Modeling and Solving Staff Scheduling with Partial Weighted maxSAT

Emir Demirović · Nysret Musliu · Felix Winter*

Abstract Employee scheduling is a well known problem that appears in a wide range of different areas including health care, air lines, transportation services, and basically any organization that has to deal with workforces. In this paper we model a collection of challenging staff scheduling instances as a weighted partial Boolean maximum satisfiability (maxSAT) problem. Using our formulation we conduct a comparison of four different cardinality constraint encodings and analyze their applicability on this problem. Additionally, we measure the performance of two leading solvers from the maxSAT evaluation 2015 in a series of benchmark experiments and compare their results to state of the art solutions. In the process we also generate a number of challenging maxSAT instances that are publicly available and can be used as benchmarks for the development and verification of modern SAT solvers.

Keywords Employee Scheduling · maxSAT · SAT encodings · cardinality constraints

1 Introduction

Staff scheduling problems arise whenever there is the need for efficient management and distribution of workforce over periods of time. Therefore, a wide range of different institutions can benefit from an optimized staff schedule, including hospitals, airlines, security personnel, transportation services, and basically any organization that has to deal with a large number of employees. Finding the ideal workforce roster however is not an easy task, and the general problem belongs to the class of NP-hard problems.

* corresponding author

Vienna University of Technology
Database and Artificial Intelligence group
E-mail: {demirovic,musliu,winter}@dbai.tuwien.ac.at

A variety of different staff scheduling problems are described in the literature and many solving methods have been proposed in the past. Corresponding surveys regarding employee scheduling can be found in [11] and [3]. While a lot of approaches are based on mathematical programming and heuristic methods, the application of solution strategies which model the problem as a maximum Boolean satisfiability problem (maxSAT) has not been considered. The intuitive way of working with propositional formulas, as well as growing developments in the SAT community motivate the investigation of such an approach.

In this paper we concentrate on the employee scheduling instances introduced in [9]. According to the authors those instances were designed to describe realistic and challenging staff scheduling problems while still being straightforward to use. The included scheduling periods range from one week up to one year, requiring up to 180 employees and 32 shift types to be assigned.

Recent publications provided an integer programming (IP) model as well as a metaheuristic approach to this problem. The best known solutions using these techniques as well as a description of the IP formulation can be found in [9]. A detailed description of the used algorithms, namely a branch and price method and a metaheuristic based on ejection chains can be found in [7] and [8]. With the use of the branch and price algorithm and ejection chains, optimal solutions could be found for most of the smaller instances and new lower/upper bounds could be determined for many instances. However, optimal solutions for a number of instances are still unknown.

In this paper we want to investigate a new solving paradigm for this problem based on maxSAT. Modeling a problem with a maximum propositional satisfiability formulation has shown to perform well on a variety of different applications in the past, including the scheduling of B2B meetings [6] and High School Timetabling [10]. However, to the best of our knowledge, such an approach has not been considered for the employee scheduling problem.

The main contributions of this paper are:

- We provide the first maxSAT formulation for the variant of the employee scheduling problem introduced in [9].
- We experiment with different encodings for cardinality constraints and compare two leading solvers from the maxSAT evaluation 2015. Additionally, we experiment with a simplification of the problem and provide a comparison with the state of the art solutions.
- We provide challenging instances which can be used by the maxSAT community to test and improve results of maxSAT solvers.

In Section 2 we first give a problem description to provide a deeper insight into the problem. We provide a brief introduction into the maximum satisfiability problem (maxSAT) and give the details of our maxSAT model in Section 3. In Section 4 we then present our experimental environment as well as the experiments which have been conducted. At the end of the paper in Section 5 we make final conclusions and provide an outlook on future work.

2 Problem description

In our work we deal with a variant of the employee scheduling problem as it is described in [9]. We chose to focus on this specific problem formulation as it provides a number instances that include challenging and realistic scheduling problems, while still being intuitive and straightforward to use.

The overall goal is to find an optimal roster for a number of given employees and shift types, where every employee may either work a single shift or have a day off on each day of a given scheduling period. For this problem the scheduling period is stated as a number of weeks and therefore the number of days is always a multitude of seven. Another property concerning the scheduling horizon ensures that the first day of the roster is always a Monday, while the last day is always a Sunday. The employees and shift types which are considered in this problem are specified by a list of unique names which are connected with a number of constraints that restrict all possible shift assignments. Some employees might for example be only allowed to work in certain shifts and patterns of consecutive working shifts might be prohibited or requested. Each problem instance specifies hard- and soft-constraints to set up a corresponding rule set. Hard constraints on the one hand are always strict and have to be fulfilled in order to generate a feasible solution. Soft constraints on the other hand may be violated, but will in case of a violation lead to an integer valued penalty. For example a hard constraint in our problem could restrict the minimum and maximum amount of time that an employee has to work in total over the whole scheduling horizon. Personal shift requests that employees can state are formulated as soft constraints in our problem instances.

Finally, the objective function of a candidate solution is defined as the sum of all violated soft constraints. We therefore deal with an optimization problem, where the optimal solution is the schedule with the lowest possible objective value. We have a deeper look on all of the constraints in next section.

3 Modeling Employee Scheduling as Partial Weighted maxSAT

3.1 The Maximum Satisfiability problem

The Satisfiability problem (SAT) is a decision problem which asks whether there exist assignments of truth values to variables, such that a propositional logic formula is evaluated true (that is, the formula is satisfied). A propositional logic formula is built from Boolean variables using logic operators and parentheses. The formula is usually given in Conjunctive Normal Form (CNF), meaning that the formula is a conjunction of clauses, where a clause is a disjunction of literals, where a literal is a variable or its negation. For example, the formula $(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_3)$ is said to be satisfiable, because the assignment $(X_1, X_2, X_3) = (true, false, false)$ satisfies the formula. However, had we inserted the clause $(\neg X_1 \vee X_2 \vee X_3)$, the same assignment would no longer satisfy the formula.

An extension to SAT that we consider in this work is Partial Weighted maxSAT, in which clauses are partitioned into two types: hard and soft clauses. Each soft clause is given a weight. The goal is to find an assignment which satisfies the hard clauses and minimizes the sum of weights of the unsatisfied soft clauses. For more in depth information about SAT and maxSAT, we direct the interested reader to [5].

In the following sections we will formulate our problems as maxSAT. The obtained maxSAT formulas which model the problem are called encodings.

3.2 Decision variables

In order to model the assignment of shifts to employees, we define variables $S_{i,d,t}, \forall i \in I, d \in D, t \in T$, where I denotes the set of all employees, D refers to the set of days in the planning horizon, and T is the set of all shift types in the problem. Each variable $S_{i,d,t}$ will be set to true if and only if employee i gets the shift type t assigned on the d-th day in the roster, otherwise it will be set to false. Additionally, we define helper variables $X_{i,d}, \forall i \in I, d \in D$ which are set to true if employee i has no shift assigned on day d. So $X_{i,d}$ is set to true if and only if employee i is considered to have a day off on this day.

To connect the X variables with the decision variables S we include the following equivalences in our formulation:

$$X_{i,d} \leftrightarrow \bigwedge_{t \in T} \neg S_{i,d,t} \qquad \forall i \in I, d \in D \tag{1}$$

In the following sections we give a description of all the constraints in our employee scheduling problem and additionally specify how each of them is encoded in our partial weighted maxSAT formulation. Clauses which are generated from soft constraints will also have weights assigned.

Since many of the constraints contain properties of cardinality constraints, we continue by shortly introducing the notion of them.

3.3 Cardinality Constraints

In order to be able to formulate all of the constraints for the problem, it is necessary to make use of cardinality constraints. Such constraints define limits on the number of truth assignments on a set of given Boolean variables. There are three different types of cardinality constraints: $atLeast_k(x_i : x_i \in X)$, $exactly_k(x_i : x_i \in X)$, and $atMost_k(x_i : x_i \in X)$ which are defined on sets of variables that should have at least, exactly, or at most k variables having their truth value assigned. For example if a cardinality constraint limits the number of true valued variables of the set x_1, x_2, x_3 to at most two $atMost_2(\{x_1, x_2, x_3\})$, the assignment $(x_1, x_2, x_3) = (1, 1, 0)$ is considered to be feasible, while the assignment $(x_1, x_2, x_3) = (1, 1, 1)$ would be considered as infeasible.

Additionally, we have to distinguish between hard- and soft cardinality constraints. While hard cardinality constraints decide whether or not the overall solution will become feasible, soft cardinality constraints will only penalize the objective function if violated. In our problem we assign a weight to a cardinality constraint and calculate the total penalty linearly depending on the difference to the violated limit. For example if we consider the constraint $atLeast_2(\{x_1, x_2, x_3\})$ with a weight of 40, the assignment $(x_1, x_2, x_3) = (0, 0, 0)$ would lead to a penalty of $40 \cdot 2 = 80$.

Different variants of dealing with cardinality constraints in Boolean satisfiability problems have been studied in the literature ([14],[2]). In this paper we investigate four different encoding types: combinatorial encoding, sequential encoding, bit adder encoding, and cardinality networks.

The combinatorial encoding enumerates all possible undesired truth assignments and forbids them explicitly by generating corresponding clauses. While this approach may provide an efficient encoding for small cardinality constraints (for example $atMost_2(\{x_1, x_2, x_3\})$ would be encoded into the single clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$), the number of generated clauses will grow exponentially with the number of variables. An alternative approach would be to explicitly enumerate all desired truth assignments.

The idea behind the sequential and bit adder encoding is to capture the sum of the considered variables and then forbid certain output values. For example, if we have the assignment $(x_1, x_2, x_3) = (1, 1, 1)$, both encodings would calculate the sum 3, but the difference lies in the way how this is encoded. The sequential encoding represents the sums as a unary number (number with base 1, e.g. $3_{10} = 111_1$), while the bit adder encoding represents the sum as a binary number (number with base 2, e.g. $3_{10} = 11_2$). The choice of the number representations has an impact on the number of generated clauses, variables, and some other maxSAT properties. Clearly, by restricting certain outputs, the desired cardinality constraint can be captured.

Cardinality networks generate helper variables that are used to sort all the considered truth assignments and then insert clauses which forbid certain outputs. The sorting is performed in a similar way as a simple merge sort algorithm would work. For example, considering an assignment $(x_1 = 0, x_2 = 1, x_3 = 0, x_3 = 1)$, the helper variables a_{1-4} would represented the sorted version of this assignment $(a_1 = 1, a_2 = 1, a_3 = 0, a_4 = 0)$. Similarly as before, additional clauses are then inserted to forbid undesired assignments of the helper variables.

In the experiments covered in later sections of this paper we compare the performance of those four encodings on two maxSAT solvers.

3.4 Modeling of Hard Constraints

An employee cannot be assigned more than one shift on a single day. Since no employee should work two shifts on the same day, we have to ensure that no two variables $S_{i,d,t}$ and $S_{i,d,x}$ may be set to true at the same time if $t \neq x$ and

$i \in I$, $d \in D$, $t, x \in T$ where I is the set of all employees, D is the set of all days in the scheduling horizon and T is the set of all possible shift types.

We model this constraint with an $atMost_1$ cardinality constraint.

$$atMost_1(\{S_{i,d,1}, S_{i,d,2}, ..., S_{i,d,|T|}\}) \qquad \forall i \in I, d \in D \tag{2}$$

Disallowed shift sequences. It is required that each employee needs to rest for a minimum amount of time after he has worked in a shift. The length of the necessary rest period varies for each shift type. Because each shift has fixed starting and ending times during the day, the set of shift types that cannot follow a certain shift type t can be determined easily by considering all pairs of shift types and comparing their difference in start and ending times with the minimum rest time. We refer to the set of all shift types that are not allowed to follow a shift t as R_t.

The constraints can therefore also be thought of as a number of disallowed shift sequences which consist of two consecutive shifts and can be included in our formulation by inserting a clause for each sequence.

$$\bigwedge_{d=1}^{|D|-1} (S_{i,d,t} \rightarrow \neg S_{i,d+1,x}) \qquad \forall t \in T, x \in R_t \tag{3}$$

The maximum numbers of shifts for each type that can be assigned to an employee. In our problem some of the employees can have contracts which only allow them to work in specific shift types for a maximum number of days. For example such a limit could restrict the number of night shifts an employee may work during the schedule to four, making any roster which assigns five night shifts to a single employee infeasible. The maximum numbers for each employee and shift type are given as parameters m_{it}^{max} with the problem instances, where $i \in I$ and $t \in T$.

Since this constraint can be seen as the basic case for a cardinality constraint, we do not discuss the detailed encoding into Boolean satisfiability clauses here, but simply state it as an $atMost$ cardinality constraint instead:

$$atMost_{m_{it}^{max}}(\{S_{i,1,t}, S_{i,2,t}, ..., S_{i,|D|,t}\}) \qquad \forall i \in I, t \in T \tag{4}$$

Minimum and maximum working time. Each shift type assigns a certain amount of working time in minutes to its associated employees. Moreover the total number of the working time in minutes is restricted for each employee and must lie between a minimum and maximum bound. Those limits are given to the problem in form of the parameters b_i^{min} and b_i^{max} for each $i \in I$.

In order to formulate this constraint efficiently, we introduce additional helper variables which help us to count the total number of minutes worked by an employee. For their definition we consider the shift lengths $l_t, \forall t \in T$ which are given as parameters to the problem and specify the number of working time in minutes required for shift t. Furthermore we define their greatest common

divisor $g = gcd(l_t : t \in T)$. If we have three different shift types, with the first one lasting for 480, the second one lasting 620, and the third one lasting 120 minutes, g would then be 20 for example.

With g, we can then calculate simplified lengths for all shifts $sl_t = \frac{l_t}{g}$ $\forall t \in T$. Additionally, we define the maximum simplified shift length $sl_{max} = max\{sl_t : t \in T\}$. Now we are able to introduce our variable set U, which counts the units of time an employee i works on day d.

For each employee and day, we introduce helper variables $U_{i,d,x}, \forall i \in I, d \in D, x \in 1, ..., sl_{max}$ and set up a number of equivalences in order to correctly connect them with our decision variables S:

$$S_{i,d,t} \leftrightarrow \bigwedge_{x=1}^{sl_t} U_{i,d,x} \bigwedge_{y=sl_t}^{sl_{max}} \neg U_{i,d,y} \tag{5}$$

All of the U variables can now be used to count the overall units of time an employee works and we can use them in order to set up two cardinality constraints that ensure the minimum and maximum working time constraint. Note that since we are using simplified lengths, we also have to divide the given limits by the common divisor g and round appropriately:

$$atMost_{\lfloor b_i^{max}/g \rfloor}(\{U_{i,d,x} | d \in D, x \in \{1, ..., sl_{max}\}\}) \quad \forall i \in I \tag{6}$$

$$atLeast_{\lceil b_i^{min}/g \rceil}(\{U_{i,d,x} | d \in D, x \in \{1, ..., sl_{max}\}\}) \quad \forall i \in I \tag{7}$$

Maximum consecutive shifts. Each employee is only allowed to work for a maximum number of consecutive days before he must have a day off. This maximum limit is given to the problem as c_i^{max} for each $i \in I$. In our formulation we state this constraint by introducing clauses that require a day off during all possible sequences of length c_i^{max}. Since this constraint assumes that the last day before the scheduling horizon sets a day off and the first day after the scheduling horizon also sets a day off, we do not need to consider any corner cases.

$$\bigvee_{x=0}^{c_i^{max}} X_{i,d+x} \qquad \forall i \in I, d \in \{1, ..., |D| - c_i^{max}\} \tag{8}$$

Minimum consecutive shifts. Our problem requires that each employee works at least for a minimum of consecutive days. In other words there is a minimum for the number of consecutive shifts, which is given as parameter c_i^{min} for all $i \in I$, before an employee is allowed to have a day off.

Again we do not have to consider corner cases, since this constraint always assumes an infinite number of consecutive working days before and after the scheduling horizon. For all the other cases, we formulate this constraint by implicating the minimum length shift sequence whenever a new shift sequence starts after a day off:

$$(X_{i,d} \wedge \neg X_{i,d+1}) \rightarrow (\bigwedge_{x=2}^{c_i^{min}} \neg X_{i,j+x}) \qquad \forall i \in I, d \in \{1, ..., |D| - 3\} \tag{9}$$

Minimum consecutive days off. This can be formulated similarly to the minimum consecutive shifts constraint. No corner cases have to be considered, as this constraint assumes an infinite sequence of days off before and after the scheduling horizon. The minimum limit of consecutive days off is given to the problem as parameter o_i^{min} for each employee $i \in I$.

We again use a formulation variant which applies an implication of a minimum length day off sequence, similar as described for the minimum consecutive shifts constraint which we described previously:

$$(\neg X_{i,d} \wedge X_{i,d+1}) \rightarrow (\bigwedge_{x=2}^{o_i^{min}} X_{i,d+x}) \qquad \forall i \in I, d \in \{1, ..., |D| - 3\} \tag{10}$$

Maximum number of weekends. Whenever an employee has to work a shift on a Saturday or a Sunday in the schedule, the corresponding weekend is considered as a working weekend for this employee. The problem restricts the number of such working weekends for each employee i as parameter a_i^{max}. Because the scheduling always starts on Monday and ends on Sunday, the number of weekends can be easily calculated as $w = \frac{|D|}{7}$. We can now introduce additional helper variables $W_{i,x}$ to state if an employee i works on the x-th weekend. We introduce the following equivalences to connect the W variables with the existing X variables in our formulation. Note that the x variables are multiplied with 7 in order to determine the day index of the x-th Sunday.

$$W_{i,x} \leftrightarrow (\neg X_{i,(x \cdot 7)-1} \vee \neg X_{i,x \cdot 7}) \qquad \forall i \in I, x \in \{1, ..., w\} \tag{11}$$

With the help of those variables we can then construct the following cardinality constraints to formulate the maximum number of weekends constraint:

$$atMost_{a_i^{max}}(\{W_{i,1}, W_{i,2}, ..., W_{i,w}\}) \qquad \forall i \in I \tag{12}$$

Days off. An employee may have certain days on which it is strictly required that he has a day off. Those are given to the problem as sets of day indices N_i for each employee i. We can introduce this in our formulation by simply generating the corresponding unit clauses:

$$X_{i,d} \qquad \forall i \in I, d \in N_i \tag{13}$$

3.5 Modeling of Soft Constraints

Requested shift types. Each employee may have some days where a certain shift type is requested for them to work in. Since this is not a hard constraint, a violation will be penalized with a given weight. The corresponding penalties are given to the problem as parameters $q_{i,d,t}$, where $i \in I$, $d \in D$ and $t \in T$. We handle this constraint by inserting simple weighted unit clauses for all shift requests into our formulation:

$$S_{i,d,t} \cdot q_{i,d,t} \qquad \forall (i,d,t) \quad where \quad \exists q_{i,d,t} \tag{14}$$

Unpreferred shift types. Similar to the requested shifts constraint, our problem may contain requests that require an employee to not work a particular shift on a certain day. Our formulation is again based on weighted unit clauses depending on problem parameters $p_{i,d,t}$ that set the weight of an unpreferred shift, where $i \in I$, $d \in D$ and $t \in T$:

$$\neg S_{i,d,t} \cdot p_{i,d,t} \qquad \forall (i,d,t) \quad where \quad \exists p_{i,d,t} \tag{15}$$

Cover requirements. A preferred number of employees that should be working in a shift type is defined for each day. This preferred value of working employees for shift t on day d is given to the problem in form of parameters u_{dt} for all $d \in D$ and $t \in T$. Furthermore for each of these values two penalty parameters v_{dt}^{min} and v_{dt}^{max} are used to penalize a possible under- or over-coverage of the preferred value.

We introduce two cardinality constraints per cover requirement to formulate this constraint. One for the over-coverage, which is penalized linearly depending on the weight v_{dt}^{max}, and the second one for the under-coverage also penalized linearly depending on the weight v_{dt}^{min}:

$$atMost_{u_{dt}}(\{S_{1,d,t}, S_{2,d,t}, ..., S_{|I|,d,t}\}) \cdot v_{dt}^{max} \qquad \forall d \in D, t \in T \tag{16}$$

$$atLeast_{u_{dt}}(\{S_{1,d,t}; S_{2,d,t}; ...; S_{|I|,d,t}\}) \cdot v_{dt}^{min} \qquad \forall d \in D, t \in T \tag{17}$$

4 Computational Results

We now give an overview of our experimental environment and describe how our benchmark tests were executed and evaluated.

4.1 Experimental environment

We conducted a large number of experiments with generated maxSAT encodings for the 24 instances described in [9]. The planning horizon of the instances ranges from two weeks to 52 weeks, while the number of considered employees ranges from 8 to 150. As this dataset contains very large instances it provides

challenging benchmarks for solution techniques. If not noted otherwise we ran all of our experiments on an Intel Xeon E5345 2.33GHz machine with a total of 48GB RAM. The encoded maxSAT instances are available online in DIMACS format and can be downloaded at [1].

In our benchmarks we used two solvers which performed well on timetabling instances in the maxSAT evaluation 2015: WPM3 [1] and Optiriss using the default configuration. The latter uses the riss framework [12] in combination with the publicly available OpenWBO solver [13]. Both solvers were ranked first and second in the industrial category for partial weighted maxSAT problems. Besides being the leaders in their category, both solvers have also shown to provide good results for high school timetabling and timetabling instances, which share similarities with the considered employee scheduling problem.

4.2 Comparison of different cardinality constraint encodings

Because our model utilizes a number of cardinality constraints, a crucial point in the configuration of our experiments turned out to be the determination of which cardinality constraint encodings we should use in order to get good results with the maxSAT solvers. There are five constraints which are affected in our formulation: The *cover requirement* constraint, the *workload requirement* constraint, the *maximum number of shifts* constraint, the *maximum number of weekends*, and the *One shift per day* constraint. For those, we applied four different encoding variants: *combinatorial encoding*, *sequential encoding*, *cardinality networks encoding*, and *bit adder encoding*. We used the implementation from [10] to encode those constraints.

If we would consider all possible combinations for encoding the cardinality constraints in our model, we would have to generate and compare a total of $4^5 = 1024$ different formulations for each problem instance. In order to reduce this large amount of possibilities, we decided to investigate the number of generated variables and clauses for all constraint/encoding pairs in order to gather a first insight on their importance. We can see the results for one instance in Table 1.

The combinatorial encoding turned out to be impractical in most cases and we were often not able to generate maxSAT encodings for many of the instances when using it. The huge amount of produced clauses required by this encoding forced our model generator to run out of memory when dealing with larger instances. When looking at the numbers displayed in Table 1, we can also see that the *maximum number of weekends* and the *one shift per day* constraints have a relatively low impact when compared with the other constraints. As this behavior appeared also in other instances, we decided to use only the sequential encoding for the *maximum number of weekends* constraint and only the combinatorial encoding for the *one shift per day* constraint in the remainder of our experiments. With the elimination of the combinatorial

[1] http://www.dbai.tuwien.ac.at/research/project/arte/maxsat_employeescheduling/

Table 1 Overview on the number of generated variables (vars.) as well as the hard- and soft-clauses (h.c. and s.c.) for all the cardinality constraint/encoding pairs for instance 5.

		Combinatorial	Sequential	Cardinality N.	Bit adders
Cover Req.	vars.	10192	7616	7056	5096
	h.c.	35616	28672	21168	17808
	s.c.	896	896	896	896
Workload Req.	vars.	Out of memory	6176	8032	5088
	h.c.	Out of memory	20240	21760	14864
	s.c.	Out of memory	0	0	0
Max shifts	vars.	Out of memory	3010	3520	6374
	h.c.	Out of memory	11928	10658	22646
	s.c.	Out of memory	0	0	0
Max weekends	vars.	0	124	160	176
	h.c.	46	420	496	602
	s.c.	0	0	0	0
One shift per day	vars.	0	896	896	1344
	h.c.	448	2688	3136	4480
	s.c.	0	0	0	0

encoding in our configuration options because of the caused inconveniences with larger instances, and only three constraints remaining, we now have to examine only $3^3 = 27$ possible combinations.

In order to determine the best configuration for both WPM3 and Optiriss, we selected nine instances of different sizes and ran experiments with all the 27 possible encoding variants under a time limit of 30 minutes. The results of those experiments can be seen in Table 2 and Table 3 for Optiriss and WPM3 respectively.

A comparison of those results reveals that there is no general best combination of cardinality constraint encodings and good encodings are highly dependent on the solver which is used. While Optiriss prefers the adder encoding for the *cover requirements* constraint, the sequential encoding shows the best results for WPM3. We selected the best candidates for each solver by considering the sums of the results over all instances for each combination of cardinality encodings. The encodings which led to the minimum of all those sums were then taken to generate the instances for our final experiments. Therefore, the combinations of cardinality constraint encodings used for Optiriss were as follows: bit adder encoding for the *cover requirements* constraint, cardinality networks for the *workload requirements* constraint, and the sequential encoding for the *maximum number of shifts* constraint. The combinations of cardinality constraint encodings for WPM3 on the other hand were: The sequential encoding for the *cover requirements* constraint, the *workload requirements* constraint, and the encoding which uses cardinality networks for the *maximum number of shifts* constraint.

4.3 Final experiments and comparison of solvers

By using the encodings mentioned above, we were able to create maxSAT instances for the original problems 1-21. Although our formulation can be used to encode Instances 22-24, unfortunately we could not generate maxSAT

Table 2 Best results found by Optiriss using different combinations of cardinality encodings. The first column describes the cardinality encodings used for the *cover requirement/workload requirement/maximum number of shifts* constraints. Encoding names have been abbreviated: seq. = sequential encoding, card. = cardinality networks, adder = bit adders. In each column the best result is formatted in boldface.

Optiriss	**Best solutions found in 30 minutes time limit**				
Cardinality encoding	**Inst. 2**	**Inst. 4**	**Inst. 7**	**Inst. 9**	**Inst. 11**
seq./seq./seq.	837	5626	13333	12655	40435
seq./seq./card.	837	5626	12000	11659	40435
seq./seq./adder	839	5122	10078	11533	23720
seq./card./seq.	840	6002	15318	12460	32768
seq./card./card.	840	6002	12111	12242	32768
seq./card./adder	838	5215	11474	12758	24905
seq./adder/seq.	841	5407	14319	11044	34612
seq./adder/card.	841	5407	15148	11662	34612
seq./adder/adder	840	5331	11785	12752	25633
card./seq./seq.	841	5609	13813	14353	32281
card./seq./card.	841	5609	15211	11423	32281
card./seq./adder	**834**	5723	11987	13250	25631
card./card./seq.	**834**	6210	14080	12156	37028
card./card./card.	**834**	6210	13779	13154	37028
card./card./adder	841	5316	10682	10641	22130
card./adder/seq.	837	5711	13002	12570	32618
card./adder/card.	837	5711	13492	12785	32618
card./adder/adder	838	5504	9689	12976	24844
adder/seq./seq.	844	3900	5762	7729	15916
adder/seq./card.	844	3900	5741	7526	15916
adder/seq./adder	852	3720	5228	7437	16624
adder/card./seq.	853	**3608**	5421	**6394**	**15420**
adder/card./card.	853	**3608**	5852	6804	**15420**
adder/card./adder	847	3918	5452	7239	16464
adder/adder/seq.	845	3907	5411	7716	16627
adder/adder/card.	845	3907	5746	7422	16627
adder/adder/adder	850	3798	**5040**	7215	16436
Optiriss	**Best solutions found in 30 minutes time limit**				
Cardinality encoding	**Inst. 12**	**Inst. 14**	**Inst. 16**	**Inst. 18**	
seq./seq./seq.	57680	17959	15584	35073	
seq./seq./card.	58369	16665	15584	39555	
seq./seq./adder	34964	17549	13263	25829	
seq./card./seq.	57575	16761	15635	37084	
seq./card./card.	54138	16630	15635	37641	
seq./card./adder	33939	18362	14544	23932	
seq./adder/seq.	61229	17454	16013	34284	
seq./adder/card.	52854	16043	16013	28074	
seq./adder/adder	36632	15555	13937	27604	
card./seq./seq.	72062	19358	16093	37501	
card./seq./card.	49699	18247	16093	38147	
card./seq./adder	32074	17934	14776	28188	
card./card./seq.	56279	19044	16903	37778	
card./card./card.	50404	15546	16903	35638	
card./card./adder	32239	16918	14880	26855	
card./adder/seq.	62154	18980	17419	38269	
card./adder/card.	49096	18565	17419	30601	
card./adder/adder	33340	18593	15990	29781	
adder/seq./seq.	28602	10076	12546	21039	
adder/seq./card.	31000	9875	12546	22548	
adder/seq./adder	28694	**8777**	12223	21095	
adder/card./seq.	28598	9776	13026	20710	
adder/card./card.	30324	9144	13026	**20225**	
adder/card./adder	29596	9555	13049	20280	
adder/adder/seq.	**27193**	9758	11939	20462	
adder/adder/card.	29606	9931	11939	20504	
adder/adder/adder	29417	9756	**11707**	20996	

Table 3 Best results found by WPM3 using different combinations of cardinality encodings. The first column describes the cardinality encodings used for the *cover requirement*/*workload requirement*/*maximum number of shifts* constraints. Encoding names have been abbreviated: seq. = sequential encoding, card. = cardinality networks, adder = bit adders. In each column the best result is formatted in boldface.

WPM3	**Best solution found in 30 minutes time limit**				
Cardinality encoding	**Inst. 2**	**Inst. 4**	**Inst. 7**	**Inst. 9**	**Inst. 11**
seq./seq./seq.	**828**	3189	5510	10631	12183
seq./seq./card.	**828**	3189	**4596**	10949	12183
seq./seq./adder	**828**	3494	8959	10248	23420
seq./card./seq.	**828**	3090	7446	11132	**11516**
seq./card./card.	**828**	3090	6545	11405	**11516**
seq./card./adder	**828**	**2688**	8351	12154	24114
seq./adder/seq.	**828**	2784	7712	12178	12478
seq./adder/card.	**828**	2784	8553	**10033**	12478
seq./adder/adder	**828**	2893	9364	10964	24195
card./seq./seq.	835	3394	5230	10605	17224
card./seq./card.	835	3394	6815	11037	17224
card./seq./adder	**828**	4082	7562	11062	25444
card./card./seq.	**828**	3087	7143	10240	13888
card./card./card.	**828**	3087	8147	10942	13888
card./card./adder	839	3704	9670	10531	25626
card./adder/seq.	840	3695	7543	11871	15393
card./adder/card.	840	3695	7760	11235	15393
card./adder/adder	**828**	3103	9287	12374	22719
adder/seq./seq.	1550	3718	10502	13982	29673
adder/seq./card.	1550	3718	11315	12780	29673
adder/seq./adder	1159	3198	9478	14674	26133
adder/card./seq.	1563	3994	9365	11256	31595
adder/card./card.	1563	3994	9253	12773	31595
adder/card./adder	856	4212	9791	12693	25827
adder/adder/seq.	1469	4108	10292	10771	29083
adder/adder/card.	1469	4108	9476	11963	29083
adder/adder/adder	1359	3702	10100	10935	26467
WPM3	**Best solution found in 30 minutes time limit**				
Cardinality encoding	**Inst. 12**	**Inst. 14**	**Inst. 16**	**Inst. 18**	
seq./seq./seq.	23937	18045	**10292**	19771	
seq./seq./card.	**18770**	16303	**10292**	18498	
seq./seq./adder	1697590	15297	12738	21408	
seq./card./seq.	22010	15419	12528	19191	
seq./card./card.	19845	16285	12528	19241	
seq./card./adder	1697590	16654	16099	22100	
seq./adder/seq.	22536	17130	12550	**17277**	
seq./adder/card.	22734	16330	12550	20139	
seq./adder/adder	1697590	15155	15031	20793	
card./seq./seq.	24142	18272	12015	21095	
card./seq./card.	23726	18948	12015	22605	
card./seq./adder	32150	18455	14126	29910	
card./card./seq.	24206	16321	12848	25567	
card./card./card.	23716	16864	12848	21097	
card./card./adder	1697590	**14915**	16176	24417	
card./adder/seq	25331	17055	13360	24620	
card./adder/card.	26272	18104	13360	25051	
card./adder/adder	1697590	17490	16998	30144	
adder/seq./seq.	48422	18356	18064	31426	
adder/seq./card.	44948	19731	18064	29955	
adder/seq./adder	38822	18376	16497	27336	
adder/card./seq.	42744	19131	16259	28860	
adder/card./card.	44272	18959	16259	31694	
adder/card./adder	37582	18494	16343	30929	
adder/adder/seq.	42648	15723	17590	31791	
adder/adder/card.	45583	20184	17590	27403	
adder/adder/adder	35857	18143	18593	29081	

instances for those two problems, since our generator ran out of memory due to their large size (about 20 GB). Our final experiments were conducted using both solvers, giving them a time limit of four hours for each of the 21 instances. The results of those benchmark tests can be seen in Table 4.

Table 4 The final results obtained for Instance 1-21 using WPM3 and Optiriss, using the selected cardinality constraint encodings described in this paper. For comparison, the best known solutions using the exact methods described in [9] are also included. Results formatted in bold face denote proven optimal solutions.

Instance	WPM3	Optiriss	Branch and Price [9]	Gurobi [9]
Instance 1	**607**	**607**	**607**	**607**
Instance 2	**828**	835	**828**	**828**
Instance 3	1009	3475	**1001**	**1001**
Instance 4	3102	3608	**1716**	**1716**
Instance 5	4037	3645	1160	**1143**
Instance 6	6150	6941	1952	**1950**
Instance 7	4596	5421	1058	**1056**
Instance 8	11018	7617	1308	1323
Instance 9	10949	6394	439	439
Instance 10	16435	15350	**4631**	**4631**
Instance 11	12183	15420	**3443**	**3443**
Instance 12	18770	28598	4046	**4040**
Instance 13	6110163	69203	-	3109
Instance 14	16303	9776	-	1280
Instance 15	30833	16506	-	4964
Instance 16	10292	13026	3323	3233
Instance 17	22002	22073	-	5851
Instance 18	18498	14433	-	4760
Instance 19	1698538	50274	-	5420
Instance 20	5519316	147325	-	-
Instance 21	14715064	-	-	-

If we compare the outcomes for WPM3 and Optiriss we are not able to point out a clear winner which performs better over all the instances. While WPM3 performs significantly better on the smaller instances (Instances 1-7 and 11-12), it does not produce good solutions for the larger instances (Instances 8-10 and 13-21). Using Optiriss provides better results when it comes to solving the larger instances, except for the last two instances where the solver could not find any solution under four hours.

Comparing our approach with another existing exact method based on integer programming, which was provided in [9] (last two columns in the table) we can conclude that both maxSAT solvers could not find new unknown optimal results. However they could provide optimal solutions for instances 1 and 2. Running the maxSAT solvers for four hours resulted in finding solutions for two of the instances which could not be solved by the integer programming approach within one hour on a different environment. Although the integer programming method could also possibly find those solutions within four hours, the results show that maxSAT as an exact method gives promising results for employee scheduling problems. As many maxSAT solvers are publicly avail-

able and their performance is consistently improving, this approach could be useful to find solutions for employee scheduling problems.

4.4 Analyzing the influence of the under-coverage soft-constraint

To further investigate the problem we performed additional experiments by simplifying the instances. We omitted all soft constraints except under-coverage (Equation 17). We wanted to investigate this constraint because it shows to have the highest weight in all instances, and as such contributed to the objective value significantly more than others.

Optiriss provides the option to experiment with the Linear maxSAT algorithm [4]. The Linear algorithm is an iterative upper bounding algorithm in which the SAT solver is repeatedly called and in each call clauses are added which constrain the objective value to be less than in the previous iteration. Therefore, this process is only repeated until the SAT solver reports unsatisfiable, in which case the previously calculated solution is the optimal one. The Linear algorithm is invoked in Optiriss by supplying the parameter *-algorithm=1*. As this algorithm is appropriate to be used in this scenario, below we report experiments using it.

Every feasible solution for the simplified instances is a feasible solution for the original problem as well, as removing soft constraints does not impact feasibility. In Table 5 we provide the results obtained after running experiments for one hour. We compare the performance of Optiriss with the Linear algorithm on the original and simplified instances (see column 1 and column 2 of Table 5). In the case of the simplified instances, we present the costs obtained after converting the solution to the original instance. We used the same cardinality constraint encoding as we did previously for Optiriss.

The results obtained in Table 5 are interesting for two reasons. Firstly, in most cases when a solution could be generated, the obtained results with the described technique with the simplified instances outperformed the previous maxSAT experiments even though less time has been allocated. Secondly, the simplification proved to be a very useful improvement for the Linear maxSAT algorithm. Instances 19-24 where not included in the table as no solution could be generated with either encoding technique using the Linear maxSAT algorithm.

These results indicate that the under-coverage constraint has a high influence on the objective value, at least for the Linear maxSAT algorithm. Because of this, leaving the solver all the time to focus on the under-coverage constraint showed to be valuable. Using this technique new unknown optimal results could not be found, but the results suggest the importance of the under-coverage constraint. Focusing only on the under-coverage constraint could also be applied to other solving techniques, like integer programming methods or local search.

Table 5 The results obtained by running Optiriss with the Linear maxSAT algorithm for one hour on simplified and original instances. For comparison purposes, we provide the corresponding solution for Optiriss from Table 4 and the best known solutions obtained by exact methods described in [9]. Optimal solutions are formatted in bold face.

Instance	Linear	Simplified-Linear	Optiriss (Table 4)
Instance 1	**607**	620	**607**
Instance 2	847	858	835
Instance 3	1236	1050	3475
Instance 4	1859	1787	3608
Instance 5	2202	1534	3645
Instance 6	5763	2637	6941
Instance 7	6541	1625	5421
Instance 8	15105	2894	7617
Instance 9	13496	1991	6394
Instance 10	-	6649	15350
Instance 11	-	6434	15420
Instance 12	-	22838	28598
Instance 13	-	70242	69203
Instance 14	-	6634	9776
Instance 15	-	24988	16506
Instance 16	18074	4867	13026
Instance 17	-	14315	22073
Instance 18	18498	13143	14433

5 Conclusion

In this paper we have introduced, to our best knowledge, for the first time a partial weighted Boolean maximum satisfiability model for a variant of the employee scheduling problem. We further generated maxSAT instances using four different cardinality encoding methods. Additionally, we compared the effects of the different cardinality encoding methods on two maxSAT solvers. We have shown that there is a need to experimentally select an efficient combination of cardinality encodings for each solver separately. A comparison between the two solvers could not point out a clear winner for all of the considered benchmark tests. While WPM3 performed better on smaller instances, Optiriss was able to produce better results for many of the larger instances.

Currently an exact approach based on integer programming provides better results than maxSAT for most of the considered instances. However, maxSAT could provide optimal solutions for two of the instances and obtained solutions for two very large instances within four hours, which could not be solved by integer programming within one hour. Therefore, as nowadays different maxSAT solvers are available and their performance is consistently improving, exact maxSAT techniques can be useful for solving employee scheduling problems in the future.

Possible improvements and extensions could be subject of future work when working with the proposed model. It would be interesting to investigate if we can break symmetries in our model. Further, given the findings regarding the under-coverage constraint, developing a lexicographic optimization approach

for Employee Scheduling might be valuable. In this approach, one would first optimize for the under-coverage constraint and then optimize the rest of the soft constraints. Apart from that, using the results of the simplified instances as a starting point for local search could be useful. Furthermore, a hybridization of maxSAT with heuristic techniques within the framework of very large neighbourhood search could be considered.

Acknowledgements The work was supported by the Austrian Science Fund (FWF): P24814-N23 and the Vienna PhD School of Informatics.

References

1. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in maxsat. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pp. 283–289 (2015)
2. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings, pp. 167–180 (2009)
3. den Bergh, J.V., Beliën, J., Bruecker, P.D., Demeulemeester, E., Boeck, L.D.: Personnel scheduling: A literature review. European Journal of Operational Research (3), 367–385 (2013)
4. Berre, D.L., Parrain, A.: The sat4j library, release 2.2. JSAT (2-3), 59–6 (2010)
5. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
6. Bofill, M., Garcia, M., Suy, J., Villaret, M.: Maxsat-based scheduling of B2B meetings. In: Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings, pp. 65–73 (2015)
7. Burke, E.K., Curtois, T.: New approaches to nurse rostering benchmark instances. European Journal of Operational Research **237**(1), 71–81 (2014)
8. Burke, E.K., Curtois, T., Post, G.F., Qu, R., Veltman, B.: A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operational Research (2), 330–341 (2008)
9. Curtois, T., Qu, R.: Computational results on new staff scheduling benchmark instances. Tech. rep., ASAP Research Group, School of Computer Science, University of Nottingham, NG8 1BB, Nottingham, UK (2014)
10. Demirovic, E., Musliu, N.: Modeling high school timetabling as partial weighted maxsat. Tech. rep., Technical University Vienna (2014)
11. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research (1), 3–27 (2004)
12. Kahlert, L., Krüger, F., Manthey, N., Stephan, A.: Riss solver framework v5. 05. SAT-Race (2015)
13. Martins, R., Manquinho, V.M., Lynce, I.: Open-wbo: A modular maxsat solver,. In: Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, pp. 438–445 (2014)
14. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings, pp. 827–831 (2005)