

The Multi-Skill Project Scheduling Problem: To SAT, or to PB, That is the Question

Wilber Bermeo¹, Jordi Coll¹ and Mateu Villaret¹

¹University of Girona (UdG), Girona, Spain

Abstract

This paper addresses the Multi-Skill Project Scheduling Problem (MSPSP), an NP-hard optimization problem. We propose and evaluate MaxSAT and pseudo-Boolean optimization encoding for the MSPSP. We explore the fundamental trade-off between the expressive power of pseudo-Boolean (PB) constraints and the raw efficiency of MaxSAT solvers. By adapting implied constraints naturally expressed as PB constraints from the literature, we examine whether such additional reasoning provides a greater performance boost to MaxSAT or PB solvers. Our initial hypothesis—that including natural PB constraints into our MSPSP model would provide a greater incremental boost to PB solvers than to MaxSAT solvers—was tested through an exhaustive experimental evaluation using state-of-the-art solvers. Contrary to our expectations, MaxSAT solvers consistently outperform PB solvers on the MSPSP. Moreover, our results demonstrate the benefits of integrating implied constraints, as they significantly reduce the number of timeouts, regardless of the formalism used.

Keywords

Satisfiability, MaxSAT, pseudo-Boolean Optimization, Multi-Skill Project Scheduling Problem, Implied Constraints

1. Introduction

Scheduling problems arise in a wide range of practical domains, such as manufacturing, logistics, and computer systems, where finding feasible solutions requires navigating complex constraints over resources and time [1, 2, 3, 4]. SAT-based methods have proven to be highly effective for handling these kind of combinatorial problems [5, 6, 7, 8, 9]. More recently, pseudo-Boolean solving and pseudo-Boolean optimization (PBO) have gained attention by allowing linear constraints over Boolean variables. Although pseudo-Boolean (PB) offers more expressive modelling and the potential for stronger inference than standard SAT [10, 11], a significant performance gap persists in many fields, with state-of-the-art SAT solvers often outperforming their PB counterparts [12, 13, 14].

The MSPSP is a generalization of the Resource-Constrained Project Scheduling Problem (RCPSP), a fundamental scheduling problem that models a wide range of industrial applications [15, 16, 17, 18]. The RCPSP is the problem of finding an optimal makespan determining the start times of a set of non-preemptive activities while respecting some constraints, namely precedence relations between activities and correct usage of shared resources with limited capacity. The MSPSP also considers a project composed of a set of non-preemptive activities to be scheduled, subject to end-start precedence constraints, with the objective of minimizing the makespan (see **Figure 1** and **Figure 2**). However, in the MSPSP, activities do not require resources but rather specific skills. These skills are provided by the renewable resources, each of which masters a subset of the skills. A resource can supply only the skills it masters and can contribute at most one unit of skill at any given time. The assignment of resources to a skill of an activity is non-preemptive, meaning that the assignment of resources to skills of an activity remains fixed throughout the execution of the activity.

In [19] it was argued that an instance of the MSPSP can be reformulated as an instance of the Multi-Mode Resource-Constrained Project Scheduling Problem—another variant of the RCPSP—by representing

17th Pragmatics of SAT International Workshop 2026 (PoS'26)

✉ wilber.bermeo@udg.edu (W. Bermeo); jordicoll@udg.edu (J. Coll); mateu.villaret@udg.edu (M. Villaret)

🌐 <https://wilberquito.github.io/> (W. Bermeo)

🆔 0009-0001-3748-8164 (W. Bermeo); 0000-0002-9385-5723 (J. Coll); 0000-0002-8066-3458 (M. Villaret)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

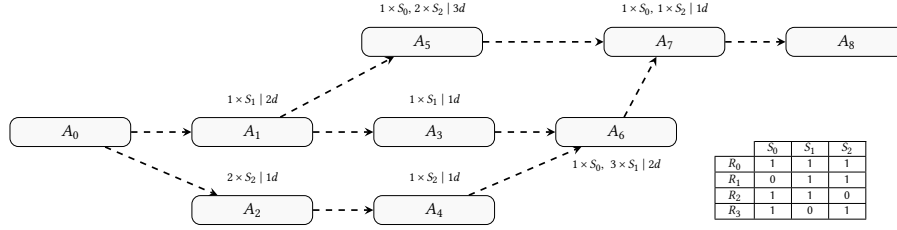


Figure 1: A MSPSP instance presented as an activity-on-node graph. It consists of seven non-dummy activities, four resources, and three skills. The units of required skills as well as the duration (in days) of each activity are set below each node. On the bottom right, the mastering matrix indicates what skills a resource master.

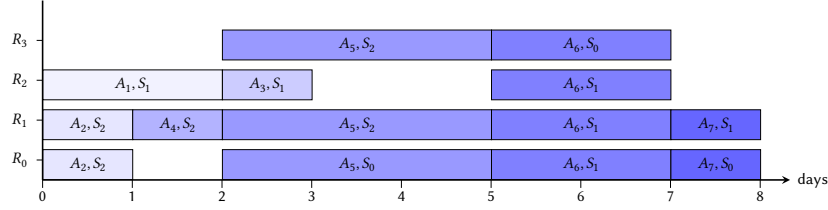


Figure 2: The MSPSP instance presented in **Figure 1** has an optimal makespan of eight days. Notice that some activities have been parallelized, for instance, $\{A_1, A_2\}$ and $\{A_3, A_5\}$.

different combinations of resource assignments to an activity as distinct execution modes. However, this transformation leads to a combinatorial explosion in the number of modes, making the approach computationally infeasible for larger instances. As a result, the MSPSP is strongly NP-hard and is typically addressed using problem-specific techniques rather than general-purpose formulations.

Several exact solutions have been developed to address the MSPSP. Mixed-Integer Linear Programming models, particularly time-indexed formulations, have demonstrated robust performance [20, 21]. These models frequently incorporate cumulative cuts to tighten the search space and accelerate convergence. Subsequently, [22] introduced a Constraint Programming model that employs a tailored search strategy and a Lazy Clause Generation solver and cumulative cuts were also exploited as implied constraints that strengthen propagation. More recently, [23] presented a Satisfiability Modules Theory formulation for the MSPSP that incorporates these same cuts.

Motivated by recent research into exact solutions, we present direct SAT and PB encodings for the MSPSP. This paper conducts a performance comparison between MaxSAT and PB solvers on the proposed encodings, augmented with the cumulative cuts introduced in [21] as implied constraints. While these implied constraints can be naturally added to the PB encoding as they are expressed as PB constraints, they need to be encoded into clausal form for SAT. We investigate whether the expressive modelling capabilities of PB can surpass the empirical efficiency of state-of-the-art MaxSAT solvers.

The rest of the paper is structured as follows: Section 2 provides preliminaries, and Section 3 defines the MSPSP and its solution criteria. Section 4 details the MaxSAT and PB optimization encodings, including implied constraints. Section 5 reports and discusses experimental results for both approaches. In Section 6 we speculate how exploiting PB proof systems can potentially reverse the observed performance trends.

2. Preliminaries

The SAT problem is the NP-complete problem *par excellence* [24]. NP-complete problems are considered hard due to their combinatorial nature (assuming $P \neq NP$). The goal of SAT is to determine whether a given Conjunctive Normal Form (CNF) *formula* F , represented as a finite conjunction of clauses—where each clause is a literal (a Boolean variable or its negation) or a finite disjunction of literals—is *satisfiable*, meaning that F admits an *assignment* ρ that maps every variable to a *truth* value, either *true* (1) or *false* (0) such that all clauses are satisfied. A clause is satisfied by an assignment ρ if any

of its literals is satisfied by ρ . A positive literal is satisfied if its variable takes the value *true* in the assignment whilst a negative literal is satisfied if its variable takes the value *false*. For example, the formula $\Gamma = (p \vee q) \wedge (p \vee \bar{q} \vee \bar{s}) \wedge s$ is satisfied by the assignment $\rho = \{p = 1, q = 0, s = 1\}$. A formula that admits no satisfying assignment is called *unsatisfiable*.

The underlying algorithm of modern SAT solvers is the Davis-Putnam-Logemann-Loveland (DPLL) procedure [25, 26]. In short, DPLL is a backtracking algorithm that incorporates Boolean Constraint Propagation (BCP)—BCP is complete for clauses: whenever a clause becomes unit under a partial assignment, the remaining literal must be set to true. At each step, a decision is made: a variable is chosen for branching and is assigned a truth value. Then the consequences of that decision are propagated by BCP. Each time a conflict is identified, the decision is backtracked: all assignments up to the last decision are undone, and the branching variable is forced to take the negated value. When all truth values have been tried without success, the previous decision is reversed. If there is no previous decision, then it can be asserted that the formula is unsatisfiable. This simplified description is anyhow far from state-of-the-art implementations of SAT solvers. What explains their astonishing success is the so-called Conflict-Driven Clause-Learning (CDCL) scheme, which enhances DPLL with other techniques: conflict analysis and backjumping (i.e., non-chronological backtracking) [27], learning new clauses from conflicts during backtrack search [27], variable decision heuristics that select the most active variables in recent conflicts, like VSIDS [28], data structures such as the 2-watched literal scheme [28] for efficiently identifying propagations and conflicts, restarts [29], clause cleanups that periodically delete counterproductive learned clauses, [30], among others. CDCL-based SAT solvers rely on the *resolution proof system*, although they do not explicitly prove unsatisfiability by deriving the empty clause through a direct, algorithmic application of the *resolution rule* [31]. Nevertheless, it is used to derive clauses (*lemmas*) that are logically entailed by the formula at hand. Resolution is *sound* and *complete* for propositional logic as propositional formulas can be transformed into clausal form.

Lately, the logic community has shown growing interest in the PB formalism, although it has been studied since the late 1960s in the context of operations research and 0-1 integer programming [32]. A PB formula is a set of constraints of linear inequalities on literals l_i in the form $\sum_i c_i l_i \bowtie A$, where $\bowtie$ is $\geq$ or $\leq$ and c_i and A are integer constants. A PB constraint can always be rewritten in normalized form $\sum_i c_i l_i \geq A$ with all literals over distinct variables and all c_i and A non-negative. As in SAT, the satisfiability of a PB formula is NP-complete and, similarly, it consists of assigning truth values to all variables so that all PB constraints are satisfied. For instance, the PB formula $\Gamma = (2x + 3\bar{y} + 4z \geq 3) \wedge (4\bar{z} \geq 3)$ is satisfied by the assignment $\rho = \{z = 0, y = 0, x = 1\}$.

PB is a rich area of study, since a wide variety of problems arising from different fields can be naturally formulated as the optimization of a PB function [33, 34, 35]. Compared to clauses, PB constraints offer a more expressive and natural way of modeling such problems. At the same time, the formalism remains sufficiently close to SAT to benefit from advances in SAT solving. However, a challenge arises when reducing a PB satisfiability problem to SAT. In SAT, clauses are the only primitive constraints, and PB constraints cannot be handled directly. One way to address this limitation is to translate PB constraints into clauses and solve them using a native SAT solver. Encoding-based approaches have been widely studied in the literature and can be quite efficient [36, 37, 38]. Another promising direction is to extend SAT-solving techniques so that PB constraints become first-class citizens of the solver and can be handled natively [39, 40, 41]. This extension requires, among other things, adapting propagation mechanisms, as well as algorithms and data structures, to efficiently detect propagations and conflicts. These extension do not come for free, and have trade-offs that need to be mentioned, for example, the propagation in SAT is based on BCP a relatively weak, but extremely efficient inference method. PB solvers, on the other hand, typically use PB propagation. In terms of raw expressive and deductive strength, PB propagation is strictly stronger than BCP because a single PB constraint can propagate many literals at once, whereas SAT requires many clauses to simulate the same reasoning, however, in practice PB propagation incurs a significantly higher computational cost [42]. Moreover, despite the theoretical advantage of the *cutting plane* proof system over *resolution*—demonstrated by *Pigeonhole Principle* (PHP) formulas [43]—CDCL-based PB solvers are rarely reported to outperform SAT solvers in practical applications.

Both SAT and PB formalisms offer NP-hard optimization variants: MaxSAT and PBO. In the case of MaxSAT, a widely studied variant is the Weighted Partial MaxSAT (WPMS). The WPMS combines *hard* and *soft* clauses, where all hard clauses must be satisfied while satisfying a subset of the weighted soft clauses such that the sum of the weights of the clauses of this subset is maximal [44]. For the remainder of this paper, we use the term MaxSAT to refer specifically to the WPMS problem. Analogously, a PBO problem consists of a set of PB constraints paired with a linear objective function $\sum_{i=1}^n c_i l_i$ to be minimized [45].

3. The MSPSP

The MSPSP is defined by a tuple (V, A, p, E, R, S, m, b) where:

V : Is a set of activities, i.e., $V = \{A_0, A_1, \dots, A_n, A_{n+1}\}$. Activities $\{A_0, A_{n+1}\}$ are dummy activities, by convention, they represent the start and the end of the schedule.

A : The set of non-dummy activities $A = \{A_1, \dots, A_n\}$.

p : Is a vector of naturals, with p_i being the duration of activity A_i . For the dummy activities, $p_0 = p_{n+1} = 0$, and $\forall A_i \in A : p_i > 0$.

E : Is a set of pairs of activities representing the precedence relations between end and start. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j , i.e., activity A_j must start after activity A_i has finished. We assume that we are given a precedence activity-on-node graph $G = (V, E)$ that contains no cycles, that A_0 is a predecessor of all other activities, and that A_{n+1} is a successor of all other activities.

R : Is a set of unary renewable resources, i.e., $R = \{R_1, \dots, R_r\}$.

S : Is a set of skills, i.e., $S = \{S_1, \dots, S_s\}$.

$m \in \mathbb{B}^{r \times s}$: Is a Boolean matrix. With $m_{r,s}$ being true iff resource R_r masters the skill S_s .

$b \in \mathbb{N}^{(n+2) \times s}$: Is a matrix of naturals, where $b_{i,s}$ represents the number of resources mastering the skill S_s that activity A_i requires during its execution. The start and end dummy activities do not require skills, i.e., $\forall S_s \in S : b_{0,s} = b_{n+1,s} = 0$.

A MSPSP *solution* is a tuple (τ, σ) where:

$\tau \in \mathbb{N}^{n+2}$: Is a vector of natural numbers that contains the start time of each activity.

$\sigma \in \mathbb{B}^{(n+2) \times r \times s}$: Is a 3D Boolean matrix. With $\sigma_{i,r,s}$ being *true* iff an activity A_i uses a resource R_r with a particular skill S_s .

The solution is subject to:

Precedence Constraints: These constraints ensure that for any pair of activities $(A_i, A_j) \in E$, activity A_j cannot start until A_i is completed. More formally: $\forall (A_i, A_j) \in E : \tau_j - \tau_i \geq p_i$.

Resource allocation consistency: A resource cannot perform a skill that it does not master. More formally: $\forall A_i \in A, \forall R_r \in R, \forall S_s \in S : \bar{m}_{r,s} \implies \bar{\sigma}_{i,r,s}$.

Satisfy skill requirements: Each activity must be assigned enough resources to satisfy its requirement. More formally: $\forall A_i \in A, \forall S_s \in S : \sum_{R_r \in R} \sigma_{i,r,s} \geq b_{i,s}$.

Resource constraints: A resource can only work on one skill at one activity at a time. More formally: $\forall t \in H, \forall R_r \in R : \sum_{S_s \in S} \sum_{A_i \in A} \text{ite}((\tau_i \leq t < \tau_i + p_i); \sigma_{i,r,s}; 0) \leq 1$. Where $\text{ite}(c; e_1; e_2)$ is the expression *if-then-else* denoting e_1 if c is true and e_2 otherwise.

An *optimal solution* is achieved by minimizing the start time of the end dummy activity A_{n+1} .

4. Encodings

In this section, we present the viewpoint and the high-level encoding of the MSPSP.

4.1. Viewpoint

To solve the MSPSP, we adopt the *time-indexed* formulation in which decision variables and constraints are defined over a discrete *scheduling horizon* H , ranging from zero to an *upper bound* (UB). Each activity is associated with a *time window*, namely the set of time instants during which the activity may be executed. Decision variables and the corresponding constraints are introduced only for these feasible instants.

A significant limitation of time-indexed formulations is their poor scalability, as the number of decision variables and constraints monotonically grows with the scheduling horizon. To mitigate this issue, at the encoding phase, we utilize the Parallel Scheduling Generation Scheme heuristic [46] to calculate a tight estimate of the project's UB . As a result, the number of variables and constraints introduced are reduced.

In addition, we precompute the *extended precedence graph* $G^* = (V, E^*)$, where E^* denotes the transitive closure of E . An edge $(A_i, A_j, l_{i,j}) \in E^*$ exists if and only if there is a path from A_i to A_j in G . The length of the critical path from A_i to A_j in G is denoted as the *time lag* $l_{i,j}$. Based on these time lags, we define the *earliest* and *latest* start and completion times of each activity, as well as the time windows at which an activity can be running (the *run-time window*) or can start (the *start-time window*).

$ES(A_i)$: Earliest start time of activity A_i , equal to $l_{0,i}$.

$LS(A_i)$: Latest start time of activity A_i , equal to $UB - l_{i,n+1}$.

$EC(A_i)$: Earliest close time of activity A_i , equal to $l_{0,i} + p_i$.

$LC(A_i)$: Latest close time of activity A_i , equal to $UB - l_{i,n+1} + p_i$.

$RTW(A_i)$: The run time window is the set of time instants at which an activity can be running, that is $ES(A_i) \dots LC(A_i) - 1$.

$STW(A_i)$: The start time window is the set of time instants at which an activity can start, that is $ES(A_i) \dots LS(A_i)$.

4.2. High-level encoding

Both PB and the SAT encoding can be described using a common high-level encoding, which we present in this section. This encoding relies on the observation that Boolean variables can also be interpreted as 0/1 integer (pseudo-Boolean) variables; with a slight abuse of notation, we use them interchangeably as summands or within logical expressions.

The high-level encoding requires a formalism-specific instantiation. In the case of SAT, cardinality and PB constraints must be translated into CNF. To preserve the solver's deductive efficiency, the chosen encoding should be arc-consistent, ensuring that unit propagation on the CNF achieves the same level of pruning as the original constraint [47].

In the PB setting, constraints are assumed to be normalized. Moreover, reifying a PB constraint—by introducing a *fresh variable*—yields a pair of constraints that preserve equisatisfiability [48]. Importantly, PB constraints should not be reformulated as clauses, as this would cause PB solvers to degenerate into CDCL SAT solvers [49].

To facilitate understanding of the high-level encoding, we first present the precomputed sets that tightly bound the number of introduced variables and constraints. We then introduce the variables used in the encoding and their semantics and finally the high-level encoding and the implied constraints.

4.2.1. Auxiliary pre-computed sets

$R(A_i)$: Subset of resources that master at least one of the skills required by activity A_i .

$S(A_i)$: Subset of skills that activity A_i demand, i.e. skills in S such that $b_{i,s} > 0$.

4.2.2. Decision variables

$s_{i,t}$: Boolean variables, which are true iff A_i has already started at time t , $\forall A_i \in V$, $\forall t \in STW(A_i)$.

$x_{i,t}$: Boolean variables, which are true iff activity A_i is running at time t , $\forall A_i \in A$, $\forall t \in RTW(A_i)$.

$ar_{i,r}$: Boolean variables, which are true iff activity A_i uses resource R_r , $\forall A_i \in A$, $\forall R_r \in R(A_i)$.

$ars_{i,r,s}$: Boolean variables, that are true iff activity A_i uses resource R_r for skill S_s , $\forall A_i \in A$, $\forall R_r \in R(A_i)$, $\forall S_s \in S(A_i)$ such that $m_{r,s}$, i.e., the resource R_r masters the skill S_s .

$art_{i,r,t}$: Boolean variables, which are true iff activity A_i uses resource R_r at time t , $\forall A_i \in A$, $\forall R_r \in R(A_i)$, $\forall t \in RTW(A_i)$.

4.2.3. High-level encoding

$$s_{0,0} \geq 1 \tag{1}$$

$$\bar{s}_{i,t} + s_{i,t+1} \geq 1 \quad \forall A_i \in A, \forall t \in STW(A_i) \tag{2}$$

$$\bar{s}_{j,t} + s_{i,t-p_i} \geq 1 \quad \forall (A_i, A_j) \in E, \forall t \in STW(A_i) \tag{3}$$

$$x_{i,t} \leftrightarrow s_{i,t} + \bar{s}_{i,t-p_i} \geq 2 \quad \forall A_i \in A, \forall t \in RTW(A_i) \tag{4}$$

$$ar_{i,r} \leftrightarrow \sum_{\substack{S_s \in S(A_i), \\ s.t. : m_{r,s}}} ars_{i,r,s} \geq 1 \quad \forall A_i \in A, \forall R_r \in R(A_i) \tag{5}$$

$$art_{i,r,t} \leftrightarrow ar_{i,r} + x_{i,t} \geq 2 \quad \forall A_i \in A, \forall R_r \in R(A_i), \forall t \in RTW(A_i) \tag{6}$$

$$\sum_{\substack{R_r \in R, \\ s.t. : m_{r,s}}} ars_{i,r,s} \geq b_{i,s} \quad \forall A_i \in A, \forall S_s \in S(A_i) \tag{7}$$

$$\sum_{\substack{S_s \in S(A_i), \\ s.t. : m_{r,s}}} ars_{i,r,s} \leq 1 \quad \forall A_i \in A, \forall R_r \in R(A_i) \tag{8}$$

$$\sum_{\substack{A_i \in A, s.t. : \\ t \in RTW(A_i), \\ R_r \in R(A_i)}} art_{i,r,t} \leq 1 \quad \forall R_r \in R, \forall t \in H \tag{9}$$

$$\text{minimizing} \quad \sum_{\forall t \in H} \bar{s}_{n+1,t} \tag{10}$$

Constraint (1) ensures that the initial dummy activity starts at time 0. Constraints (2) states that if an activity has started at time t_i for sure is running at time t_{i+1} . Constraints (3) define the precedences. Constraints (4), (5), and (6) are channeling constraints that introduce and define the semantics of the fresh variables $x_{i,t}$, $ar_{i,k}$ and $art_{i,r,t}$ respectively. Constraints (7) ensure that each activity has enough resources to cover each one of its skills. Note that it is implicitly forbidden that a resource R_r perform a skill S_s that it does not master, since the corresponding variable $ars_{i,r,s}$ is not introduced. Constraints (8) state that a resource can contribute with at most one skill in an activity. Constraints (9) ensure that a resource does not contribute to more than one activity at a time. For the SAT encoding instantiation, we

encode the *ALK* cardinality constraints¹ using the sorting network-based encoding from [50]. The *AMO* constraints are encoded using pairwise mutual exclusions between variables². Finally, the optimization function (10) maximizes the number of variables $s_{n+1,t}$ that hold true; consequently, the makespan of the project is reduced. For MaxSAT, the optimization function is emulated by introducing *weighted soft clauses*, i.e., $\forall t \in STW(A_{n+1}) : (s_{n+1,t}, \omega)$, where $\omega = 1$.

4.2.4. Implied constraints

Several works in the literature benefit greatly from the use of implied constraints [51, 52, 53]. The primary purpose of implied constraints is to provide additional, though not mandatory, information resulting in a pruned search space. We incorporate the implied constraints (11), (12), and (13) presented as cumulative cuts in [21].

$$\sum_{\substack{A_i \in A, \text{ s.t. :} \\ t \in RTW(A_i)}} x_{i,t} \cdot b_{i,s} \leq \sum_{R_r \in R} m_{r,s} \quad \forall t \in H, \forall S_s \in S \quad (11)$$

$$\sum_{\substack{A_i \in A, \text{ s.t. :} \\ t \in RTW(A_i)}} \left(x_{i,t} \cdot \sum_{S_s \in S} b_{i,s} \right) \leq |R| \quad \forall t \in H \quad (12)$$

Let us denote by $PW(S)$ the power set of S , and by $RL(S')$ the set of resources that master at least one of the skills in S' , with $S' \in PW(S)$.

$$\sum_{\substack{A_i \in A, \text{ s.t. :} \\ t \in RTW(A_i)}} \left(x_{i,t} \cdot \sum_{S_s \in S} b_{i,s} \right) \leq |RL(S')| \quad \forall t \in H, \forall S' \in PW(S) \quad (13)$$

Implied I (11) stipulates that for any skill, the demand from activities active at any time point cannot exceed the number of resources possessing that skill. Implied II (12) ensures that the total demand for skill units at any time does not exceed the total available resources. Similarly, Implied III (13) generalizes this logic to any subset of skills. While the number of constraints in Implied III can be massive for large skill sets, [21] noted that many of these constraints are redundant (dominated by others) and therefore, can be removed. Moreover, any constraint introduced by Implied I and II appears in implied III.

5. Results

This section evaluates the performance of the encodings introduced in Section 4. Benchmarks were conducted using publicly available instance sets from [22]. In **Table 1** we provide a summary of the characteristics of each set. Performance was assessed using EXACT [39] and ROUNDINGSAT [40] as PB solvers, and EVALMAXSAT [54] and MAXCDCL [55] as MaxSAT solvers, all drawn from their respective 2024 competitions using x86_64 Linux machines equipped with Intel® Xeon® E3-1220 v2 CPUs at 3.10 GHz and 8GB of RAM. To ensure reproducibility, the code, datasets, and additional material are available at: <https://github.com/wilberquito/mspsp-05-2026>.

The results of **Table 2** indicate that the MaxSAT solvers consistently outperform the PB solvers in the MSPSP. Furthermore, the varying degrees of inferential power of the implied constraints are empirically notable; Implied I (11) provides a limited boost to solver performance because it reasons about resource limits for a single skill at a time, resulting in relatively weak pruning. In contrast, Implied

¹By *ALK* we refer to the *at least k* constraint requiring at least k many values of a *list* to be true.

²Alternatives to the pairwise mutual exclusion encoding could be considered due to the large number of such constraints; however, this choice does not affect the conclusions of this work.

Table 1

Characteristics of the MSPSP instance sets used in the experiments. Values represent the number of instances (N), activities (including dummy activities), required skills, and the range of available resources.

Instance Set	Instances (N)	Activities	Skills	Resources
set-1a	216	22	4	10–30
set-1b	216	42	4	20–60
set-2a	110	20–51	2–8	5–14
set-2b	77	32–62	9–15	5–19
set-2c	91	22–32	3–12	4–15
set-3a	216	82	4	30–45
set-3b	216	62	4	30–45

Table 2

Performance impact of implied constraints on solver timeouts ($N = 1142$). Bitmask positions (left-to-right) denote the activation of the first, second, and third implied constraints. Cells report the number of timeouts, absolute reduction ($\downarrow \Delta$), and percentage (%) reduction relative to the baseline (000).

Solver	Implied Constraints (Bitmask)				
	000	100	010	110	001
EVALMAXSAT	616 (—)	613 ($\downarrow 3$; 0.5%)	458 ($\downarrow 158$; 25.6%)	448 ($\downarrow 168$; 27.3%)	423 ($\downarrow 193$; 31.3%)
MAXCDCL	628 (—)	621 ($\downarrow 7$; 1.1%)	510 ($\downarrow 118$; 18.8%)	503 ($\downarrow 125$; 19.9%)	466 ($\downarrow 162$; 25.8%)
EXACT	678 (—)	671 ($\downarrow 7$; 1.0%)	559 ($\downarrow 119$; 17.6%)	550 ($\downarrow 128$; 18.9%)	523 ($\downarrow 155$; 22.9%)
ROUNDINGSAT	729 (—)	723 ($\downarrow 6$; 0.8%)	629 ($\downarrow 100$; 13.7%)	619 ($\downarrow 110$; 15.1%)	599 ($\downarrow 130$; 17.8%)

II (12) exhibits significantly higher propagation power; by accounting for the total number of resources, it imposes more meaningful constraints. The simultaneous activation of Implied I and II results in perfectly additive performance gains for MAXCDCL, but other solvers exhibit a super-additive effect, i.e., the combination of both constraints, yields a greater reduction in timeouts than the sum of their individual contributions. The most significant boost is achieved by Implied III (13), by reasoning over skill subsets not only implicitly introduces the reasoning of Implied I and II, but it also captures other resource-skill interactions ignored by them.

Figure 3 illustrates that the hardness of a MSPSP instance is important driven by the scheduling scale. Instances with ≥ 42 activities are consistently more difficult than those with fewer activities. For example, *set-1a* and *set-2c* are clustered in the bottom-left region—indicating relatively “easy” instances—whereas *set-3a* and *set-3b* tend to appear in the upper-right region—indicating “harder” instances. Notably, the sets of unsolved instances do not fully overlap: EXACT proves optimality on some instances where EVALMAXSAT times out. This divergence suggests that identifying structural features favoring EXACT could inform new strategies to improve PB solving and optimization.

Finally, we present a numerical performance comparison between the EXACT and EVALMAXSAT solvers (see **Table 3**), as each performs better within its respective formalism. This table provides a more detailed view of the information summarized in **Table 2**. Interestingly, enabling only Implied I appears to benefit the Exact solver more than the EvalMaxSAT solver. For instance, in the *set-1a*, activating only Implied I results in two additional timeouts compared to the baseline for the EvalMaxSAT solver, whereas for the Exact solver it reduces the number of timeouts by one. Although this effect might seem negligible due to the small differences, it is also seen in other sets.

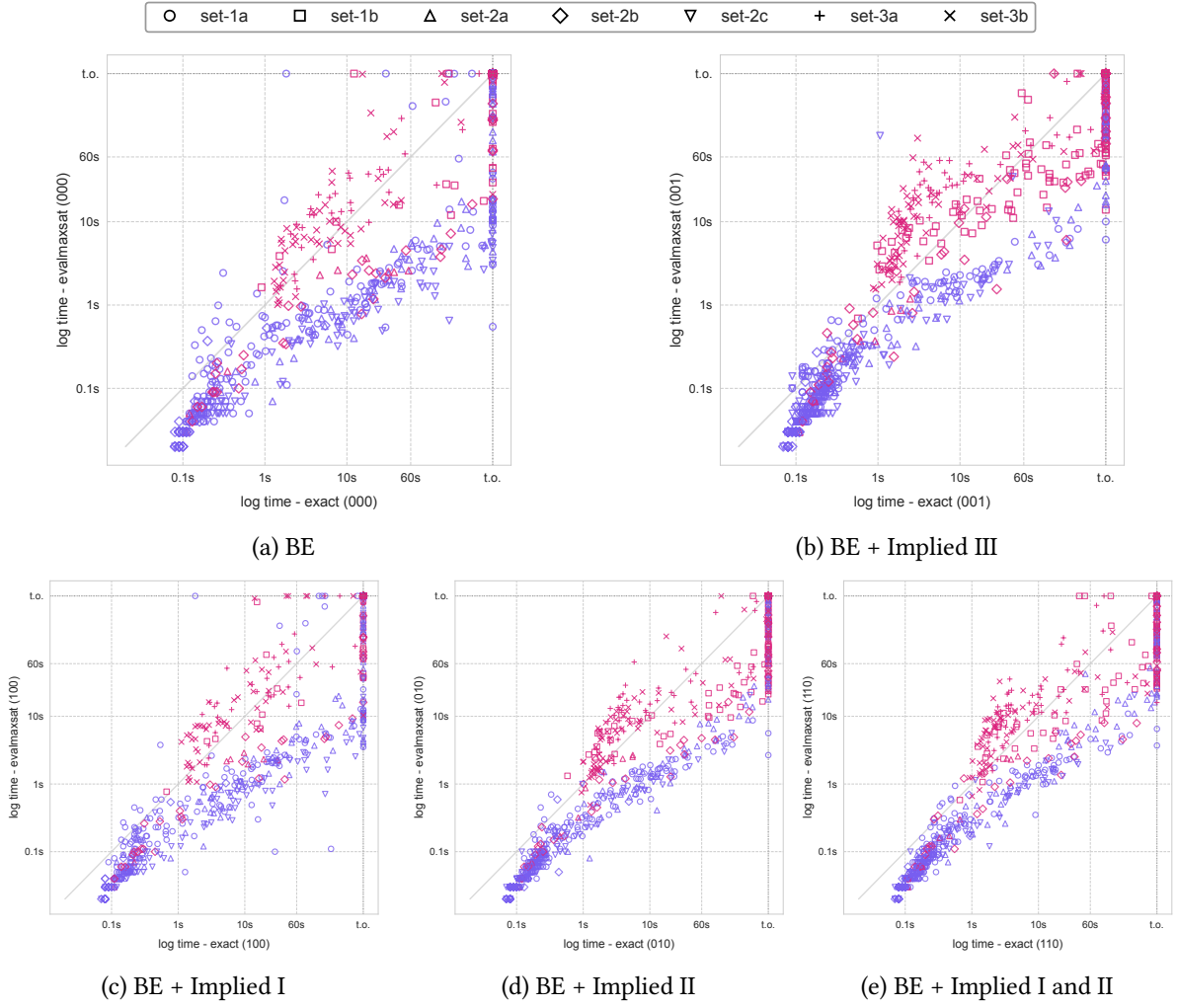


Figure 3: Pairwise log–log scatter plot comparing the runtimes of EVALMAXSAT and EXACT on the base encoding (BE) with implied constraints. Pink markers denote instances with ≥ 42 activities (including dummy activities), and violet markers otherwise. The optimality timeout is set to $t.o. = 600$.

Table 3 Solving time results for the MSPSP instances (values in seconds; 600s timeout). Bitmask positions (left-to-right) denote the activation of the first, second, and third implied constraints. Δ t.o. represents the absolute reduction in timeouts compared to the 000 baseline for each solver. Bold values indicate the best performer per configuration.

Set	Mask	Solver	Q1	med.	Q3	avg.	t.o.	Δ t.o.
set-1a	000	EVALMaxSAT	0.07	0.55	4.82	86.69	26	—
		EXACT	0.20	1.18	69.65	125.01	37	—
	100	EVALMaxSAT	0.07	0.29	4.14	84.77	28	+2
		EXACT	0.19	0.66	60.01	119.09	36	-1
	010	EVALMaxSAT	0.06	0.09	0.45	0.53	0	-26
		EXACT	0.17	0.24	1.47	13.03	2	-35
	110	EVALMaxSAT	0.06	0.09	0.34	0.56	0	-26
		EXACT	0.16	0.24	0.63	12.64	2	-35
	001	EVALMaxSAT	0.06	0.11	0.29	0.66	0	-26
		EXACT	0.16	0.21	0.40	12.59	2	-35
set-1b	000	EVALMaxSAT	t.o.	t.o.	t.o.	559.46	197	—
		EXACT	t.o.	t.o.	t.o.	573.09	204	—
	100	EVALMaxSAT	t.o.	t.o.	t.o.	548.51	193	-4
		EXACT	t.o.	t.o.	t.o.	573.13	206	+2
	010	EVALMaxSAT	104.16	t.o.	t.o.	413.95	133	-64
		EXACT	t.o.	t.o.	t.o.	515.21	178	-26
	110	EVALMaxSAT	99.04	t.o.	t.o.	410.35	131	-66
		EXACT	t.o.	t.o.	t.o.	495.87	172	-32
	001	EVALMaxSAT	44.41	414.29	t.o.	343.69	101	-96
		EXACT	t.o.	t.o.	t.o.	t.o.	t.o.	t.o.

Continued on next page...

Table 3 – continued from previous page

Set	Mask	Solver	Q1	med.	Q3	avg.	t.o.	Δ t.o.
		EXACT	239.15	t.o.	t.o.	447.85	149	-55
set-2a	000	EvalMaxSAT	1.23	8.24	t.o.	217.16	34	—
		EXACT	11.33	292.14	t.o.	317.12	53	—
	100	EvalMaxSAT	0.87	7.02	t.o.	203.37	33	-1
		EXACT	6.12	242.46	t.o.	301.17	50	-3
	010	EvalMaxSAT	0.52	4.62	205.37	139.57	16	-18
		EXACT	1.88	82.98	t.o.	265.05	44	-9
	110	EvalMaxSAT	0.27	6.74	125.42	110.75	10	-24
		EXACT	1.00	53.19	t.o.	260.67	41	-12
	001	EvalMaxSAT	0.23	1.96	122.85	106.86	9	-25
		EXACT	0.48	7.36	t.o.	239.72	41	-12
set-2b	000	EvalMaxSAT	0.03	0.17	4.64	89.53	8	—
		EXACT	0.10	0.26	142.33	145.60	17	—
	100	EvalMaxSAT	0.03	0.11	4.99	89.33	9	+1
		EXACT	0.09	0.30	164.81	147.80	17	0
	010	EvalMaxSAT	0.03	0.13	4.76	76.19	8	0
		EXACT	0.10	0.25	95.16	137.13	16	-1
	110	EvalMaxSAT	0.03	0.12	5.34	74.99	7	-1
		EXACT	0.10	0.27	43.05	135.12	16	-1
	001	EvalMaxSAT	0.03	0.23	9.68	70.77	7	-1
		EXACT	0.10	0.24	20.91	108.92	12	-5
set-2c	000	EvalMaxSAT	0.32	0.80	3.38	40.87	3	—
		EXACT	1.82	12.32	321.74	166.19	19	—
	100	EvalMaxSAT	0.24	0.84	3.26	43.16	4	+1
		EXACT	1.22	10.28	255.40	158.80	18	-1
	010	EvalMaxSAT	0.03	0.10	0.60	0.65	0	-3
		EXACT	0.10	0.20	1.74	11.03	0	-19
	110	EvalMaxSAT	0.04	0.10	0.43	0.67	0	-3
		EXACT	0.10	0.22	1.86	8.26	0	-19
	001	EvalMaxSAT	0.05	0.14	0.69	2.78	0	-3
		EXACT	0.12	0.32	2.40	14.81	1	-18
set-3a	000	EvalMaxSAT	t.o.	t.o.	t.o.	512.59	183	—
		EXACT	t.o.	t.o.	t.o.	511.78	182	—
	100	EvalMaxSAT	t.o.	t.o.	t.o.	511.10	181	-2
		EXACT	t.o.	t.o.	t.o.	510.39	181	-1
	010	EvalMaxSAT	t.o.	t.o.	t.o.	466.16	163	-20
		EXACT	t.o.	t.o.	t.o.	481.47	169	-13
	110	EvalMaxSAT	535.25	t.o.	t.o.	464.64	160	-23
		EXACT	t.o.	t.o.	t.o.	474.06	168	-14
	001	EvalMaxSAT	t.o.	t.o.	t.o.	477.63	166	-17
		EXACT	t.o.	t.o.	t.o.	477.62	169	-13
set-3b	000	EvalMaxSAT	t.o.	t.o.	t.o.	468.58	165	—
		EXACT	t.o.	t.o.	t.o.	465.51	166	—
	100	EvalMaxSAT	t.o.	t.o.	t.o.	467.73	165	0
		EXACT	t.o.	t.o.	t.o.	461.19	163	-3
	010	EvalMaxSAT	21.28	t.o.	t.o.	401.64	138	-27
		EXACT	30.79	t.o.	t.o.	427.26	150	-16
	110	EvalMaxSAT	24.96	t.o.	t.o.	401.76	140	-25
		EXACT	33.52	t.o.	t.o.	430.63	151	-15
	001	EvalMaxSAT	53.79	t.o.	t.o.	414.63	140	-25
		EXACT	22.23	t.o.	t.o.	424.65	149	-17

6. Conclusions

Contrary to our initial expectations, the results indicate that state-of-the-art MaxSAT solvers currently outperform PBO solvers on the MSPSP, even when incorporating implied constraints that are naturally expressed as PB inequalities. We attribute this performance discrepancy to the role of auxiliary variables introduced during the PB-to-CNF or cardinality-to-CNF translation processes [56, 57]. In MaxSAT, these variables function as high-level structural descriptors that facilitate more effective CDCL-based reasoning. In contrast, while these implied constraints provide meaningful information within the PB formalism, they do not explicitly expose an underlying PHP structure to the solver. Given that PB proof systems can theoretically produce polynomial-length proofs for PHP instances—whereas resolution-based reasoning in SAT requires exponential time [43]—we conjecture that explicitly encoding these structural abstractions could significantly enhance PB solver performance. This remains an open

direction for future work.

Acknowledgments

This work was funded by MCIN/MICIU/AEI/10.13039/501100011033 and by ERDF A way of making Europe (grants PID2021-122274OB-I00 and PID2024-157625OB-I00).

Declaration on Generative AI

During the preparation of this work, the authors used Gemini in order to: Paraphrase and reword, Improve writing style, and Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

- [1] J. Wang, X. Hu, E. Demeulemeester, Y. Zhao, A bi-objective robust resource allocation model for the rcpsp considering resource transfer costs, *International Journal of Production Research* 59 (2021) 367–387.
- [2] C. Schwindt, N. Trautmann, Batch scheduling in process industries: an application of resource-constrained project scheduling, *OR-Spektrum* 22 (2000) 501–524.
- [3] P. Brucker, S. Knust, Resource-constrained project scheduling and timetabling, in: *International conference on the practice and theory of automated timetabling*, Springer, 2000, pp. 277–293.
- [4] G. Simonin, C. Artigues, E. Hebrard, P. Lopez, Scheduling scientific experiments for comet exploration, *Constraints* 20 (2015) 77–99.
- [5] C. Ansótegui, M. Bofill, M. Palahí, J. Suy, M. Villaret, Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem, in: *Proceedings of the 9th Symposium on Abstraction, Reformulation and Approximation (SARA 2011)*, 2011, pp. 2–9.
- [6] M. Bofill, M. Palahí, J. Suy, M. Villaret, Solving constraint satisfaction problems with sat modulo theories, *Constraints* 17 (2012) 273–303.
- [7] H. Jiang, F. Yang, C. Yan, X. Zeng, Sat-based scheduling algorithm for high-level synthesis considering resource sharing, in: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 3244–3248. doi:10.1109/ISCAS48785.2022.9937754.
- [8] E. Demirović, N. Musliu, F. Winter, Modeling and solving staff scheduling with partial weighted maxsat, *Annals of Operations Research* 275 (2019) 79–99.
- [9] M. Bofill, J. Coll, J. Suy, M. Villaret, Smt encodings for resource-constrained project scheduling problems, *Computers & Industrial Engineering* 149 (2020) 106777. doi:https://doi.org/10.1016/j.cie.2020.106777.
- [10] J. Nordström, On the interplay between proof complexity and sat solving, *ACM SIGLOG News* 2 (2015) 19–44.
- [11] M. L. Bonet, J. L. Esteban, N. Galesi, J. Johannsen, On the relative complexity of resolution refinements and cutting planes proof systems, *SIAM Journal on Computing* 30 (2000) 1462–1484.
- [12] J. Elffers, J. Giráldez-Cru, J. Nordström, M. Vinyals, Using combinatorial benchmarks to probe the reasoning power of pseudo-boolean solvers, in: *International Conference on Theory and Applications of Satisfiability Testing*, Springer, 2018, pp. 75–93.
- [13] P. Abate, R. Di Cosmo, G. Gousios, S. Zacchiroli, Dependency solving is still hard, but we are getting better at it, in: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2020, pp. 547–551.
- [14] M. Vinyals, J. Elffers, J. Giráldez-Cru, S. Gocht, J. Nordström, In between resolution and cutting planes: A study of proof systems for pseudo-boolean sat solving, in: *International Conference on Theory and Applications of Satisfiability Testing*, Springer, 2018, pp. 292–310.

- [15] S. H. Dike, Project scheduling with resource constraints, *IEEE Transactions on Engineering Management* (1964) 155–157.
- [16] J. D. Wiest, Some properties of schedules for large projects with limited resources, *Operations research* 12 (1964) 395–418.
- [17] E. W. Davis, Resource allocation in project network models-a survey, *Journal of Industrial Engineering* 17 (1966) 177–188.
- [18] O. Icmeli, S. Selcuk Erenguc, C. J. Zappe, Project scheduling problems: A survey, *International Journal of Operations & Production Management* 13 (1993) 80–91.
- [19] C. Montoya, O. Bellenguez-Morineau, E. Pinson, D. Rivreau, Branch-and-price approach for the multi-skill project scheduling problem, *Optimization Letters* 8 (2014) 1721–1734.
- [20] O. Bellenguez-Morineau, Methods to solve multi-skill project scheduling problem, *4OR* 6 (2008) 85–88.
- [21] C. E. M. Casas, New methods for the multi-skills project scheduling problem, Nantes, France, 2012. PhD thesis, Automatic Control Engineering. NNT: 2012EMNA0077, TEL: tel-00789769.
- [22] K. D. Young, T. Feydy, A. Schutt, Constraint programming applied to the multi-skill project scheduling problem, in: *International conference on principles and practice of constraint programming*, Springer, 2017, pp. 308–317.
- [23] J. C. Caballero, Scheduling through logic-based tools, *Constraints* 28 (2023) 510. doi:10.1007/s10601-023-09357-0.
- [24] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021.
- [25] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, *Communications of the ACM* 5 (1962) 394–397.
- [26] M. Davis, H. Putnam, A computing procedure for quantification theory, *Journal of the ACM (JACM)* 7 (1960) 201–215.
- [27] J. P. Marques-Silva, K. A. Sakallah, Grasp: A search algorithm for propositional satisfiability, *IEEE Transactions on computers* 48 (2002) 506–521.
- [28] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient sat solver, in: *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.
- [29] A. Biere, A. Fröhlich, Evaluating cdcl restart schemes, *Proceedings of Pragmatics of SAT* (2015) 1–17.
- [30] T. Krüger, J.-H. Lorenz, F. Würz, Too much information: Why cdcl solvers need to forget learned clauses, *Plos one* 17 (2022) e0272967.
- [31] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM (JACM)* 12 (1965) 23–41.
- [32] P. L. Hammer, S. Rudeanu, *Boolean methods in operations research and related areas*, Springer Science & Business Media, 2012.
- [33] F. Barahona, M. Grötschel, M. Jünger, G. Reinelt, An application of combinatorial optimization to statistical physics and circuit layout design, *Operations Research* 36 (1988) 493–513.
- [34] D. Laughhunn, D. Peterson, Computational experience with capital expenditure programming models under risk, *J. Business Finance* 3 (1971) 43–48.
- [35] Y. Crama, J. B. Mazzola, Valid inequalities and facets for a hypergraph model of the nonlinear knapsack and the fms part selection problems, *Annals of Operations Research* 58 (1995) 99–128.
- [36] M. Bofill, J. Coll, J. Suy, M. Villaret, Sat encodings of pseudo-boolean constraints with at-most-one relations, in: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2019, pp. 112–128.
- [37] M. Bofill, J. Coll, J. Suy, M. Villaret, An mdd-based sat encoding for pseudo-boolean constraints with at-most-one relations, *Artificial Intelligence Review* 53 (2020) 5157–5188.
- [38] M. Bofill, J. Coll, P. Nightingale, J. Suy, F. Ulrich-Oltean, M. Villaret, Sat encodings for pseudo-boolean constraints together with at-most-one constraints, *Artificial Intelligence* 302 (2022) 103604.
- [39] Nonfiction Software, EXACT: An exact pseudo-boolean solver, <https://gitlab.com/>

nonfiction-software/exact, 2025. GitLab repository, accessed January 2026.

- [40] J. Elffers, J. Nordström, Divide and conquer: Towards faster pseudo-boolean solving., in: IJCAI, volume 18, 2018, pp. 1291–1299.
- [41] D. Le Berre, A. Parrain, The Sat4j Library, Release 2.2: System Description, Journal on Satisfiability, Boolean Modelling and Computation 7 (2011) 59–64. doi:10.3233/SAT190075.
- [42] R. L. M. Nieuwenhuis, A. Oliveras Llunell, E. Rodríguez Carbonell, R. Zhao, Speeding up pseudo-boolean propagation, in: SAT2024: International Conference on Theory and Applications of Satisfiability Testing: Sahyadri Park, TCS Campus, Pune, India, August 21-24, 2024, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2024.
- [43] W. Cook, C. R. Coullard, G. Turán, On the complexity of cutting-plane proofs, Discrete Applied Mathematics 18 (1987) 25–38.
- [44] C. Ansótegui, M. L. Bonet, J. Levy, Solving (weighted) partial maxsat through satisfiability testing, in: International conference on theory and applications of satisfiability testing, Springer, 2009, pp. 427–440.
- [45] E. Boros, P. L. Hammer, Pseudo-boolean optimization, Discrete applied mathematics 123 (2002) 155–225.
- [46] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research 90 (1996) 320–333.
- [47] I. P. Gent, Arc consistency in sat, in: ECAI, volume 2, 2002, pp. 121–125.
- [48] S. Gocht, J. Nordström, Certifying parity reasoning efficiently using pseudo-boolean proofs, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 3768–3777.
- [49] D. Le Berre, R. Wallon, On dedicated cdcl strategies for pb solvers, in: International Conference on Theory and Applications of Satisfiability Testing, Springer, 2021, pp. 315–331.
- [50] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints, in: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP), Springer, 2013, pp. 80–96.
- [51] M. Bofill, J. Giráldez-Cru, J. Suy, M. Villaret, A study on implied constraints in a maxsat approach to B2B problems, in: Proceedings of the 22nd International Conference of the Catalan Association for Artificial Intelligence, CCIA 2019, Frontiers in Artificial Intelligence and Applications, IOS Press, 2019, pp. 183–192. URL: <https://doi.org/10.3233/FAIA190123>. doi:10.3233/FAIA190123.
- [52] M. Bofill, J. Coll, J. Giráldez-Cru, J. Suy, M. Villaret, The impact of implied constraints on maxsat B2B instances, Int. J. Comput. Intell. Syst. 15 (2022) 63. URL: <https://doi.org/10.1007/s44196-022-00121-5>. doi:10.1007/s44196-022-00121-5.
- [53] A. M. Frisch, C. Jefferson, I. Miguel, Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern, in: ECAI, volume 16, 2004, p. 171.
- [54] F. Avellaneda, A short description of the solver evalmaxsat, MaxSAT Evaluation 8 (2020) 364.
- [55] C.-M. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, K. He, Boosting branch-and-bound maxsat solvers with clause learning, AI Communications 35 (2022) 131–151.
- [56] G. S. Tseitin, On the complexity of derivation in propositional calculus, in: Automation of reasoning: 2: Classical papers on computational logic 1967–1970, Springer, 1983, pp. 466–483.
- [57] I. Abío, P. J. Stuckey, Encoding linear constraints into sat, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2014, pp. 75–91.