.

3. Specialized optimization and symmetry-breaking constraints (in Section 5) that effectively prune the search space while maintaining completeness.

Experimental evaluation on datasets with up to 200 tasks and 200 resources demonstrates that CSE significantly outperforms the state-of-the-art ES3 encoding [14] and leading optimization solvers (Google OR-Tools (**https://developers.google.com/optimization**), IBM CPLEX (**www.ibm.com/products/ilog-cplex-optimization-studio**), and Gurobi (**www.gurobi.com**)), achieving substantial reductions in solving time and superior scalability.

The paper is organized as follows: Section 2 reviews related work; Section 3 presents the CSE scheme; Section 4 details Block encoding; Section 5 describes symmetry breaking constraints; Section 6 presents experimental results; and Section 7 provides the conclusion.

2. Related work

Non-preemptive task scheduling on identical resources is an NP-hard problem [2, 3] addressed by exact methods, heuristics, and metaheuristics. Exact methods include Mixed Integer Programming (MIP) [15] and Constraint Programming (CP) [16], which guarantee optimality but struggle with scalability on larger instances due to combinatorial explosion. CP employs specialized propagation and search algorithms [17, 18] and has been applied to car sequencing [19]. However, traditional exact methods, such as branch-and-bound [20] or A* search [21], often become prohibitive for larger instances.

Boolean Satisfiability (SAT) based techniques have emerged as powerful exact methods for combinatorial scheduling problems [22, 23]. The approach encodes scheduling problems as SAT instances, leveraging advances in SAT solver technology [24]. SAT-based encodings have been applied to diverse scheduling problems, including employee timetabling [8], tournament scheduling [25], job-shop scheduling [26], minimizing latency in data-dependent tasks [27], mapping synchronous dataflow graphs [28, 29], and real-time system allocation [30, 31]. Heuristic approaches provide approximate solutions using techniques like genetic algorithms, simulated annealing, and tabu search [32, 33], though they cannot guarantee optimality.

The ES3 scheme by M a y a n k and M o n d a l [14] represents a notable SAT encoding for non-preemptive task scheduling. ES3 utilizes Boolean variables for task-resource assignments and temporal execution, with constraints ensuring the validity of schedules. However, ES3's scalability is limited by rapid growth in variables and clauses, particularly for task execution continuity constraints, making larger instances intractable.

Our Compact SAT Encoding (CSE) addresses these limitations through three main contributions: (1) refined constraint formulations based on ES3 [14] that improve efficiency; (2) a Block encoding technique described in Section 4 that achieves linear clause complexity for continuity constraints; (3) specialized symmetry-breaking constraints discussed in Section 5, which help reduce the search space while maintaining completeness.

3. SAT encoding scheme

This section presents the CSE scheme, which is an improvement over ES3 [14]. The integration of CSE encoding with the Block encoding technique and symmetry-breaking constraint representations has resulted in significantly better solution outcomes compared to other methods.

3.1. Problem formulation and variables

We consider a system of n independent, aperiodic tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled on m identical, non-preemptive resources. Each task τ_i has triplet (r_i, e_i, d_i) for release time, execution time, and deadline, with $e_i > 0$ and $r_i + e_i \leq d_i$. All parameters are integers (continuous values require discretization).

For task τ_i :

- Task time window $W_i = d_i - r_i$: Duration from release to deadline.
- Execution window: Actual e_i time units during execution.
- Feasible start time window $T_i = \{t \mid r_i \leq t \leq d_i - e_i\}$.

The scheduling horizon extends to $T_{\max} = \max_i(d_i)$. To model this problem as a SAT instance, we define two primary sets of Boolean variables:

- y_{ij} : True if task τ_i is assigned to resource j.
- z_i^t : True if task τ_i is active at time $t \in [r_i, d_i - 1]$.

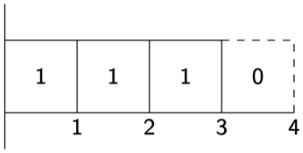

Fig. 1. Example of task execution: Task $\tau_i = (r_i = 0, e_i = 3, d_i = 4)$ starting at $t = 0$

Fig. 1 illustrates an example execution for task τ_i with parameters $r_i = 0$, $e_i = 3$, $d_i = 4$. In this scenario, the task starts at its earliest possible time, $t = 0$. It executes for $e_i = 3$ time units, so $z_i^0 = 1$, $z_i^1 = 1$, $z_i^2 = 1$. It must be completed by its deadline $d_i = 4$. The task is not running at $t = 3$, so $z_i^3 = 0$. The set of feasible start times for this task is $T_i = \{t \mid 0 \leq t \leq 4 - 3\} = \{0, 1\}$. This illustrates the general case where a task has a tight deadline relative to its execution time, constraining the feasible start time window T_i to only two possible values. More generally, for any task τ_i, the feasible start times are $T_i = \{t \mid r_i \leq t \leq d_i - e_i\}$, and the task must execute continuously for e_i time units once started.

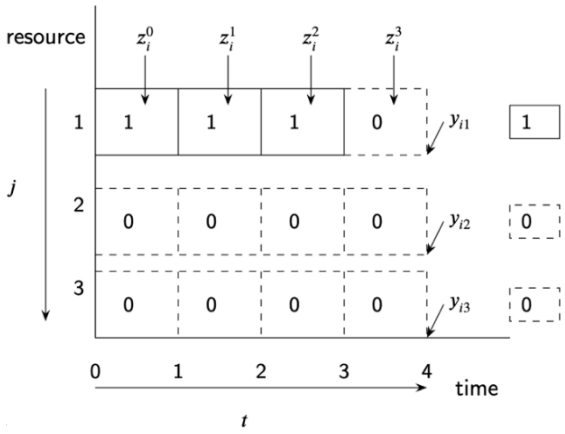

Fig. 2. Illustration of Boolean variables for a task τ_i (e.g., $e_i = 3$, starting at $t = 0$, assigned to resource $j = 1$). The central grid entries visualize the states of $z_i^t \wedge y_{ij}$ for each resource-time combination, where a value of 1 indicates that task i uses resource j at time t

Fig. 2 illustrates Boolean variables z_i^t (time axis) and y_{ij} (resource axis) for task τ_i. The example shows task τ_i with $e_i = 3$ executing on resource 1 from $t = 0$ to $t = 2$.

The encoding uses three constraint types:

- C1: Each task is assigned to exactly one resource.
- C2: Each resource is assigned to at most one task at any time.
- C3: Tasks execute continuously for e_i time units within their feasible windows.

3.2. Core constraints

Constraint C1: One resource per task. This constraint ensures that each task τ_i is assigned to exactly one resource. This is achieved through two sets of clauses:

- At most one resource per task:

(1) $$\neg y_{ij} \vee \neg y_{ij'} \quad \forall i, j \neq j'.$$

- At least one resource per task:

(2) $$\bigvee_{j=1}^{m} y_{ij} \quad \forall i.$$

Constraint C2: At most one task per resource. This constraint ensures that a resource is not simultaneously used by more than one task. For any two distinct tasks $i,\ i'$, for any resource j, and for any time unit t within the scheduling horizon where both tasks could potentially be active (i.e., $t \in [r_i,\ \min(d_i,\ d_j)]$), the condition $\left(z_i^t \wedge y_{ij}\right) \rightarrow \neg\left(z_{i'}^t \wedge y_{i'j}\right)$ must hold. This is equivalent to the following CNF clause:

(3) $$\neg z_i^t \vee \neg y_{ij} \vee \neg z_{i'}^t \vee \neg y_{i'j} \quad \forall i \neq i',\ \forall j,\ \forall t \in \left[0,\ \min(d_i,\ d_j)\right].$$

Constraint C3: Task execution continuity. Constraint C3 ensures that tasks only access resources during their designated execution windows and maintain continuous, non-preemptive execution once started. It also enforces that once a task starts executing, it continues without interruption for its entire execution time e_i.

Firstly, constraint C3.1 ensures that each task starts within its feasible time window (i.e., is active at some point that can be a valid start). We have:

$$\bigvee_{t=r_i}^{d_i-e_i} z_i^t \quad \forall i. \tag{4}$$

Secondly, we enforce the continuity of execution by constraint C3.2. This requirement ensures that if a task starts at any time s, it must run continuously for e_i time units and not run outside this execution window. For each task τ_i, this constraint consists of the following CNF clauses:

- If the task starts at release time, ensure continuous execution:

$$\begin{gathered} \neg z_i^{r_i} \vee z_i^{r_i+k} \quad \forall k \in [1,\ e_i - 1]\ \forall i, \\ \neg z_i^{r_i} \vee \neg z_i^{t} \quad \forall t \in [r_i + e_i,\ d_i - 1]\ \forall i. \end{gathered} \tag{5}$$

- If the task starts at time $t+1$ (transition from not running to running), ensure continuous execution:

$$\begin{gathered} z_i^t \vee \neg z_i^{t+1} \vee z_i^{t+k} \quad \forall t \in [r_i,\ d_i - e_i - 1],\ k \in [1,\ e_i - 1]\ \forall i, \\ z_i^t \vee \neg z_i^{t+1} \vee \neg z_i^{s} \quad \forall t \in [r_i,\ d_i - e_i - 1],\ s \in [t + e_i,\ d_i - 1]\ \forall i. \end{gathered} \tag{6}$$

The total variables $V = nm + \sum_{i=1}^{n} W_i$ and total number of clauses C is $C_{(1)} + C_{(2)} + C_{(3)} + C_{(4)} + C_{(5)} + C_{(6)}$:

- $C_{(1)} = n \cdot \frac{m(m-1)}{2} = O(nm^2)$;
- $C_{(2)} = n$;
- $C_{(3)} \approx \frac{n(n-1)}{2} \cdot m \cdot (d_i - e_i) = O(n^2 m W_i)$;
- $C_{(4)} = n$;
- $C_{(5)} + C_{(6)} = \sum_{i=1}^{n} \frac{(W_i - e_i + 1)(W_i + e_i - 2)}{2} = O\left(\sum_{i=1}^{n} W_i^2\right)$ (for large task windows where $W_i \gg e_i$).

The quadratic $C_{C3.2}$ $(C_{(5)} + C_{(6)})$ complexity motivates the Block encoding optimization in Section 4, which reduces this to $O(W_i)$ complexity. The effectiveness of shorter clauses in enhancing solver performance through unit propagation has been highlighted in seminal work [34], thereby enhancing computational efficiency.

4. Block encoding for task execution continuity

Constraint C3.2 enforces non-preemptive task execution continuity, generating a quadratic number of clauses, $O\left(W_i^2\right)$, for each task τ_i with time window $W_i = d_i - r_i$ (as analyzed in Section 3). Block encoding reduces this to $O(W_i)$ complexity using auxiliary variables and reusable template structures (detailed in Section 4.5).

4.1. Block encoding conceptual overview

For clarity in the following descriptions, we define:

- $X_t^i \equiv z_i^t$: Boolean variable indicating whether task τ_i is active at time t.
- $N_{\text{mb}} = \lceil W_i/(e_i - 1) \rceil$: Number of mini blocks needed to cover the task window.

For each task τ_i, we define pre-built template structures with auxiliary variables that efficiently encode C3.2 by representing:

- All-Zero block: Ensures no execution in tail intervals $[s + e_i,\ d_i - 1]$ using auxiliary variables R_q^{AZ}.
- All-One mini blocks: Ensure continuous execution for e_i time units using Left sub-block $\left(R_q^{\mathrm{L}}\right)$ and Right sub-block (R_q^{R}) auxiliary variables.

For each feasible start time s, merging clauses connect the original variables to auxiliary variables:

$$S_s^i \rightarrow R_{\mathrm{L},s}^i \wedge R_{\mathrm{R},s}^i \wedge \neg R_{\mathrm{AZ},s}^i,$$

where S_s^i represents the start condition for task τ_i at time s.

Fig. 3 illustrates the conceptual goal of C3.2: for different start times, specific sequences of z_i^t must be true (execution part) and others must be false (nonexecution part). In this figure, each row represents a different start time condition: if task τ_i starts at a specific time, then specific sequences of X (i.e., z_i^t) must be true for execution and false for tail non-execution.

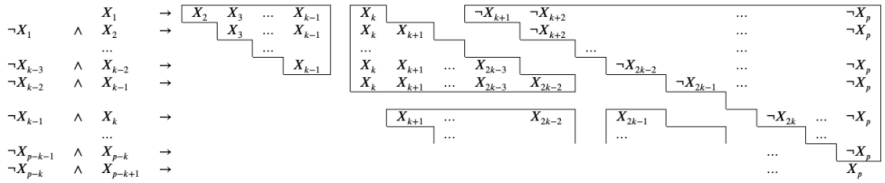

Fig. 3. Conceptual illustration of constraint C3.2 showing the staircase pattern for different start times of task $\tau_i (k = e_i,\ p = d_i - 1)$

Block encoding provides an efficient way to represent the consequences of these implications. It efficiently represents these patterns using: (1) All-Zero structures for tail non-execution segments, and (2) All-One mini blocks for continuous execution segments.

To illustrate the transformation from original C3.2 to Block encoding, consider a task τ_i with $(r_i = 1,\ e_i = 3,\ d_i = 6)$ having feasible start times $T_i = \{1,\ 2,\ 3\}$. The original C3.2 constraint generates explicit implications for each start time:

$$S_1^i \rightarrow \left(z_i^1 \wedge z_i^2 \wedge z_i^3\right) \wedge \left(\neg z_i^4 \wedge \neg z_i^5\right),$$
$$S_2^i \rightarrow \left(z_i^2 \wedge z_i^3 \wedge z_i^4\right) \wedge \left(\neg z_i^5\right),$$
$$S_3^i \rightarrow \left(z_i^3 \wedge z_i^4 \wedge z_i^5\right).$$

Block encoding replaces explicit variable patterns with auxiliary variables representing reusable template structures:

$$S_1^i \rightarrow R_{\mathrm{L},1}^i \wedge R_{\mathrm{R},1}^i \wedge \neg R_{\mathrm{AZ},1}^i,$$
$$S_2^i \rightarrow R_{\mathrm{L},2}^i \wedge R_{\mathrm{R},2}^i \wedge \neg R_{\mathrm{AZ},2}^i\ ,$$
$$S_3^i \rightarrow R_{\mathrm{L},3}^i \wedge R_{\mathrm{R},3}^i \wedge \neg R_{\mathrm{AZ},3}^i,$$

where auxiliary variables represent:

- $R_{\mathrm{L},s}^i$: Left sub-block template enforcing execution start patterns.
- $R_{\mathrm{R},s}^i$: Right sub-block template enforcing execution continuation patterns.
- $R_{\mathrm{AZ},s}^i$: All-Zero block template enforcing non-execution in tail intervals.

The key insight is that instead of enumerating every possible execution pattern explicitly, Block encoding uses prebuilt auxiliary variable structures that can be reused across different start times and tasks, achieving linear complexity while maintaining complete logical equivalence (proven in Appendix A).

4.2. Template structure definitions

Block encoding uses pre-built template structures with auxiliary variables that efficiently represent continuous execution and non-execution patterns. These templates are reusable across different tasks and start times, avoiding the quadratic clause explosion of the original C3.2 formulation.

4.2.1. Left sub-block template

The Left sub-block template handles the beginning portion of execution windows, creating a hierarchical structure of auxiliary variables that represent cumulative conjunctions. Each auxiliary variable R_q^{L} represents the conjunction $X_s \wedge X_{s+1} \wedge \ldots \wedge X_{s+q}$, enabling efficient representation of continuous execution segments.

For execution starting at time slot t, the Left sub-block covers variables from X_s to X_{s+e_i-3}, using $e_i - 3$ auxiliary variables $R_1^{\mathrm{L}}, \ldots, R_{e_i-3}^{\mathrm{L}}$. The hierarchical design allows any top-level auxiliary variable to enforce all required conjunctions below it, as illustrated in Fig. 4.

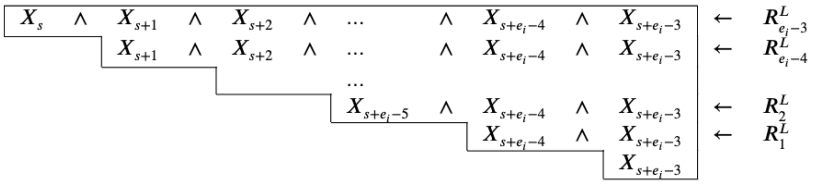

Fig. 4. Left All-One sub-block template with auxiliary variables R_q^L enforcing conjunctions over execution variables X

The CNF clauses enforce only the implication direction $R_q^{\mathrm{L}} \Longrightarrow$ conjunction. This suffices because merging clauses forces appropriate R_q^{L} variables to true when tasks start. For the Left sub-block with $e_i - 2$ Boolean variables $X_t, \ldots, X_{t+e_i-3}$ at starting position t, we have CNF clauses:

(7) For R_1^{L}: $\left(\neg R_1^{\mathrm{L}} \vee X_{t+e_i-4}\right),\ \left(\neg R_1^{\mathrm{L}} \vee X_{t+e_i-3}\right)$,
For ≥ 2 : $\left(\neg R_q^{\mathrm{L}} \vee R_{q-1}^{\mathrm{L}}\right),\ \left(\neg R_q^{\mathrm{L}} \vee X_{t+e_i-3-q}\right)$.

This template uses $e_i - 3$ auxiliary variables and $2(e_i - 3)$ clauses.

4.2.2. Right sub-block template

The Right sub-block template provides overlapping coverage with the Left sub-block to ensure complete execution continuity. For execution starting at time slot t, the Right sub-block covers variables from X_{s+e_i-2} to X_{s+2e_i-4}, using $e_i - 2$ auxiliary variables $R_1^{\mathrm{R}}, \ldots, R_{e_i-2}^{\mathrm{R}}$ to create the necessary hierarchical structure. When combined with the Left sub-block through merging logic, the combination enforces

continuous execution for all e_i time units. The hierarchical structure is shown in Fig. 5.

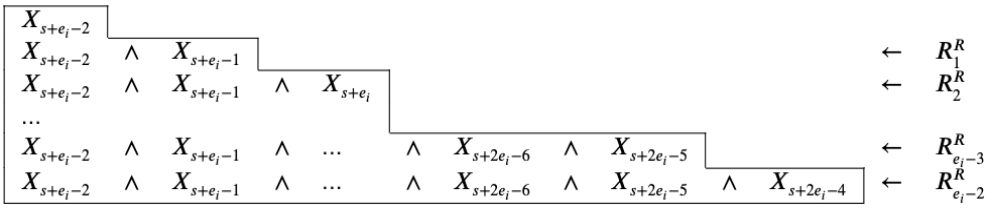

Fig. 5. Right All-One sub-block template for $e_i - 1$ variables using $e_i - 2$ auxiliary variables

Identical structure to the Left template, the CNF clauses enforce the implication direction $R_q^{\mathrm{R}} \Longrightarrow$ conjunction. This suffices because merging clauses forces appropriate R_q^{R} variables to true when tasks start. The CNF clauses are:

(8) $$\text{For } R_1^{\mathrm{R}}\text{:}\left(\neg R_1^{\mathrm{R}} \vee X_{\mathrm{s}+e_i-2}\right),\ \left(\neg R_1^{\mathrm{R}} \vee X_{\mathrm{s}+e_i-1}\right),$$
$$\text{For } q \geq 2, R_q^{\mathrm{R}}\text{:}\left(\neg R_q^{\mathrm{R}} \vee R_{q-1}^{\mathrm{R}}\right),\ \left(\neg R_q^{\mathrm{R}} \vee X_{\mathrm{s}+e_i-2+q}\right).$$

This template uses $e_i - 3$ auxiliary variables and $2(e_i - 2)$ clauses.

4.3. All-Zero block encoding template

The All-Zero block ensures tasks don't execute in tail intervals $[s + e_i,\ d_i - 1]$ after completing their designated execution. It uses auxiliary variables that represent disjunctions working backwards from the tail end, creating a hierarchical structure for efficient non-execution enforcement.

For a task starting at time s, the tail interval has length $N_{\text{tail}} = d_i - (s + e_i) = d_i - s - e_i$. The template uses $N_{\text{tail}} - 1$ auxiliary variables R_q^{AZ}, where each represents the disjunction of variables in successive tail segments. The auxiliary variable R_q^{AZ} becomes true if any execution variable in its corresponding tail segment is true. Fig. 6 illustrates this hierarchical structure.

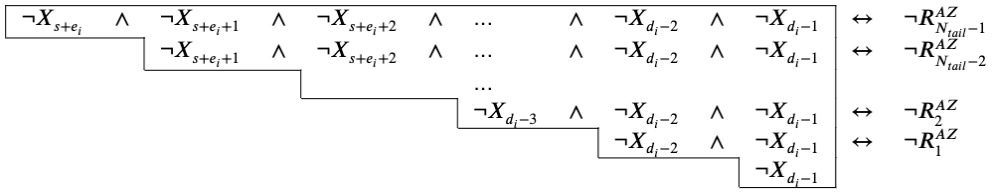

Fig. 6. All-Zero block template ensuring no execution in tail segments using auxiliary variables

The CNF clauses for the All-Zero block template implement the disjunction logic where R_q^{AZ} is true if any of the relevant z_i^t variables are true:

(9) $$\text{Base clauses for } R_1^{\mathrm{AZ}}\text{:}\left(\neg X_{d_i-2} \vee R_1^{\mathrm{AZ}}\right),\ \left(\neg X_{d_i-1} \vee R_1^{\mathrm{AZ}}\right),$$
$$\text{For } R_q^{\mathrm{AZ}}(q \geq 2)\text{:}\left(\neg R_{q-1}^{\mathrm{AZ}} \vee R_q^{\mathrm{AZ}}\right),\ \left(\neg X_{d_i-1-q} \vee R_q^{\mathrm{AZ}}\right).$$

The CNF clauses enforce the implication direction $R_q^{\mathrm{AZ}} \Longrightarrow$ conjunction. This one-way encoding is sufficient because the merging clauses will force the

appropriate $\neg R_q^{\mathrm{AZ}}$ variables to be true. For a tail of length $N_{\text{tail}} = W_i - e_i$, this uses $W_i - e_i - 1$ auxiliary variables and $2(W_i - e_i - 1)$ clauses.

4.4. Block encoding merging

The merging mechanism connects task start conditions to the appropriate auxiliary variables from pre-built template structures. For each feasible start time s of task τ_i, the start condition S_s^i is defined and linked to template auxiliary variables through merging clauses:

$$S_s^i = \begin{cases} z_i^{r_i} & \text{if } s = r_i, \\ \neg z_i^{s-1} \wedge z_i^s & \text{if } s > r_i. \end{cases}$$

The C3.2 constraint for task τ_i starting at time s is then enforced by linking S_s^i to the appropriate auxiliary variables from the pre-defined All-One and All-Zero block structures. Specifically, for task τ_i with feasible start time s, we enforce:

(10) $$\left(\neg S_s^i \vee R_{\mathrm{L},s}^i\right),\ \left(\neg S_s^i \vee R_{\mathrm{R},s}^i\right),\ \left(\neg S_s^i \vee \neg R_{\mathrm{AZ},s}^i\right).$$

The specific auxiliary variables are chosen based on start time s and execution requirements:

- $R_{\mathrm{L},s}^i$ and $R_{\mathrm{R},s}^i$: Top-level auxiliary variables from Left and Right sub-block structures ensuring continuous execution in $[s,\ s + e_i - 1]$.
- $R_{\mathrm{AZ},s}^i$: Auxiliary variable from All-Zero structure representing disjunction of variables in tail interval $[s + e_i,\ d_i - 1]$.

For $W_i - e_i + 1$ feasible start times per task, this generates $3(W_i - e_i + 1)$ merging clauses, providing an efficient encoding that maintains consistency with the block structure definitions.

4.5. Efficiency evaluation of Block encoding

The Block encoding scheme aims to reduce the C3.2 clause complexity for task τ_i from $O\left(W_i^2\right)$ in the original encoding to $O(W_i)$ in the block-encoded version.

For task τ_i, the Block encoding introduces auxiliary variables for the template structures. The total number of auxiliary variables is

$$V_{\text{aux},i} = \underbrace{(e_i - 3)}_{\text{Left sub-block}} + \underbrace{(e_i - 2)}_{\text{Right sub-block}} + \underbrace{(W_i - e_i - 1)}_{\text{All-Zero structure}} = W_i + e_i - 6.$$

The total number of clauses for the C3.2 constraint of task τ_i using Block encoding is

$$C_{3.2,i}^{\text{Block}} = \underbrace{2(e_i - 3)}_{\text{Left sub-block}} + \underbrace{2(e_i - 2)}_{\text{Right sub-block}} + \underbrace{2(W_i - e_i - 1)}_{\text{All-Zero structure}} + \underbrace{3(W_i - e_i + 1)}_{\text{Merging clauses}} =$$

$$= 2e_i - 6 + 2e_i - 4 + 2W_i - 2e_i - 2 + 3W_i - 3e_i + 3 = 5W_i - e_i - 9.$$

The Block encoding achieves $O(W_i)$ complexity because all terms in the formula scale linearly with W_i. This represents a significant asymptotic improvement over the quadratic growth of the original encoding. The trade-off is that Block encoding introduces $O(W_i)$ auxiliary variables, but empirical evidence shows that modern SAT solvers typically handle the increased variable count more

efficiently than the quadratic clause explosion, resulting in substantial performance improvements in practice.

5. Optimization and Symmetry-breaking constraints

Building upon the Block encoding improvements, we enhance SAT solver efficiency through optimization and symmetry-breaking constraints that prune the search space by eliminating redundant or infeasible solutions. These constraints target three key areas: resource conflict prevention, task ordering symmetries, and start time symmetries.

5.1. Overlapping constraint (S0)

This constraint prevents tasks with overlapping execution windows from being assigned to the same resource, directly targeting infeasible solutions:

$$\bigwedge_{i=1}^{n-1} \bigwedge_{i'=i+1}^{n} \bigwedge_{j=1}^{m} \left(\text{Overlap}(i,\, i') \rightarrow \left(\neg y_{ij} \vee \neg y_{i'j}\right)\right), \tag{11}$$

where Overlap $(i,\, i')$ evaluates to true if tasks i and i' have overlapping execution windows. This significantly reduces infeasible assignments the solver must consider, particularly in high resource contention scenarios.

5.2. Symmetry-breaking constraint (S1)

This constraint eliminates task-resource assignment symmetries by fixing tasks with early deadlines to ordered resources. This exploits the observation that tasks with limited scheduling flexibility (early deadlines, short execution times) can be assigned deterministically without losing completeness. Tasks with earliest completion times have limited flexibility and can be systematically assigned to resources in a predetermined order, eliminating permutation-based equivalent solutions.

For the set of "fixed" tasks $F = \{i \mid d_i - e_i \leq d_{\min}\}$ where

$$d_{\min} = \min_{i=1}^{n} d_i \colon \bigwedge_{j=1}^{\min(|F|,m)} y_{F_j j}. \tag{12}$$

This ensures each fixed task F_j is assigned to resource j in strict ordering, preserving at least one representative from each class of symmetrically equivalent solutions.

5.3. Time window constraint (S2)

This constraint eliminates start time symmetries by forcing tasks to execute within specific time windows when their feasible windows have limited overlap. This prevents the solver from exploring symmetrically equivalent schedules. When $d_i - e_i \leq r_i + e_i - 1$, constraint ensures task execution within the constrained window:

$$\bigwedge_{i=1}^{n} \left((d_i - e_i \leq r_i + e_i - 1) \rightarrow \bigwedge_{d_i - e_i}^{r_i + e_i - 1} z_i^t\right). \tag{13}$$

This forces tasks to execute at all time slots within the overlap window between the earliest and latest possible execution intervals. When the latest start time $d_i - e_i$ overlaps with or precedes the end of the earliest possible execution

$r_i + e_i - 1$, the task must execute during this entire overlap period regardless of its actual start time.

These constraints are designed for static task sets on identical resources and may require adaptation for dynamic task arrivals, preemption, or heterogeneous resources. They exploit specific symmetries inherent to the assumed model while maintaining solution completeness.

The enhanced CSE encoding integrates the foundational constraints (C1-C3.2) with Block encoding optimization and symmetry-breaking constraints (S0-S2) to achieve both algorithmic efficiency and search space reduction. Block encoding provides linear complexity for continuity constraints, while optimization constraints eliminate infeasible and redundant solutions, collectively resulting in a more efficient and scalable SAT-based scheduling approach.

6. Experimental results and analysis

All experiments were conducted on a machine with an Intel(R) Core (TM) i7-6700 CPU and 8GB of RAM. For our SAT-based approaches, we employed the high-performance CaDiCaL solver (V1.0.3), a winner of recent SAT competitions [35]. For comparison, we also evaluated leading general-purpose optimization solvers, including Google OR-Tools Version 9.11, IBM CPLEX Version 22.1.1.0, and Gurobi Version 9.5.0, under a 600-second timeout per instance. For At-Most-One constraints, we used PBLib [36] with BDD encoding [37].

We generated three distinct datasets: Medium, Large, and Huge to assess performance across different problem scales. Each dataset contains 100 instances to ensure statistical significance of the experimental results. The parameters for each dataset, including task release times, deadlines, and execution times, are detailed in Table 1.

Table 1. Range of parameter values

Parameter	Range I	Range II	Range III
Release time	[0, 7]	[0, 10]	[0, 10]
Deadline	[15, 35]	[20, 50]	[100, 120]
Execution time	[3, 7]	[5, 10]	[40, 70]

6.1. Experiments on Medium dataset

We first evaluated the baseline CSE scheme against the previous state-of-the-art ES3 [14] and general-purpose solvers on the Medium dataset (25-50 tasks).

Table 2. Models and configurations evaluated on the Medium dataset

Name	Description	Model basis / Constraints
ES3	Original encoding scheme from the previous work [14]	D1-D5 [14] (original ES3 constraints)
CSE	Proposed enhanced encoding scheme	(1)-(6) (proposed CSE constraints from Section 3)
OR-Tools_MIP	Using SCIP solver by Google	MIP formulation
OR-Tools_CP	Using OR-Tools CP solver	CP formulation
CPLEX_MIP	Using IBM CPLEX solver	MIP formulation
CPLEX_CP	Using Decision Optimization CP solver	CP formulation
Gurobi_MIP	Using Gurobi optimizer	MIP formulation

Table 2 lists evaluated configurations. SAT-based methods (ES3, CSE) are compared with MIP and CP solvers addressing equivalent scheduling problems. The performance results across all these configurations are then presented in Table 3.

Table 3. Performance comparison on the Medium dataset

Configuration	Time (s)
ES3	107.5
CSE	**43.7**
OR-Tools_MIP	365.1
OR-Tools_CP	529.6
CPLEX_MIP	8202
CPLEX_CP	4640
Gurobi_MIP	904.0

Medium dataset results show CSE significantly outperforming ES3 and all optimization solvers (43.7 s vs 107.5 s). Table 3 demonstrates that CSE achieves the fastest solving times among all evaluated methods. CSE reduces variables by 95.46% and clauses by 54.35% compared to ES3, as detailed in Table 4.

Table 4. Performance comparison of ES3 vs CSE on Medium dataset

Configuration	Time (s)	Number of variables	Number of clauses ($\times 10^6$)
ES3	107.5	5,135,308	115.2
CSE	**43.7**	**232,174**	**526.3**

6.2. Large and Huge dataset analysis

The primary goal of our advanced techniques is to enhance scalability. We evaluated various configurations on the Large and Huge datasets detailed in Table 5, with results presented in Table 6. The data reveals a clear trend: each optimization contributes to a significant performance gain. Original ES3 failed on Huge instances due to memory limitations. While adding symmetry-breaking to the baseline ES3 provided a marginal improvement, applying it to our CSE scheme (CSE_{SB}) reduced solving time on Huge instances to 26,065 s.

The most significant performance leap came from the introduction of the Block encoding technique. The CSE_{SB_Block} configuration further reduced the time to 23,258 s, underscoring the profound impact of linearizing the continuity constraints. The final configuration, $CSE_{SB\_Block\_PBLib}$, which combines all optimizations, yielded the best overall performance, solving the Huge instances in 23,103 s with significant variable/clause reductions. Analysis shows that Block encoding consistently reduces solving times through optimized continuity constraint representation. Combined with PBLib cardinality encoding and symmetry-breaking, the approach demonstrates superior scalability on complex instances.

Table 5. Configurations evaluated on Large and Huge datasets

Configuration	Description	Constraints
ES3	Original encoding scheme	D1-D5 [14]
$ES3_{SB}$	ES3 with Optimization and Symmetry-breaking	D1-D5 [14], S0-S2
CSE	Our proposed encoding scheme	C1.1-C3.2
CSE_{SB}	CSE with Optimization and Symmetry-breaking	C1.1-C3.2, S0-S2
CSE_{SB_PBLib}	CSE_{SB} with PBLib	$C1_{PBLib}$, C2-C3.2, S0-S2
CSE_{SB_Block}	CSE_{SB} with Block encoding	C1.1-C3.1, $C3.2_{Block}$, S0-S2
$CSE_{SB\_Block\_PBLib}$	CSE_{SB_Block} with PBLib	$C1_{PBLib}$, C2-C3.1, $C3.2_{Block}$, S0-S2

Table 6. Performance comparison on Large and Huge datasets ("-": Memory Out)

Configuration	Large dataset			Huge dataset		
	Time (s)	Number of vars ($\times 10^6$)	Number of clauses ($\times 10^8$)	Time (s)	Number of vars ($\times 10^6$)	Number of clauses ($\times 10^8$)
ES3	913	30.5	11.1	-	486.2	-
$ES3_{SB}$	870	30.5	11.1	45,341	486.2	17,882.7
CSE	480	**0.9**	6.2	-	**6.2**	-
CSE_{SB}	460	**0.9**	6.2	26,065	**6.2**	410.8
CSE_{SB_PBLib}	474	1.0	5.9	25,731	7.1	406.9
CSE_{SB_Block}	**458**	1.3	6.1	23,258	9.6	410.1
$CSE_{SB\_Block\_PBLib}$	463	1.5	**5.9**	**23,103**	10.5	**406.2**

To further investigate performance on the most challenging instances, Fig. 7 shows the top 15 longest solving times for different configurations on the Huge dataset. The plot clearly shows that the $ES3_{SB}$ configuration struggles significantly, while our CSE_{SB} offers a substantial improvement. The configurations incorporating Block encoding consistently occupy the lowest portion of the graph, confirming that this technique is particularly effective at simplifying the most computationally intensive instances.

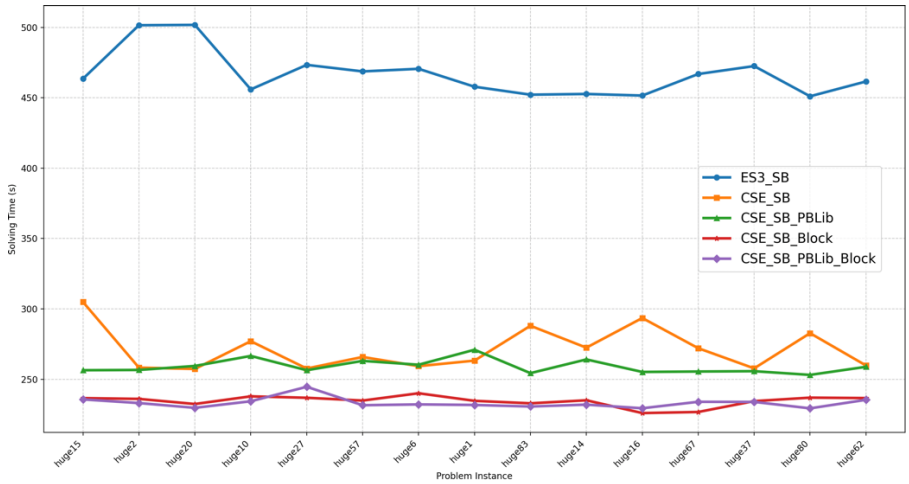

Fig. 7. Top 15 longest solving times for different configurations on Huge dataset

6.3. Solver performance comparison across dataset sizes

We meticulously analyzed the performance of leading SAT solvers-CaDiCaL, Glucose, Minisat, and MapleL-CMDistChronoBT our two most effective configurations: CSE_{SB_PBLib}, which integrates symmetry-breaking constraints and PBLib, and $CSE_{SB\_PBLib\_Block}$, which further incorporates Block encoding.

Table 7 reveals that CaDiCaL with $CSE_{SB\_PBLib\_Block}$ achieved the best total time (49,853.6 s), with performance advantages increasing on larger datasets. The 2628-second improvement over CSE_{SB_PBLib} on the Huge dataset demonstrates Block encoding's significant impact. This marked difference underscores the significant impact of Block encoding in streamlining the constraint representation, thus facilitating more efficient search space exploration by the solver. While the differences are noticeable on the smaller Large and Medium datasets, the performance gap widens significantly on the Huge dataset. This trend strongly suggests that our proposed approach, particularly with Block encoding, exhibits superior scalability compared to existing methods.

Prior research [24, 34, 38, 39] has extensively demonstrated that the performance of SAT solvers is highly sensitive to both the encoding strategy and the problem's inherent structure. In our experiments, we noted that solvers like Minisat, Glucose, which are known to utilize nonchronological backtracking [34, 40, 41], generally perform well on smaller instances. However, their performance tends to degrade more rapidly on larger datasets compared to solvers

like MLDC (MapleLCMDistChronoBT) and CaDiCaL, which employ chronological backtracking. This phenomenon aligns with the understanding that chronological backtracking, while potentially slower on smaller instances, often exhibits better scalability for larger, more complex problems. As [35] point out, chronological backtracking is often preferred when non-chronological back jumping would exceed a certain threshold, typically around 100 levels. The increased efficiency of chronological backtracking in such scenarios stems from its systematic exploration of the search space, which becomes increasingly advantageous as the problem's complexity grows. MapleLCMDistChronoBT, the winner of the SAT Competition 2019 in the main track [42], was chosen for its exceptional performance on industrial instances, demonstrating its ability to handle complex, real-world problems. Also, this solver shows its strength when dealing with our proposed encodings.

Table 7. Solver performance comparison

Solver	Configuration	Dataset size			Total time (s)
		Huge	Large	Medium	
CaDiCaL	$CSE_{SB\_PBLib\_Block}$	**23,103**	**462.5**	41.7	23,607.2
	CSE_{SB_PBLib}	25,731	474.0	**41.4**	26,246.4
CaDiCaL total		*48,834*	*936.5*	*83.1*	***49,853.6***
Glucose	$CSE_{SB\_PBLib\_Block}$	30,164	498.8	42.5	30,705.3
	CSE_{SB_PBLib}	30,775	497.2	45.4	31,317.6
Glucose total		*60,939*	*996.0*	*87.9*	*62,022.9*
Minisat	$CSE_{SB\_PBLib\_Block}$	31,241	494.9	43.2	31,779.1
	CSE_{SB_PBLib}	31,763	490.8	43.8	32,297.6
Minisat total		*63,004*	*985.7*	*87.0*	*64,076.7*
MLDC	$CSE_{SB\_PBLib\_Block}$	30,903	495.3	43.1	31,441.4
	CSE_{SB_PBLib}	31,171	519.6	45.1	31,735.7
MLDC total		*62,074*	*1014.9*	*88.2*	*63,177.1*

7. Conclusion

Our research in this paper presents an effective approach to solving the non-preemptive task scheduling problem using SAT solving. We demonstrate this through experimental results comparing the solving times of our method with those of powerful CP and MIP solvers, such as CPLEX, Gurobi, OR-Tools, and the previously proposed SAT-based ES3 scheme. Our encoding, called CSE, is more compact in terms of the number of variables and clauses when compared to ES3. The key contributions of the CSE encoding include several enhancements over ES3, the development of Block encoding for resource occupation constraints over continuous time intervals, aimed at reducing the number of clauses compared to

direct encoding, and the incorporation of symmetry-breaking constraints to narrow the search space. These improvements lead to significantly better solving times compared to ES3, and they even outperform strong solvers such as CPLEX, Gurobi, and OR-Tools. The detailed experimental configurations illustrate the progressive improvement in solving performance as we introduce the new SAT representation CSE, integrate Block encoding, and apply symmetry-breaking constraints step by step.

Furthermore, the Block encoding representation proposed in this paper is not limited to the non-preemptive task scheduling problem; it also shows potential for broader applications in resource scheduling problems, such as the Resource-Constrained Project Scheduling Problem (RCPSP), Job Shop Scheduling Problem (JSSP), and Assembly Line Balancing (ALB). These problems often require that each task continuously occupy a specific resource over a given time interval.

The findings of this research carry significant implications for the design and implementation of real-time systems. The improved SAT encoding scheme introduced in this paper equips system designers with an effective tool to tackle the complexities of non-preemptive scheduling on systems with multiple identical resources. The ability to efficiently determine the schedulability of tasks, allocate them to resources, and optimize their start times is crucial for ensuring the temporal correctness and reliability of safety critical applications, paving the way for developing more reliable and responsive real-time systems.

***Acknowledgements*:** This work has been supported by VNU University of Engineering and Technology under Project Number CN24.10.

***Data availability*:** The source code and experimental data supporting the findings of this research are openly available on GitHub at: https://github.com/kieuvantuyen01/CSE_NTSMR.

References

1. Buttazzo, G. C., M. Bertogna, G. Yao. Limited Preemptive Scheduling for Real-Time Systems: A Survey. – IEEE Transactions on Industrial Informatics, Vol. **9**, 2012, No 1, pp. 3-15.
2. Korst, J., E. Aarts, J. K. Lenstra, J. Wessels. Periodic Multiprocessor Scheduling. Berlin, Heidelberg, Springer, 1991.
3. Garey, M. R., D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, 1979.
4. Joo, I.-Y., D.-H. Choi. Optimal Household Appliance Scheduling Considering Consumer's Electricity Bill Target. – IEEE Transactions on Consumer Electronics, Vol. **63**, 2017, No 1, pp. 19-27.
5. Patel, J., G. Panchal. An IoT-Based Portable Smart Meeting Space with Real-Time Room Occupancy. – In: Intelligent Communication and Computational Technologies: Proceedings of Internet of Things for Technological Development, IoT4TD 2017, Springer, 2018, pp. 35-42.
6. Tran, L. D., A. Stojcevski, T. C. Pham, T. de Souza-Daw, N. T. Nguyen, V. Q. Nguyen, C. M. Nguyen. A Smart Meeting Room Scheduling and Management System with Utilization Control and ad-hoc Support Based on Real-Time Occupancy Detection. – In: Proc. of 6th IEEE International Conference on Communications and Electronics (ICCE'16), IEEE, 2016, pp. 186-191.

7. W a n g, C., X. L i, A. W a n g, X. Z h o u. A Classroom Scheduling Service for Smart Classes. – IEEE Transactions on Services Computing, Vol. **10**, 2015, No 2, pp. 155-164.
8. A l o u l, F. A., S. Z. Z a h i d i, A. A l-F a r r a, B. A l-R o h, B. A l-R a w i. Solving the Employee Timetabling Problem Using Advanced SAT & ILP Techniques. – Journal of Computing, Vol. **8**, 2013, No 4, pp. 851-858.
9. A l o u l, F., I. Z a b a l a w i, A. W a s f y. A SAT-Based Approach to Solve the Faculty Course Scheduling Problem. – In: 2013 Africon, IEEE, 2013, pp. 1-5.
10. P h a m, T. N., M.-F. T s a i, D. B. N g u y e n, C.-R. D o w, D.-J. D e n g. A Cloud-Based Smart-Parking System Based on Internet-of-Things Technologies. – IEEE Access, Vol. **3**, 2015, pp. 1581-1591.
11. R a o, Y. R. Automatic Smart Parking System Using Internet of Things (IoT). – International Journal of Engineering Technology Science and Research, Vol. **4**, 2017, No 5, pp. 1-8.
12. K r e m p l, M. Allocation of Trains to Platforms Optimization. – In: Proc. of 30th International Conference Mathematical Methods in Economics, Czech Republic, University of Economics, Prague, 2012, pp. 320-325.
13. C a r e y, M., S. C a r v i l l e. Scheduling and Platforming Trains at Busy Complex Stations. – Transportation Research Part A: Policy and Practice, Vol. **37**, 2003, No 3, pp. 195-224.
14. M a y a n k, J., A. M o n d a l. Efficient SAT Encoding Scheme for Schedulability Analysis of Non-Preemptive Tasks on Multiple Computational Resources. – Journal of Systems Architecture, Vol. **110**, 2020, 101818.
15. P a p a d i m i t r i o u, C. H., K. S t e i g l i t z. Combinatorial Optimization: Algorithms and Complexity. Mineola, NY, USA, Dover Publications, 1998.
16. R o s s i, F., P. V a n B e e k, T. W a l s h. Handbook of Constraint Programming. Amsterdam, Netherlands, Elsevier, 2006.
17. C u c u-G r o s j e a n, L., O. B u f f e t. Global Multiprocessor Real-Time Scheduling as a Constraint Satisfaction Problem. – In: Proc. of International Conference on Parallel Processing Workshops, IEEE, 2009, pp. 42-49.
18. W o e g i n g e r, G. J. An Efficient Algorithm for a Class of Constraint Satisfaction Problems. – Operations Research Letters, Vol. **30**, 2002, No 1, pp. 9-16.
19. D i n c b a s, M., H. S i m o n i s, P. v a n H e n t e n r y c k. Solving the Car-Sequencing Problem in Constraint Logic Programming. – In: Proc. of 8th European Conference on Artificial Intelligence (ECAI'18), 2018, London, UK, Pitman Publishing, 1988, pp. 290-295.
20. L a w l e r, E. L., D. E. W o o d. Branch-and-Bound Methods: A Survey. – Operations Research, Vol. **14**, 1966, No 4, pp. 699-719.
21. H a r t, P. E., N. J. N i l s s o n, B. R a p h a e l. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. – IEEE Transactions on Systems Science and Cybernetics, Vol. **4**, 1968, No 2, pp. 100-107.
22. B i e r e, A., M. H e u l e, H. van M a a r e n, T. W a l s h. Handbook of Satisfiability. Amsterdam, Netherlands, IOS Press, 2009.
23. A n s ó t e g u i, C., M. L. B o n e t, J. L e v y. SAT-Based MaxSAT Algorithms. – Artificial Intelligence, Vol. **196**, 2013, pp. 77-105.
24. M a r q u e s-S i l v a, J. P., K. A. S a k a l l a h. GRASP: A Search Algorithm for Propositional Satisfiability. – IEEE Transactions on Computers, Vol. **48**, 1999, No 5, pp. 506-521.
25. H o r b a c h, A., T. B a r t s c h, D. B r i s k o r n. Using a SAT-Solver to Schedule Sports Leagues. – Journal of Scheduling, Vol. **15**, 2012, pp. 117-125.
26. K o s h i m u r a, M., H. N a b e s h i m a, H. F u j i t a, R. H a s e g a w a. Solving Open Job-Shop Scheduling Problems by SAT Encoding. – IEICE Transactions on Information and Systems, Vol. **93**, 2010, No 8, pp. 2316-2318.
27. M e m i k, S. O., F. F a l l a h. Accelerated SAT-Based Scheduling of Control/Data Flow Graphs. – In: Proc. of IEEE International Conference on Computer Design: VLSI in Computers and Processors, IEEE, 2002, pp. 395-400.
28. L i u, W., M. Y u a n, X. H e, Z. G u, X. L i u. Efficient SAT-Based Mapping and Scheduling of Homogeneous Synchronous Dataflow Graphs for Throughput Optimization. – In: Proc. of 2008 Real-Time Systems Symposium, IEEE, 2008, pp. 492-504.

29. L i u, W., Z. G u, J. X u, X. W u, Y. Y e. Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems. – IEEE Transactions on Parallel and Distributed Systems, Vol. **22**, 2010, No 8, pp. 1382-1389.
30. M e t z n e r, A., M. F r a n z l e, C. H e r d e, I. S t i e r a n d. Scheduling Distributed Real-Time Systems by Satisfiability Checking. – In: Proc. of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), IEEE, 2005, pp. 409-415.
31. M e t z n e r, A., C. H e r d e. RTSAT – An Optimal and Efficient Approach to the Task Allocation Problem in Distributed Architectures. – In: Proc. of 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), IEEE, 2006, pp. 147-158.
32. D a v i s, L. Job Shop Scheduling with Genetic Algorithms. – In: Proc. of 1st International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1985, pp. 136-140.
33. S o t s k o v, Y. N., T.-C. L a i, F. W e r n e r. Measures of Problem Uncertainty for Scheduling with Interval Processing Times. – OR Spectrum, Vol. **35**, 2013, No 3, pp. 659-689.
34. E é n, N., N. S ö r e n s s o n. An Extensible SAT-Solver. – In: Proc. of International Conference on Theory and Applications of Satisfiability Testing, Springer, 2003, pp. 502-518.
35. B i e r e, A. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. – Proceedings of SAT Competition, Vol. **14**, 2017, pp. 316-336.
36. B a i l l e u x, O., Y. B o u f k h a d, O. R o u s s e l. A Translation of Pseudo Boolean Constraints to SAT. – Journal on Satisfiability, Boolean Modeling and Computation, Vol. **2**, 2006, No 1-4, pp. 191-200.
37. A b í o, I., R. A s ín, R. N i e u w e n h u i s, A. O l i v e r a s, E. R o d r í g u e z-C a r b o n e l l. BDDs for Pseudo-Boolean Constraints – Revisited. – In: Proc. of International Conference on Theory and Applications of Satisfiability Testing, 2012, Berlin, Heidelberg, Springer, 2012, pp. 63-76.
38. M o s k e w i c z, M. W., C. F. M a d i g a n, Y. Z h a o, L. Z h a n g, S. M a l i k. Chaff: Engineering an Efficient SAT Solver. – In: Proc. of 38th Annual Design Automation Conference, ACM, 2001, pp. 530-535.
39. G o l d b e r g, E., Y. N o v i k o v. BerkMin: A Fast and Robust SAT-Solver. – In: Proc. of Design, Automation and Test in Europe Conference and Exhibition 2002, IEEE, 2002, pp. 142-149.
40. A u d e m a r d, G., L. S i m o n. Predicting Learnt Clauses Quality in Modern SAT Solvers. – In: Proc. of 21st International Joint Conference on Artificial Intelligence (IJCAI'09), AAAI Press, 2009, pp. 399-404.
41. B i e r e, A. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. – In: SAT Race, 2010, pp. 1-4.
42. K o c h e m a z o v, S. F2TRC: Deterministic Modifications of SC2018-SR2019 Winners. – Proceedings of SAT Competition, 2020, pp. 21-22.
43. A f z a l i r a d, M., J. R e z a e i a n. Resource-Constrained Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times, Precedence Constraints and Machine Eligibility Restrictions. – Computers & Industrial Engineering, Vol. **98**, 2016, pp. 40-52.

Appendix A. Mathematical proof of block encoding equivalence

Theorem. Let τ_i be a task with execution time e_i and time window $W_i = d_i - r_i$. The constraint C3.2 and its Block encoding representation are logically equivalent.

Original constraint C3.2: For any feasible start time $s \in [r_i, d_i - e_i]$

$$S_s^i \rightarrow \left(\bigwedge_{k=0}^{e_i-1} z_i^{s+k} \wedge \bigwedge_{t=s+e_i}^{d_i-1} \neg z_i^t\right).$$

Block encoding representation: For the same start time s:

$$S_s^i \rightarrow \left(R_{\mathrm{L},s}^i \wedge R_{\mathrm{R},s}^i \wedge \neg R_{\mathrm{AZ},s}^i\right).$$

Lemma 1. The combination of Left and Right sub-blocks correctly represents the continuous execution requirement.

Proof: For a task starting at time s, the Left sub-block generates auxiliary variables such that $R_{L,s}^{i} \rightarrow \bigwedge_{k=0}^{e_i-3} z_i^{s+k}$ (covering $[s, s+e_i-3]$) and the Right sub-block ensures $R_{R,s}^{i} \rightarrow \bigwedge_{k=0}^{e_i-2} z_i^{s+e_i-2+k}$ (covering $[s+e_i-2, s+2e_i-4]$).

The union of Left and Right sub-blocks enforces continuous execution over the range $[s, s+2e_i-4]$, covering exactly e_i-1 time slots. Therefore, $R_{L,s}^{i} \wedge R_{R,s}^{i}$ enforces:

$$\bigwedge_{k=0}^{e_i-1} z_i^{s+k}$$

which spans from z_i^s to $z_i^{s+2e_i-4}$ with width e_i-1.

Lemma 2. The All-Zero block correctly represents the non-execution requirement.

Proof: The All-Zero block generates auxiliary variables such that $R_{\mathrm{AZ},s}^{i} \leftrightarrow \bigvee_{t=s+e_i}^{d_i-1} z_i^t$. By de Morgan's law:

$$\neg R_{\mathrm{AZ},s}^{i} \leftrightarrow \neg\left(\bigvee_{t=s+e_i}^{d_i-1} z_i^t\right) \leftrightarrow \bigwedge_{t=s+e_i}^{d_i-1} \neg z_i^t,$$

Therefore, $\neg R_{\mathrm{AZ},s}^{i}$ correctly represents the non-execution requirement for the tail interval. We have theorem proof:

Direction 1 (C3.2-Original $\Longrightarrow$ C3.2-Block): Assume the original constraint C3.2 is satisfied. Then for start time s, if S_s^i is true:

1. $\bigwedge_{k=0}^{e_i-1} z_i^{s+k}$ is true, which implies both $R_{\mathrm{L},s}^{i}$ and $R_{\mathrm{R},s}^{i}$ are true by Lemma 1.
2. $\bigwedge_{t=s+e_i}^{d_i-1} \neg z_i^t$ is true, which implies $\neg R_{\mathrm{AZ},s}^{i}$ is true by Lemma 2.
3. Therefore, $R_{\mathrm{L},s}^{i} \wedge R_{\mathrm{R},s}^{i} \wedge \neg R_{\mathrm{AZ},s}^{i}$ is satisfied.

Direction 2 (C3.2-Block $\Longrightarrow$ C3.2-Original): Assume the Block encoding constraint is satisfied. Then for start time s, if S_s^i is true:

1. $R_{\mathrm{L},s}^{i} \wedge R_{\mathrm{R},s}^{i}$ is true, which by the auxiliary variable definitions and Lemma 1 implies $\bigwedge_{k=0}^{e_i-1} z_i^{s+k}$ is true.
2. $\neg R_{\mathrm{AZ},s}^{i}$ is true, which by Lemma 2 implies $\bigwedge_{t=s+e_i}^{d_i-1} \neg z_i^t$ is true.
3. Therefore, the original constraint C3.2 is satisfied.

Since both directions hold, we have established that C3.2-Original $\equiv$ C3.2-Block, proving the logical equivalence between the original constraint and its Block encoding representation. This guarantees that the Block encoding optimization.

Received: 17.06.2025. Accepted: 07.08.07.2025