IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 6, 2021, accepted October 13, 2021, date of publication October 15, 2021, date of current version October 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3120597

Employee Scheduling With SAT-Based Pseudo-Boolean Constraint Solving

ROBERT NIEUWENHUIS, ALBERT OLIVERAS, ENRIC RODRÍGUEZ-CARBONELL, AND EMMA ROLLON

Department of Computer Science, Technical University of Catalonia (UPC), 08034 Barcelona, Spain

Corresponding author: Enric Rodríguez-Carbonell (erodri@cs.upc.edu)

Project RTI2018-094403-B-C33 funded by MCIN (Ministerio de Ciencia e Innovación)/AEI/10.13039/501100011033/FEDER "Una manera de hacer Europa."

ABSTRACT The aim of this paper is practical: to show that, for at least one important real-world problem, modern SAT-based technology can beat the extremely mature branch-and-cut solving methods implemented in well-known state-of-the-art commercial solvers such as CPLEX or Gurobi. The problem of employee scheduling consists in assigning a work schedule to each of the employees of an organization, in such a way that demands are met, legal and contractual constraints are respected, and staff preferences are taken into account. This problem is typically handled by first modeling it as a 0-1 integer linear program (ILP). Our experimental setup considers as a case study the 0-1 ILPs obtained from the staff scheduling of a real-world car rental company, and carefully compares the performance of CPLEX and Gurobi with our own simple conflict-driven constraint-learning pseudo-Boolean solver.

INDEX TERMS Employee scheduling, 0-1 integer linear program, propositional satisfiability.

I. INTRODUCTION

In essence, the problem of *employee scheduling* (also known as *staff*, *workforce*, *personnel scheduling* or *rostering*) consists in assigning a work schedule to each of the employees of an organization, in such a way that predicted demands are met. Typically valid schedules also have to satisfy several legal and contractual constraints regarding, e.g., the daily worked hours. Other restrictions are often taken into account as well for the sake of the well-being of the employees, such as staff preferences or limitations on work on evenings and weekends.

The interest in the problem stems from the fact that the quality of the schedules of a company has a major impact on its efficiency, which is key in today's highly competitive environment. By adapting better to the customer demands and improving workforce capacity utilization, production can be increased and/or labor costs can be reduced. For instance, a company may gain an edge over the competition by shortening overtime, which can be expensive for the business and disturbing for employees [33]. Moreover, when carried out by hand, scheduling also involves tedious administration and significant indirect costs, as human resources managers may have to spend valuable countless hours on trying to find schedules that satisfy all of the requirements. Last but not least, rosters heavily influence the health and personal life of the staff [21]. For example, the sleep/wake cycle may be disrupted by exposure to light and activity at night, a time when the human body is biologically programmed to be in darkness and sleep. This leads to a desynchronization of the rhythmic body functions (hormones, body temperature, digestion, ...) with the circadian rhythm, which eventually can cause health problems such as fatigue during work and sleep problems after night shifts [3]. Additionally, working at times that interfere with the social personal development, particularly evenings and weekends, is likely to impair work-life balance [40]. Even setting aside ethical considerations, since employee's satisfaction in their work place is directly linked to their engagement and the ultimate success of their activity, from the perspective of the business the design of appropriate schedules is beneficial for service quality. Furthermore, obviously employee's contentment also plays an important role in employee retention, which from the economic point of view is also worth considering: failing to retain the staff can be an expensive problem for an organization since, as some studies have shown, the total cost of turnover can reach up to 90-200 % of the employee's annual salary [8].

For all these reasons, automated employee scheduling has been investigated since as early as the 1950's. Starting with the seminal work by Edie [16] and Dantzig [10], the problem

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

has received a great deal of attention and has been applied to numerous contexts such as airlines, health care systems, police, call centres and retail stores [20], [38]. As a result of this research work, a vast bibliography has been produced in the area; e.g., a comprehensive review of about 700 references can be found in [19].

One of the reasons for this extensive literature is that each real-world application has its own particularities and legal conditions that make it necessary to design a tailored model. Just to give a glimpse of the intricacies of employee scheduling, for instance in Spain as of 2019 there were about 2600 different collective labor agreements. Also, in each application the availability of computational resources may be different, which in turn determines the appropriateness of the techniques for generating the schedules. While in some contexts it is permissible that this process takes hours, in other situations any execution that goes beyond few minutes is unacceptable, because for example the scheduler needs to take care of last-minute changes such as sudden leaves of absence or updates on the predicted demands.

Another factor that explains the wide range of techniques that are available in the literature is the computational complexity of the problem (e.g., nurse rostering, a particular case of employee scheduling, is well-known to be NP-hard [27]). Due to this high complexity, heuristic methods are commonly used (simulated annealing, tabu search, genetic algorithms, etc.). Here we focus on *complete* methods, those that with sufficient time (and memory) ensure that:

- If the instance is feasible (i.e., some solution satisfying all constraints exists), they always eventually find the optimal solution.
- If the instance is infeasible (no solution exists), they always eventually prove it, and moreover they are able to give a simplified ''reason'' for that infeasibility (i.e., a –frequently small– subset of the input constraints that is already infeasible by itself). This may be useful, e.g., in order to identify when and why a problem specification does not allow any valid schedule, which has recently been raised as an important issue by the community [21].

Moreover, the complete methods that will be considered in this work have the property that, while running, they can report about the progress they are making, indicating:

- the cost of the best solutions found so far;
- a *lower bound* for the optimal cost (the minimal cost that any solution must have);
- the ''gap'' between both, that is, the remaining room for improvement, which gets smaller while solutions improve (their cost gets lower) and the lower bound rises as it gets more precise.

All these guarantees of the complete methods are obviously crucial in many real-world employee scheduling applications. On the other hand, incomplete heuristic approaches are sometimes able to find good solutions quickly, but they typically offer no such guarantees: if the instance is infeasible, they cannot prove it, nor pinpoint any reason for it, they cannot ensure that the generated solution is optimal, and they usually give no lower bounds either.

In the (complete) methods considered in this paper, the employee scheduling problem instance is first (automatically) translated into an integer linear program (ILP), and more specifically as a 0-1 ILP, i.e., one in which all variables are Boolean. Traditionally, 0-1 ILPs are handled with a solver based on the simplex algorithm and branch-and-cut [37]. It has to be highlighted that the technology behind this kind of solvers is extremely mature, after decades of improvements: according to [7], between 1991 and 2017 they have experienced a 1.3 million times speedup from *machine-independent* algorithmic improvements only (that is, a factor of 1.8 per year)!

In parallel, in the last two decades solvers for the *propositional satisfiability* (SAT) problem [6] have also made a breathtaking progress, so tremendous that it is already being referred to as the *SAT revolution* [39]. Just to illustrate the capabilities of state-of-the-art SAT solvers, nowadays they routinely deal with huge formulas coming from real-world applications with millions of variables and clauses (and leading developers foresee that, in the short term, handling formulas with thousands of millions of variables will be possible [5]). It is therefore of no surprise that SAT solvers and their extensions [32] have emerged as a serious alternative to simplex-based integer linear programming tools when solving NP-complete problems, in particular those of a combinatorial and logical (rather than numerical) nature.

Following this idea, in this paper we focus on the staff scheduling of a car rental company as a case study. We model the problem as a 0-1 ILP, and then we consider two complete methods for solving the resulting 0-1 ILPs:

- branch-and-cut-based solving, as in the commercial solvers **CPLEX**[1] and **Gurobi**[2];
- conflict-driven, constraint learning pseudo-Boolean solving, inspired by modern SAT solving techniques.

To the best of our knowledge, for this problem no other complete approach (SMT, direct encodings into SAT) is competitive with these two (see Section II-D below why). The main contribution in this work is the experimental observation that, for this particular application, a pseudo-Boolean solver can outperform state-of-the-art commercial Operations Research software. This is a remarkable result, since up to now, it was strongly believed in the community that commercial MIP solvers such as Gurobi and CPLEX are much more powerful than the current pseudo-Boolean technology [14].

The paper is structured as follows. After reviewing the background on 0-1 ILPs and solving techniques in Section II, in Section III we describe in detail the employee scheduling problem we aim at solving. A 0-1 ILP model for this problem is given in Section IV. The results of the experimental

[1] www.ibm.com/products/ilog-cplex-optimization-studio
[2] www.gurobi.com

comparison between the different algorithmic alternatives are shown and discussed in Section V. Finally, Section VI presents the conclusions and sketches some ideas for future work.

II. PRELIMINARIES

A. 0-1 INTEGER LINEAR PROGRAMS

Let X be a finite set of *integer variables* $\{x_1 \dots x_n\}$. An *(integer linear) constraint* over X is an expression of the form $a_1\, x_1 + \cdots + a_n x_n \geq a_0$ where, for all i in $0 \dots n$, the coefficients a_i are integers.

An *integer linear program (ILP)* over X is a set S of integer linear constraints over X, called the *constraints* of the ILP, together with a linear expression of the form $c(x_1, \cdots, x_n) = c_1\, x_1 + \cdots + c_n x_n$, called the *cost function*. A *solution* to a set of constraints S over X (and, by extension, to an ILP whose set of constraints is S) is a function $\sigma : X \to \mathbb{Z}$ that *satisfies* every constraint $a_1\, x_1 + \cdots + a_n x_n \geq a_0$ in S, that is, $a_1\, \sigma(x_1) + \cdots + a_n\, \sigma(x_n) \geq_{\mathbb{Z}} a_0$. A *minimal solution* to an ILP with constraints S and cost function c is a solution σ to S such that $c(\sigma) \leq c(\sigma')$ for any solution σ' to S. The problem of *Integer Linear Programming* consists in finding a minimal solution to a given integer linear program.

A variable x is *0-1* (also called *binary* or *Boolean*) in an ILP if its set of constraints S contains the constraints $0 \leq x$ and $x \leq 1$, so that effectively x can only take values either 0 or 1. An ILP such that all variables are 0-1 is a *0-1 linear program*, and the problem of finding a minimal solution in this case is referred to as *0-1 Linear Programming*. Constraints are then called $0 - 1$ *constraints* or *(linear) pseudo-Boolean constraints*.

Traditionally 0-1 linear programs have been handled with generic Operations Research ILP software packages using *branch-and-cut*. The idea of this procedure is to drop the integrality constraints (thus obtaining the so-called *linear programming relaxation* of the problem) and apply the simplex algorithm on it. If the optimal solution to the relaxation is integral, then the search is over. Otherwise a *cut* inequality that discards the spurious solution may be added to tighten the relaxation, and the simplex algorithm is applied again. If it is considered that adding cuts is no longer useful, eventually *branching* is performed: a variable which is assigned a non-integer value is chosen and the problem is split in two by adding a new bound on this variable. A more thorough description of the algorithm of branch-and-cut can be found, e.g., in [37].

B. SAT SOLVING

The problem of SAT consists in finding solutions to sets of *clauses* of the form

$$x_1 \vee \ldots \vee x_m \vee \neg y_1 \vee \ldots \vee \neg y_n,$$

that is, finding 0-1 values for the variables such that for each such a clause some x_i is 1 or some y_j is 0. Note that such a clause is equivalent to a $0 - 1$ constraint

$$x_1 + \ldots + x_m + (1 - y_1) + \ldots + (1 - y_n) \geq 1.$$

The underlying algorithm of modern SAT solvers is the *Davis-Putnam-Logemann-Loveland (DPLL)* procedure [11], [12]. In short, DPLL is a backtracking algorithm that searches for a solution by smartly exploring the search space. 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*, and variables that are forced to a value are detected. 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, then the previous decision is backtracked. If the decision is the first one, and so 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 a number of techniques:

- *conflict analysis* and *backjumping* (i.e., non-chronological backtracking) [28];
- *learning* (that is, addition) of new clauses generated from conflicts [13];
- variable decision heuristics that select the most active variables in recent conflicts, like *VSIDS* [30];
- value decision heuristics that select promising values for the decision variable, such as the *last phase* strategy [34];
- data structures such as the *2-watched literal scheme* [30] for efficiently identifying propagations and conflicts;
- *restarts* [23];
- *in-processing* and simplification of learnt clauses [26];
- *clause cleanups* that periodically delete counterproductive learnt clauses, e.g. based on their activity in conflicts [22] or their *literal block distance* [4].

C. PSEUDO-BOOLEAN SOLVING

Given a 0-1 ILP, its minimal solution can be found by iteratively solving a sequence of satisfiability problems, typically successively strengthened by enforcing that a better solution should exist. However, there is a complication if these satisfiability problems are to be solved with a SAT solver: in the language of SAT the only primitive constraints are clauses, and the pseudo-Boolean constraints in these problems cannot be directly handled.

A possibility to circumvent this situation is to extend the SAT techniques so that linear constraints become first-class citizens in the language of the solver and can be dealt with natively. Among other things, this requires to adapt the propagation mechanism, as well as the algorithms and data structures for quickly detecting when a propagation can be triggered or a conflict has arisen. This is not trivial, because

although it is possible to extend the watched literal scheme from CDCL SAT, unlike clauses a pseudo-Boolean constraint may need to watch more than 2 literals, which complicates an efficient implementation.

Most importantly, conflict analysis has to be generalized in such a way that the backjumping and learning of the CDCL scheme are possible. Conflict analysis in SAT is based on resolution [35], which can be extended to pseudo-Boolean constraints with the *generalized resolution* rule [15], [25]. Unfortunately, a direct replacement of resolution by generalized resolution in conflict analysis breaks the invariants of the algorithms [6]. In order to overcome this problem, constraints arising in conflict analysis have to be processed, for example with *weakening* and *saturation* [9] or *rounding* [18].

From now on we will refer to solvers of 0-1 ILPs under this framework, which is the one that we will follow in this work, as *(CDCL) pseudo-Boolean solvers*.

D. OTHER COMPLETE SAT-BASED METHODS FOR 0-1 ILP SOLVING

There exist alternative approaches for handling 0-1 ILPs with SAT-related techniques. A possibility is to eagerly encode linear constraints into SAT by adding new clauses and possibly new variables in such a way that the set of solutions is preserved, for instance using sorting networks, adder networks or binary decision diagrams [1], [17], [29], [36]. However, the number of clauses that are needed for expressing the pseudo-Boolean constraints may be so large that the resulting formula becomes impractical.

Another technique for dealing with pseudo-Boolean constraints is to apply a lazy approach using SMT [2], [32]. Under this paradigm, first of all each pseudo-Boolean constraint is translated into an equivalent clause if possible. After this preprocessing step, a SAT solver, called the *engine*, produces assignments that satisfy the clauses. For each of the remaining constraints a *theory solver* is defined, which certifies given an assignment whether or not the associated constraint holds for that assignment. While this method turns out to be very efficient when the number of non-clausal pseudo-Boolean constraints is small, when this is not the case the overhead of handling each constraint individually with a theory solver becomes prohibitive.

III. PROBLEM DESCRIPTION

The work presented in this article was developed in the context of a proof of concept of Barcelogic (https://barcelogic.com), a spin-off of the Technical University of Catalonia, for a car rental company.

The car rental company operates in the main airports and train stations in Spain. For every day of the year and for the time slots:

- from 7 to 9,
- from 9 to 11,
- from 11 to 13,
- from 13 to 15,
- from 15 to 17,
- from 17 to 19,
- from 19 to 21, and
- from 21 to 24,

the company has an estimation of the number of services (collection or delivery) that should be handled based on historical data and statistical models. The months from April to September are of particular interest to the company, since in spring and summer tourism and therefore car rental is at its highest; as an illustration, see Figure 1 for a representative sample of the estimations on a day in February, July and November. In order to provide the services the company has a permanent staff, and if needed it can also hire temporary workers on a monthly basis. If a temporary worker is ever hired, there is a once-and-forall cost to be paid (recruiting and training expenses, etc.). To approximate the minimum number of workers that should be available at each slot, the company has computed a productivity measure that counts the number of services that an employee can process per hour.

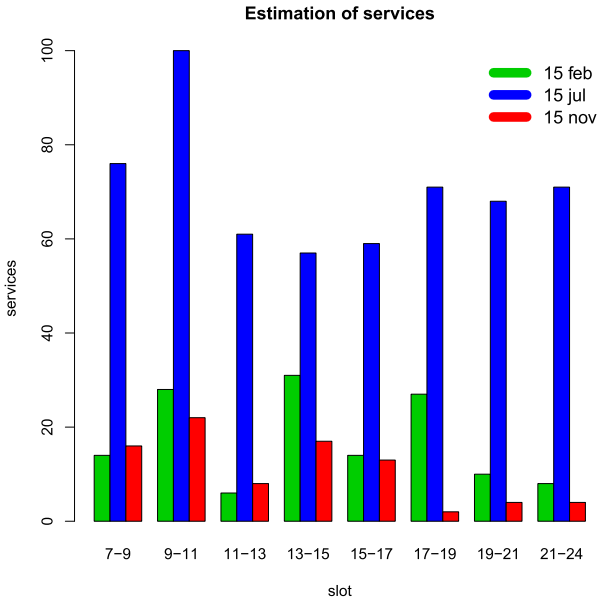


FIGURE 1. Estimation of the number of services on a day of February, July and November.

Apart from ensuring that the demand is met, schedules have to satisfy several legal constraints. For example, according to the collective labor agreement and to the contract, there is a minimum and a maximum number of minutes that an employee can work the same day, and a maximum number of minutes of *daily presence*, that is, the time an employee can be at the work place daily (including breaks). There is also a minimum and a maximum for the number of minutes of consecutive work, i.e., in which the employee works without interruption. Finally, each contract has a different monthly cost.

Daily time slots are divided into morning shift (slots ending not later than 13) and afternoon shift (slots starting not sooner than 13). On a particular day, an employee can only work in the morning shift or in the afternoon shift. Moreover, for the sake of fairness, for each window of a certain prefixed number of consecutive days, the (absolute) difference between days

with a morning shift and days with an afternoon shift should be limited.

There are other restrictions that constrain the schedule of consecutive days. In particular, there is a maximum on the number of days that an employee can work consecutively, that is, without a rest day in between. Similarly, there is a maximum on the number of days that an employee can rest consecutively, i.e., without a work day in between.

Finally, workers can ask for a leave of daily absence. The company foresees that there may be a non-negligible number of these leaves and that they may be communicated on a very short notice. Also, the estimation of the number of services for each daily time slot is regularly being updated. The need for these updates has become palpable in 2020 and 2021, with the outbreak of the COVID-19 pandemic and the ever-changing restrictions on mobility that still exist as of the date of this writing (June 2021). Altogether, it is necessary that schedules can be recomputed efficiently in few minutes.

IV. 0-1 ILP MODEL

In this section we will describe a 0-1 ILP model for our problem. To that end, let us introduce the sets:

$\mathcal{E}$:	set of employees
$\mathcal{S}$:	set of slots (possibly of different length)
$\mathcal{MS}$:	set of morning slots
$\mathcal{AS}$:	set of afternoon slots
$\mathcal{D}$:	set of (consecutive) days
$\mathcal{M}$:	set of (consecutive) months

Slots and days will sometimes be viewed as chronologically ordered integer identifiers, so that $s+1$ and $d+1$ are the slot and the day that follow slot s and day d, respectively. Note also that the morning and afternoon slots form a partition of the set of slots, i.e., $\mathcal{MS} \cup \mathcal{AS} = \mathcal{S}$ and $\mathcal{MS} \cap \mathcal{AS} = \emptyset$. Moreover, when it is convenient we will consider a month $m \in \mathcal{M}$ as the sets of days it consists of, so that e.g. $\forall d \in m$ ranges over all days of month m.

In order to formulate our model, let us also consider the following input parameters, all of which are natural numbers except for the last item, which are Boolean values.

Req_{sd}:	number of employees that are requested
($\mathbb{N}$)	at slot s on day d for each $s \in \mathcal{S}$ and $d \in \mathcal{D}$
$MinW_e$:	minimum of minutes of daily work
($\mathbb{N}$)	of employee e for each $e \in \mathcal{E}$
$MaxW_e$:	maximum of minutes of daily work
($\mathbb{N}$)	of employee e for each $e \in \mathcal{E}$
$MaxP_e$:	maximum of minutes of daily presence
($\mathbb{N}$)	of employee e for each $e \in \mathcal{E}$
$MinC_e$:	minimum of minutes of consecutive work
($\mathbb{N}$)	of employee e for each $e \in \mathcal{E}$
$MaxC_e$:	maximum of minutes of consecutive work
($\mathbb{N}$)	of employee e for each $e \in \mathcal{E}$
Per:	length of period of consecutive days used when
($\mathbb{N}$)	comparing morning and afternoon shifts
$MaxD$:	every window of Per days, the difference of
($\mathbb{N}$)	morning and afternoon shifts is at most $MaxD$
$MaxCW_e$:	maximum number of days that employee e
($\mathbb{N}$)	can work consecutively for each $e \in \mathcal{E}$
$MaxCR_e$:	maximum number of days that employee e
($\mathbb{N}$)	can rest consecutively for each $e \in \mathcal{E}$
Wm_{em}:	cost to be paid if employee e works in month m
($\mathbb{N}$)	for each $e \in \mathcal{E}$ and $m \in \mathcal{M}$
W_e:	cost to be paid if employee e ever works
($\mathbb{N}$)	for each $e \in \mathcal{E}$
Abs_{ed}:	employee e is absent on day d
($\mathbb{B}$)	for each $e \in \mathcal{E}$ and $d \in \mathcal{D}$

Note that the demand of workforce is expressed with the input parameters Req_{sd}, which are assumed to be precomputed from the estimation of requested services per slot and the number of services an employee can handle per hour.

We now introduce the following 0-1 decision variables:

wsd_{esd}:	employee e works at slot s on day d for each $e \in \mathcal{E}$, $s \in \mathcal{S}$ and $d \in \mathcal{D}$
wd_{ed}:	employee e works on day d for each $e \in \mathcal{E}$ and $d \in \mathcal{D}$
ms_{ed}:	employee e works on day d with morning shift for each $e \in \mathcal{E}$ and $d \in \mathcal{D}$
as_{ed}:	employee e works on day d with afternoon shift for each $e \in \mathcal{E}$ and $d \in \mathcal{D}$
wm_{em}:	employee e works in month m for each $e \in \mathcal{E}$ and $m \in \mathcal{M}$
w_e:	employee e works for each $e \in \mathcal{E}$

Observe that while input parameter names start with an upper-case letter, variable names start with lower-case.

Now we are ready to show our model for the problem of finding a feasible schedule for employees $\mathcal{E}$ spanning the months $\mathcal{M}$ and minimizing labor costs:

$$\min \sum_{e \in \mathcal{E}} W_e \, w_e + \sum_{\substack{e \in \mathcal{E} \\ m \in \mathcal{M}}} Wm_{em} \, wm_{em}$$

$$\text{subject to } \sum_{e \in \mathcal{E}} wsd_{esd} \geq Req_{sd} \quad \forall s \in \mathcal{S}, d \in \mathcal{D} \tag{1}$$

$$\neg wsd_{esd} \vee wd_{ed} \quad \forall e \in \mathcal{E}, s \in \mathcal{S}, d \in \mathcal{D} \tag{2}$$

$$\neg wd_{ed} \vee wm_{em} \quad \forall e \in \mathcal{E}, m \in \mathcal{M}, d \in m \tag{3}$$

$$\neg wm_{em} \vee w_e \quad \forall e \in \mathcal{E}, m \in \mathcal{M} \tag{4}$$

$$\neg wd_{ed} \quad \forall e \in \mathcal{E}, d \in \mathcal{D}. Abs_{ed} = 1 \tag{5}$$

$$\neg wsd_{esd} \vee ms_{ed} \quad \forall e \in \mathcal{E}, s \in \mathcal{MS}, d \in \mathcal{D} \tag{6}$$

$$\neg wsd_{esd} \vee as_{ed} \quad \forall e \in \mathcal{E}, s \in \mathcal{AS}, d \in \mathcal{D} \tag{7}$$

$$\neg ms_{ed} \vee \neg as_{ed} \quad \forall e \in \mathcal{E}, d \in \mathcal{D} \tag{8}$$

$$\sum_{k=0}^{Per-1} ms_{e\,d+k} - as_{e\,d+k} \leq MaxD$$

$$\forall e \in \mathcal{E}, \{d, d+1, \ldots, d+Per-1\} \subseteq \mathcal{D} \quad (9)$$
$$\sum_{k=0}^{Per-1} ms_{e\ d+k} - as_{e\ d+k} \geq -MaxD$$

$$\forall e \in \mathcal{E}, \{d, d+1, \ldots, d+Per-1\} \subseteq \mathcal{D} \quad (10)$$
$$\neg wsd_{esd} \vee \neg wsd_{es'd}$$

$$\forall e \in \mathcal{E}, s, s' \in \mathcal{MS}, d \in \mathcal{D}.\ \mathrm{e}(s') - \mathrm{s}(s) > MaxP_e \quad (11)$$
$$\neg wsd_{esd} \vee \neg wsd_{es'd}$$

$$\forall e \in \mathcal{E}, s, s' \in \mathcal{AS}, d \in \mathcal{D}.\ \mathrm{e}(s') - \mathrm{s}(s) > MaxP_e \quad (12)$$

$$\sum_{s \in \mathcal{MS}} \mathrm{mins}(s)\ wsd_{esd} \leq MaxW_e \quad \forall e \in \mathcal{E}, d \in \mathcal{D} \quad (13)$$

$$\sum_{s \in \mathcal{AS}} \mathrm{mins}(s)\ wsd_{esd} \leq MaxW_e \quad \forall e \in \mathcal{E}, d \in \mathcal{D} \quad (14)$$

$$\sum_{s \in \mathcal{S}} \mathrm{mins}(s)\ wsd_{esd} \geq MinW_e wd_{ed} \quad \forall e \in \mathcal{E}, d \in \mathcal{D} \quad (15)$$

$$wsd_{es_1d} \vee \neg wsd_{es_2d} \vee wsd_{es_3d} \quad \forall e \in \mathcal{E}, d \in \mathcal{D}, \forall s_1, s_2, s_3 \in \mathcal{S}.\ \mathrm{e}(s_1) = \mathrm{s}(s_2), \mathrm{e}(s_2) \leq \mathrm{s}(s_3), \mathrm{s}(s_3) - \mathrm{s}(s_2) < MinC_e \quad (16)$$

$$\bigvee_{k=0}^{K} \neg wsd_{e\ s+k\ d} \quad \forall e \in \mathcal{E}, d \in \mathcal{D}, \{s, s+1, \ldots, s+K\} \subseteq \mathcal{S}.\ \mathrm{e}(s+K-1) - \mathrm{s}(s) \leq MaxC_e < \mathrm{e}(s+K) - \mathrm{s}(s) \quad (17)$$

$$\bigvee_{k=0}^{MaxCW_e} \neg wd_{e\ d+k} \quad \forall e \in \mathcal{E}, \{d, d+1, \ldots, d+MaxCW_e\} \subseteq \mathcal{D} \quad (18)$$

$$\bigvee_{k=0}^{MaxCR_e} wd_{e\ d+k} \quad \forall e \in \mathcal{E}, \{d, d+1, \ldots, d+MaxCR_e\} \subseteq \mathcal{D} \quad (19)$$

Some remarks about the notation used in the model above are in order. Some constraints are written as clauses both for highlighting the logical part of the problem as well as for the sake of clarity. Other notational conventions are that, given a time slot $s \in S$, the values $\mathrm{s}(s)$ and $\mathrm{e}(s)$ represent the start time and the end time of s, expressed as minutes since midnight. Therefore, in Constraints 11 and 12 the expression $\mathrm{e}(s') - \mathrm{s}(s)$ represents the minutes that lapse between the start of s and the end of s'. In particular, $\mathrm{e}(s) - \mathrm{s}(s)$ is the length of s in minutes, which we also denote by $\mathrm{mins}(s)$ in Constraints 13, 14, and 15 for readability.

Let us review the parts of the model and explain their intuitive meaning. The cost function is aimed at minimizing the aggregated labor costs over all employees, decomposed in a setup cost ($W_e w_e$) and a monthly cost ($Wm_{em} wm_{em}$). In practice the costs of the fixed staff have already been discounted, so it can be assumed that $W_e = Wm_{em} = 0$ for any $e \in \mathcal{E}$ who is permanent.

Constraints 1 ensure that the demand of employees is met at any time. Clauses 2, 3 and 4 activate the variables wd_{ed}, wm_{em} and w_e respectively in order to indicate that there has been daily, monthly or overall work. Leaves are encoded with Clauses 5 by forbidding employee e to work on day d if they must be absent on that day. Constraints 6 activate the variables ms_{ed} of the morning shift when employee e works on day d on a morning slot s; and symmetrically Constraints 7 activate the variables as_{ed} of the afternoon shift when employee e works on day d on an afternoon slot s. Variables ms_{ed} and as_{ed} also appear in Clauses 8, which exclude that an employee could work on the same day with a morning and an afternoon shift. Constraints 9 and 10 impose that, every Per consecutive days, the absolute difference of the number of days an employee works with a morning shift and with an afternoon shift is at most $MaxD$.

The next constraints restrain the daily schedule. Clauses 11 and 12 guarantee that the presence of an employee e is at most $MaxP_e$. Note that here it is used that morning and afternoon shifts are mutually exclusive. This property is also used in Constraints 13 and 14, which enforce that employee e works at most $MaxW_e$ minutes. As regards the minimum work per day, Constraints 15 encode the implication

$$wd_{ed} \rightarrow \sum_{s \in \mathcal{S}} \mathrm{mins}(s)\ wsd_{esd} \geq MinW_e$$

which expresses that, if employee e works on day d, then at least $MinW_e$ minutes should be worked.

The following constraints take care of consecutive work on a day. In Clauses 16, where s_1, s_2 and s_3 are slots in increasing order of time such that s_1 and s_2 are consecutive and $\mathrm{s}(s_3) - \mathrm{s}(s_2) < MinC_e$, the constraint can be equivalently viewed as an implication

$$(\neg wsd_{es_1d} \wedge wsd_{es_2d}) \rightarrow wsd_{es_3d}.$$

I.e., if e starts working at slot s_2 after having had a rest at the previous slot s_1, since s_2 and s_3 are too close in time, e should work at slot s_3 too. On the other hand, Clauses 17 express that employee e cannot work more than $MaxC_e$ minutes without a break: for any sequence of consecutive slots spanning more than $MaxC_e$ minutes, we enforce that e should rest in at least one of the slots. The condition $\mathrm{e}(s+K-1) - \mathrm{s}(s) \leq MaxC_e < \mathrm{e}(s+K) - \mathrm{s}(s)$ ensures that the sequence of consecutive slots $\{s, s+1, \ldots, s+K\}$ is the shortest one starting at s which covers more than $MaxC_e$ minutes.

Finally, Clauses 18 and 19 limit work and rest consecutive days. Namely, Clauses 18 express that employee e can work

at most $MaxCW_e$ consecutive days by imposing that in any interval of $MaxCW_e + 1$ days, at least one must be a rest day. Similarly, Clauses 19 ensure that employee e can rest at most $MaxCR_e$ consecutive days by imposing that in any interval of $MaxCR_e + 1$ days, at least one must be a work day.

V. EXPERIMENTS

This section is devoted to the experimental comparison of different tools for solving our employee scheduling problem with the 0-1 ILP model proposed in Section IV.

To that end, we have built a benchmark suite of 25 instances obtained from real data provided by the car rental company involved in this project. Each instance corresponds to a period of two months in spring-summer, from April to September. The reason for this is that, as pointed out in Section I, this season is the busiest time for tourism. As a consequence, the demand is at its highest point, the labor capacity of the permanent staff get close to its limit (or even overpassed), and scheduling becomes critical. Moreover, planning two months ahead gives the car rental company a time margin with which to react in case bottlenecks are detected and temporary workers have to be hired. To give an idea of the complexity of the instances, they all have about 21k (Boolean) variables and 65k constraints.

In this experimental evaluation we compare the following solvers:

1) Gurobi (v.9.1.2, latest version)
2) CPLEX (v.20.1.0, latest version)
3) PB

Gurobi and CPLEX are well-known 0-1 ILP solvers, and are widely considered to be representative of the state of the art in branch-and-cut solving. On the other hand, PB is our basic implementation of the CDCL-based integer linear programming solver described in [31] but specialized for 0-1 variables. It is worth noting that, while Gurobi and CPLEX are long-standing mature commercial tools (the first version of CPLEX dates back to 1988!), PB is a prototype for research purposes only. In fact, there are possibly other better pseudo-Boolean solvers that PB. The reason for not including other pseudo-Boolean solvers in these experiments is that our aim is not to compare PB with other solvers of the same kind, but rather to make a point on the competitiveness of conflict-driven constraint-learning pseudo-Boolean technology against state-of-the-art commercial branch-and-cut solvers.

All experiments reported next were carried out on a standard 3.00 GHz 8-core Intel i7-9700 desktop with 16 Gb of RAM. The time limit of all executions was set to 300 seconds of wall-clock time. Given that the car rental company has several centers to schedule at the same time, this was considered a reasonable choice. It is important to highlight that our implementation PB is *sequential* and only uses one core, while on the other hand Gurobi and CPLEX are run in *parallel* mode. Therefore, they use all of the *eight* cores that are available in the computer used for the experiments.

TABLE 1. Cost of the best solution and time required for finding it for each solver and for each instance.

Instance	Value (Time)		
	Gurobi	CPLEX	PB
a-apr-may	11400 (296.53)	39600 (104.42)	**10400 (191.00)**
a-may-jun	— (TO)	— (TO)	**21500 (178.36)**
a-jun-jul	— (TO)	— (TO)	**24200 (294.37)**
a-jul-aug	— (TO)	— (TO)	**21000 (263.49)**
a-aug-sep	— (TO)	39600 (246.45)	**20200 (116.28)**
b-apr-may	1900 (280.06)	5600 (286.27)	**1900 (44.49)**
b-may-jun	— (TO)	— (TO)	**10200 (151.76)**
b-jun-jul	— (TO)	— (TO)	**11200 (187.06)**
b-jul-aug	— (TO)	— (TO)	**8000 (299.67)**
b-aug-sep	— (TO)	— (TO)	**9900 (175.81)**
c-apr-may	1900 (12.98)	1900 (19.67)	**1900 (1.39)**
c-may-jun	4600 (17.60)	4600 (178.66)	**4600 (8.00)**
c-jun-jul	8000 (33.48)	8000 (163.40)	**8000 (21.52)**
c-jul-aug	**6000 (24.99)**	6000 (128.35)	6000 (36.66)
c-aug-sep	1200 (104.62)	2400 (297.40)	**1200 (16.75)**
d-apr-may	— (TO)	26800 (43.73)	**3100 (38.34)**
d-may-jun	— (TO)	— (TO)	**10900 (32.33)**
d-jun-jul	— (TO)	— (TO)	**12900 (144.43)**
d-jul-aug	— (TO)	— (TO)	**7000 (259.41)**
d-aug-sep	— (TO)	26800 (183.28)	**7800 (293.42)**
e-apr-may	1400 (124.36)	4300 (299.38)	**1400 (33.73)**
e-may-jun	— (TO)	— (TO)	**3800 (192.49)**
e-jun-jul	6500 (300.14)	— (TO)	**3800 (247.82)**
e-jul-aug	2600 (155.10)	— (TO)	**2600 (82.04)**
e-aug-sep	3100 (242.68)	— (TO)	**2600 (258.06)**

The results of the experiments are shown in Table 1. The first column identifies the instance, while the other three correspond to each of the solvers. There are two rows for

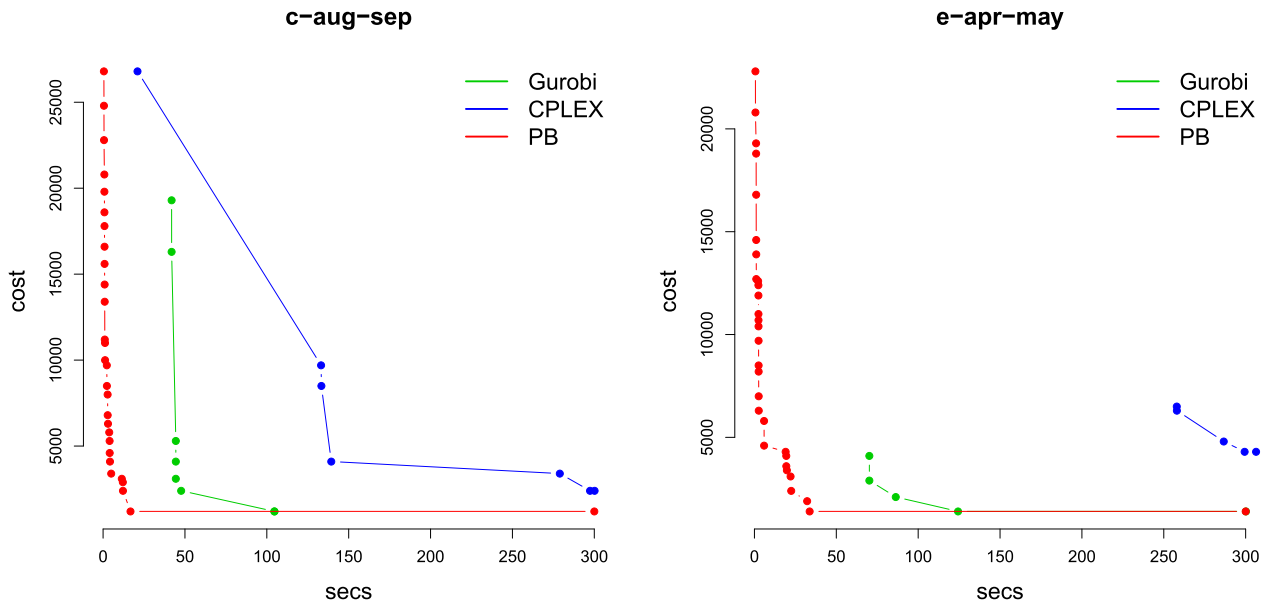


FIGURE 2. For a sample of instances, evolution of the cost of the best solution along time for each of the solvers. Lines not reaching the time limit of 300 s. indicate that optimality was proved.

each instance. In the top one we indicate the cost of the best solution found with each solver, and in the bottom one the time in seconds required for finding that solution (between parentheses). A dash means that the time limit was reached without any solution. The timing TO stands for time out. Over all solvers, the one with the best cost is highlighted in bold face (or in case of a tie, the one that found the solution with that cost the earliest).

The performance of PB is very good compared to the competition. As can be seen in Table 1, PB finds good solutions very efficiently, even in instances for which the other solvers do not report a solution within the timeout. Except for one instance, PB is the solver that gives the best results. One of the reasons for this success is that, as highlighted in the representation of the constraints in Section IV, the problem has an important logical component, which makes SAT-based techniques particularly appropriate.

In order to make a more precise analysis, in Figure 2 we show the graphs with the evolution of the cost of the best solution along time for two representative instances. These graphs reveal that PB finds many solutions before reaching the time limit, while Gurobi and CPLEX find far fewer. On the other hand, in both cases Gurobi is able to prove that its best solution is in fact optimal, which is indicated in the plot by the line not reaching the time limit. As regards proving optimality, PB does not appear to be as powerful as Gurobi, and once the optimal solution has been found, it gets stagnant and the time limit is reached before the search can be completed. This suggests that PB and Gurobi could be combined in a two-phase process, in which PB is run first for a little time and then Gurobi uses the best solution found with PB as a starting point. This idea requires further experimentation and is left as future work.

VI. CONCLUSION AND FUTURE WORK

We have shown that SAT-based tools for solving 0-1 ILPs can be competitive with commercial Operations Research software. We have illustrated it with the case study of a car rental company, for which we have provided a 0-1 ILP model that we have used as the basis for our experimental comparison. Since the problem that is addressed here does not show any significant differences from the typical employee scheduling problem, we foresee that the positive results that have been obtained can be also reproduced with other more general employee scheduling variants. For this reason, as future work we plan to enlarge the language for specifying schedules in order to incorporate new constraints, e.g., limiting the weekly, monthly and yearly worked time, as well as enforcing the fairness and balance of the schedules between employees. More broadly, we would also like to investigate other scheduling problems with a core combinatorial structure, for which SAT-based techniques may also outperform state-of-the-art Operations Research tools.

Last but not least, we also plan to improve our pseudo-Boolean solver and study how to complete the search and produce an optimality proof more efficiently, possibly by combining it with simplex relaxations as outlined above. Another idea for speeding up the performance is to take advantage of the nowadays widespread multicore architecture of computers and to implement a parallel solver, as commercial Operations Research tools do. As a first step, a portfolio solver where the workers share some of the constraints

(e.g., unit or binary clausal constraints) could be developed, following the current practice in pure SAT solving [24].

REFERENCES

[1] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, "A new look at BDDs for pseudo-Boolean constraints," *J. Artif. Intell. Res.*, vol. 45, pp. 443–480, Nov. 2012.

[2] I. Abio, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and P. J. Stuckey, "To encode or to propagate? The best choice for each constraint in SAT," in *Proc. 19th Int. Conf. Princ. Pract. Constraint Program.*, Berlin, Germany: Springer, 2013, pp. 97–106.

[3] J. Arendt, "Shift work: Coping with the biological clock," *Occupational Med.*, vol. 60, no. 1, pp. 10–20, Jan. 2010.

[4] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proc. 21st Int. Joint Conf. Artif. Intell. (IJCAI)*, C. Boutilier, Ed. Pasadena, CA, USA, Jul. 2009, pp. 399–404.

[5] A. Biere. (2021). *A Personal History of Practical SAT Solving*. 50 Years of Satisfiability: The Centrality of SAT in the Theory of Computing. [Online]. Available: https://simons.berkeley.edu/talks/tbd-308

[6] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds., "Handbook of satisfiability," in *Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands: IOS Press, 2009, vol. 185.

[7] B. Bixby, "Presentation: Progress in linear and mixed-integer programming," in *Proc. Joint EURO/ORSC/ECCO Conf. Combinat. Optim.*, May 2017, p. 14.

[8] W. F. Cascio, *Managing Human Resources: Productivity, Quality of Work Life, Profits*. New York, NY, USA: McGraw-Hill, 1995.

[9] D. Chai and A. Kuehlmann, "A fast pseudo-Boolean constraint solver," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 24, no. 3, pp. 305–317, Mar. 2005.

[10] B. G. Dantzig, "A comment on Edie's 'traffic delays at toll booths'" *J. Oper. Res. Soc. Amer.*, vol. 2, no. 3, pp. 339–341, 1954.

[11] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.

[12] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960.

[13] R. Dechter, "Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition," *Artif. Intell.*, vol. 41, no. 3, pp. 273–312, Jan. 1990.

[14] J. Devriendt, A. Gleixner, and J. Nordström, "Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search," *Constraints*, pp. 1–30, Jan. 2021, doi: 10.1007/s10601-020-09318-x.

[15] H. E. Dixon, M. L. Ginsberg, and A. J. Parkes, "Generalizing Boolean satisfiability I: Background and survey of existing work," *J. Artif. Intell. Res.*, vol. 21, pp. 193–243, Feb. 2004.

[16] L. C. Edie, "Traffic delays at toll booths," *J. Oper. Res. Soc. Amer.*, vol. 2, no. 2, pp. 107–138, May 1954.

[17] N. Eén and N. Sörensson, "Translating pseudo-Boolean constraints into SAT," *J. Satisfiability, Boolean Model. Comput.*, vol. 2, nos. 1–4, pp. 1–26, 2006.

[18] J. Elffers and J. Nordström, "Divide and conquer: Towards faster pseudo-Boolean solving," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 1291–1299.

[19] A. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier, "An annotated bibliography of personnel scheduling and rostering," *Ann. Oper. Res.*, vol. 127, no. 1, pp. 21–144, 2004.

[20] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *Eur. J. Oper. Res.*, vol. 153, no. 1, pp. 3–27, Feb. 2004.

[21] J. Gärtner, P. Bohle, A. Arlinghaus, W. Schafhauser, T. Krennwallner, and M. Widl, "Scheduling matters—Some potential requirements for future rostering competitions from a practitioner's view," in *Proc. 12th Int. Conf. Pract. Theory Automated Timetabling (PATAT)*, Vienna, Austria, Aug. 2018, pp. 33–42. [Online]. Available: http://www.patatconference.org/patat2018/files/proceedings/paper61.pdf

[22] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, 2002, pp. 142–149.

[23] C. P. Gomes, B. Selman, and N. Crato, "Heavy-tailed distributions in combinatorial search," in *Principles and Practice of Constraint Programming-CP97*, G. Smolka, Ed. Berlin, Germany: Springer, 1997, pp. 121–135.

[24] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: A parallel SAT solver," *J. Satisfiability, Boolean Model. Comput.*, vol. 6, no. 4, pp. 245–262, Jun. 2009.

[25] J. N. Hooker, "Generalized resolution for 0?1 linear inequalities," *Ann. Math. Artif. Intell.*, vol. 6, nos. 1–3, pp. 271–286, Mar. 1992.

[26] M. Järvisalo, M. J. H. Heule, and A. Biere, "Inprocessing rules," in *Automated Reasoning*, B. Gramlich, D. Miller, and U. Sattler, Eds. Berlin, Germany: Springer, 2012, pp. 355–370.

[27] H. C. Lau, "Manpower scheduling with shift change constraints," in *Algorithms and Computation*, D.-Z. Du and X.-S. Zhang, Eds. Berlin, Germany: Springer, 1994, pp. 616–624.

[28] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999.

[29] R. Martins, V. M. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.* in Lecture Notes in Computer Science, vol. 8561, C. Sinz and U. Egly, Eds. Vienna, Austria: Springer, Jul. 2014, pp. 438–445.

[30] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. 38th Conf. Design Autom. (DAC)*, 2001, pp. 530–535.

[31] R. Nieuwenhuis, "The intsat method for integer linear programming," in *Proc. 20th Int. Conf. Princ. Pract. Constraint Program. (CP)* in Lecture Notes in Computer Science, vol. 8656, B. O'Sullivan, Ed. Lyon, France: Springer, Sep. 2014, pp. 574–589.

[32] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, pp. 937–977, Nov. 2006.

[33] F. Pega, B. Náfrádi, N. C. Momen, Y. Ujita, K. N. Streicher, A. M. Prüss-Üstün, A. Descatha, T. Driscoll, F. M. Fischer, L. Godderis, H. M. Kiiver, J. Li, L. L. M. Hanson, R. Rugulies, K. Sørensen, and T. J. Woodruff, "Global, regional, and national burdens of ischemic heart disease and stroke attributable to exposure to long working hours for 194 countries, 2000–2016: A systematic analysis from the WHO/ILO joint estimates of the work-related burden of disease and injury," *Environ. Int.*, vol. 154, Sep. 2021, Art. no. 106595.

[34] K. Pipatsrisawat and A. Darwiche, "Rsat 2.0: Sat solver description," Comput. Sci. Dept., Automated Reasoning Group, UCLA, Los Angeles, CA, USA, Tech. Rep. D–153, 2007.

[35] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *J. ACM*, vol. 12, no. 1, pp. 23–41, Jan. 1965.

[36] M. Sakai and H. Nabeshima, "Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers," *IEICE Trans. Inf. Syst.*, vol. E98.D, no. 6, pp. 1121–1127, 2015.

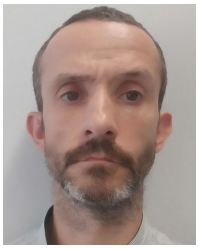
[37] A. Schrijver, *Theory of Linear and Integer Programming*. Hoboken, NJ, USA: Wiley, 1986.

[38] J. Van den Bergh, J. Beliën, P. D. Bruecker, E. Demeulemeester, and L. D. Boeck, "Personnel scheduling: A literature review," *Eur. J. Oper. Res.*, vol. 226, no. 3, pp. 367–385, 2013.

[39] M. Vardi. (2015). *The SAT Revolution: Solving, Sampling, and Counting*. Mathematical Colloquium. [Online]. Available: https://slideslive.com/38894537/the-sat-revolution-solving-sampling-and-counting

[40] A. Wirtz, O. Giebel, C. Schomann, and F. Nachreiner, "The interference of flexible working times with the utility of time: A predictor of social impairment?" *Chronobiol. Int.*, vol. 25, nos. 2–3, pp. 249–261, Jan. 2008.

ROBERT NIEUWENHUIS received the B.S. and Ph.D. degrees in computer science from the Technical University of Catalonia (UPC), Barcelona, Spain, in 1987 and 1990, respectively. He has been a Professor of computer science with UPC, since 2003. He is well known for his research at UPC and abroad, namely at the Max-Planck Institute, on automated reasoning, constraints, SAT, SMT, or ILP, with highly cited publications and recognition as an invited speaker, the program committee chair, and an editorial board membership in main conferences and journals. He is also known from the creation of the Barcelogic software tools for SAT and SMT, as well the company of the same name specialized in hard combinatorial optimization problems for scheduling and timetabling.



ALBERT OLIVERAS received the B.S. degree in mathematics and the Ph.D. degree in computer science from the Technical University of Catalonia, Barcelona, Spain, in 2002 and 2006, respectively. From 2006 to 2012, he was an Assistant Professor with the Department of Computer Science, Technical University of Catalonia, where he has been an Associate Professor, since 2012. He has authored more than 40 research papers, and the developer of several tools for SAT, SMT, and ILP. His research interests include logics in computer, with special interest in SAT, SMT, and variants of those problems.

ENRIC RODRíGUEZ-CARBONELL received the B.S. degree in mathematics and the Ph.D. degree in computer science from the Technical University of Catalonia (UPC), Barcelona, Spain, in 2002 and 2006, respectively. Since 2012, he has been an Associate Professor with the Department of Computer Science, Technical University of Catalonia. His main research interests include the applications of logics to computer science, in particular to program verification and analysis and to combinatorial problem solving.

EMMA ROLLON received the B.S. and Ph.D. degrees in computer science from the Technical University of Catalonia (UPC), in 2001 and 2008, respectively. She was a Research Assistant with the Ecole Polytechnique Federale de Lausanne (EPFL), in 2001, and the Universitat de Girona, in 2002. From 2003 to 2010, she was an Assistant Professor at UPC, where she has been an Associate Professor, since 2011. Her research interests include discrete optimization, constraint programming, and Boolean satisfiability.