Solver Independent Rotating Workforce Scheduling

Nysret Musliu[1], Andreas Schutt[2,3], and Peter J. Stuckey[2,3]

[1] TU Wien, Vienna, Austria
[2] Data61, CSIRO, Australia
[3] University of Melbourne, Victoria, Australia

Abstract. The rotating workforce scheduling problem aims to schedule workers satisfying shift sequence constraints and ensuring enough shifts are covered on each day, where every worker completes the same schedule, just starting at different days in the schedule. We give two solver independent models for the rotating workforce scheduling problem and compare them using different solving technology, both constraint programming and mixed integer programming. We show that the best of these models outperforms the state-of-the-art for the rotating workforce scheduling problem, and that solver independent modeling allows us to use different solvers to achieve different aims: *e.g.*, speed to solution or robustness of solving (particular for unsatisfiable problems). We give the first complete method able to solve all of the standard benchmarks for this problem.

1 Introduction

Rotating workforce scheduling is a specific personnel scheduling problem arising in many spheres of life such as, *e.g.*, industrial plants, hospitals, public institutions, and airline companies. Table 1 shows a workforce schedule for 7 employees during one week, in which a row represents the weekly schedule of one employee. There are three shifts: day shift (D), afternoon shift (A), and night shift (A). The first employee works from Monday till Thursday in the afternoon shift and has days-off in the remaining week. The second employee has a day-off on Thursday and Friday and works in the day shift in the other days. The last employee

Table 1. A typical week schedule for 7 employees.

Emp./day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	A	A	A	A	-	-	-
2	D	D	D	-	-	D	D
3	D	-	-	N	N	N	N
4	-	-	-	-	A	A	A
5	D	D	D	D	D	-	-
6	N	N	N	N	N	-	-
7	N	N	N	-	-	D	D

starts the week with 3 night shifts, then rests for two days, and ends the week with 2 day shifts. A schedule must meet many constraints such as workforce requirements for shifts and days, minimal and maximal length of shifts, and shift transition constraints, which are described in detail in the next section. For rotating workforce scheduling, the schedule is rotating (or cyclic), *i.e.*, the i^{th} employee has the schedule of the $((k \bmod n) + 1)^{th}$ employee after the k^{th} week, where n is the number of employees. Due to that, no personal preferences of employees can be considered. The aim is to find a schedule satisfying all the constraints. Rotating workforce scheduling problems are NP-complete [7]. According to [1,25], the problem studied can be characterised as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints.

Many practical real-life rotating workforce scheduling problems have been solved by complete techniques [2,21,18,27,11] and heuristic algorithms [19,20], but solving large problems is still a challenging task. Balakrishnan and Wong [2] formulate the problem as a network flow problem. Laporte [17] proposes a Integer Linear Programming approach. Methods based on Constraint Programming (CP) techniques are studied in [18,21,27]. Recently, Erkinger and Musliu [11] propose a Satisfiability Modulo Theory (SMT) approach, which—to our best knowledge—defines the state of the art complete method. All methods have been evaluated on the benchmark set with 20 instances [19,20], which are based on real life problems from different business areas, or on a sub-set of them. The state of the art complete method [11] was able to solve 18 of them, whereas the state-of-the-art heuristic approach [19,20] based on min-conflicts heuristic and tabu search (MC-T) found a solution for all of them. Other research studies focus at the creation of efficient rotation schedules by hand [16], and the design and the analysis of rotating schedules with an algebraic computational approach [12].

There are various variants of personnel scheduling (see, *e.g.*, the surveys [5,4]), one group of them is the (multi-activity) shift scheduling problem, which is generally concerned about a finer schedule of the shifts over a planning horizon of one day considering, *e.g.*, meal breaks and more workforce regulations. Because of the finer nature, there has been a great deal of work in formalising languages and their automata or network flows for capturing most of the regulations (see, *e.g.*, [24,10,9,26,15]). Beside the technologies mentioned in the previous paragraph, researchers has been investigating in Column Generation based methods (see, *e.g.*, [25,14]) to tackle large personnel scheduling problems.

We define two solver-independent models for rotating workforce scheduling, and compare them using a CP and a Mixed-Integer Programming (MIP) solver. The first model is rather direct, where each constraint is separately stated. The second model attempts to model as much as possible of the regulations using a single `regular` constraint [23], which models the regulations as a deterministic finite automaton. We consider redundant constraints, and symmetry breaking constraints that can be added to the model to possibly improve them. We compare the variations of the models experimentally using the two solvers, and explore good search strategies to be used with the CP solver. Moreover, we generated 1980 additional instances and show that our two best methods outperform

the state of the art approaches [11,19,20], on both the standard 20 benchmark instances and the extended 2000 instances.

2 The Rotating Workforce Scheduling Problem

We focus on a specific variant of a general workforce scheduling problem, which we formally define in this section. The following definition is from Musliu *et al.* [21] and proved to be able to satisfactorily handle a broad range of real-life scheduling instances in commercial settings. A rotating workforce scheduling problem as discussed in this paper consists of:

- n: Number of employees.
- $\mathbf{A}$: Set of m shifts (activities). There is also a "day-off" activity denoted O. We let $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$.
- w: Length of the schedule. A typical value is $w = 7$, to assign one shift type for each day of the week to each employee. The total length of a planning period is $n \times w$ due to the schedule's cyclicity as discussed below.
- R: Temporal requirements matrix, an $m \times w$-matrix where each element $R_{i,j}$ shows the required number of employees that need to be assigned shift type $i \in \mathbf{A}$ during day j. The number o_j of day-off "shifts" for a specific day j is implicit in the requirements and can be computed as $o_j = n - \sum_{i=1}^{m} R_{i,j}$. In an abuse of notation we let $R_{O,j} = o_j$.
- Sequences of shifts not permitted to be assigned to employees. We consider two kinds of forbidden sequences: length 2 sequences, *e.g.*, ND (Night Day): after working in the night shift, it is not allowed to work the next day in the day shift; and length 3 sequences, *e.g.*, DON (Day Off Night): after working a day shift and then having a day off, it is not allowed to work the next day in the night shift. In length 3 sequences the middle shift is always O (Off). A typical rotating workforce instance forbids several shift sequences, often due to legal reasons and safety concerns. These two kinds are sufficient for all the cases we have met in practice. We represent the forbidden sequence as two sets of pairs $(sh_1, sh_2) \in F_2$ if it is forbidden to take shift sh_2 directly after sh_1; and $(sh_1, sh_2) \in F_3$ if it is forbidden to take shift sh_2 directly after a single O shift after sh_1.
- l_s and u_s: Each element of these vectors shows, respectively, the required minimal and permitted maximal length of periods of consecutive shifts $s \in \mathbf{A}^+$ of the same type.
- l_w and u_w: Minimal and maximal length of blocks of consecutive work shifts. This constraint limits the number of consecutive days on which the employees can work without having a day off.

The task in rotating workforce scheduling is to construct a cyclic schedule, which we represent as an $n \times w$ matrix $S_{i,j} \in \mathbf{A}^+, 1 \leq i \leq n, 1 \leq j \leq w$. Each element $S_{i,j}$ denotes the shift that employee i is assigned during day j in the first period of the cycle, or whether the employee has time off on that day. In a

cyclic schedule, the schedule for one employee consists of a sequence of all rows of the matrix S.

The task is called rotating or cyclic scheduling because the last element of each row is adjacent to the first element of the next row, and the last element of the matrix is adjacent to its first element. Intuitively, this means that employee i ($i < n$) assumes the place (and thus the schedule) of employee $i + 1$ after each week, and employee n assumes the place of employee 1. This cyclicity must be taken into account for the last three constraints above.

In the present paper, we consider the satisfaction problem satisfying all constraints given in the problem definition, which is usually sufficient in practice. This means the generation of one schedule is sufficient. The commercial software FCS [21,13] uses the same constraints for generating rotating workforce schedules. This system has been used since 2000 in practice by many companies in Europe and the scheduling variant we discuss in this paper proved to be sufficient for a broad range of uses.

3 Direct Model

The direct model of the problem asserts each of the constraints individually. To make it easy to handle the cyclic nature of the schedule we define a new view on the schedule $T_k = S_{k \div w+1, k \mod w+1}, 0 \leq k \leq n \times w - 1$ which simply maps the days of the schedule to a list of length $n \times w$ indexed from $TT = \{0, \ldots, n \times w - 1\}$. Let $t(x) = x \mod (n \times w)$ be a map from days to indexes of the list. We can then assert the constraints individually

$$\sum_{k \in 0}^{u_w} (T_{t(j+k)} = O) > 0, \quad j \in TT \tag{1}$$

$$\sum_{k \in 1}^{l_w} (T_{t(j+k)} = O) = 0, \quad j \in TT, T_j = O \wedge T_{t(j+1)} \neq O \tag{2}$$

$$\sum_{k \in 0}^{u_O} (T_{t(j+k)} \neq O) > 0, \quad j \in TT \tag{3}$$

$$\sum_{k \in 1}^{l_O} (T_{t(j+k)} \neq O) = 0, \quad j \in TT, T_j \neq O \wedge T_{t(j+1)} = O \tag{4}$$

$$\sum_{k \in 0}^{u_{sh}} (T_{t(j+k)} \neq sh) > 0, \quad j \in TT, sh \in \mathbf{A} \tag{5}$$

$$\sum_{k \in 1}^{l_{sh}} (T_{t(j+k)} \neq sh) = 0, \quad j \in TT, sh \in \mathbf{A}, T_j \neq sh \wedge T_{t(j+1)} = sh \tag{6}$$

$$T_j = sh_1 \rightarrow T_{t(j+1)} \neq sh_2, \quad j \in TT, (sh_1, sh_2) \in F_2 \tag{7}$$

$$T_j = sh_1 \wedge T_{t(j+1)} = O \rightarrow T_{t(j+2)} \neq sh_2, \quad j \in TT, (sh_1, sh_2) \in F_3 \tag{8}$$

Constraint (1) enforces there are no sequences of length $u_w + 1$ with no O shift, *i.e.*, the maximum length of a work block. Constraint (2) enforces there are no sequences of length less than l_w of work shifts, *i.e.*, the minimum length of a work block. Constraint (3) enforces there are no sequences of length $u_O + 1$ of just O shifts, *i.e.*, the maximum length of an off block. Constraint (4) enforces there are no sequences of length less than l_O of off shifts, *i.e.*, the minimum length of

an off block. Constraint (5) enforces there are no sequences of length $u_{sh}+1$ of just sh shifts, *i.e.*, the maximum length of an sh block. Constraint (6) enforces there are no sequences of length less than l_{sh} of sh shifts, *i.e.*, the minimum length of an sh block. Constraint (7) enforces no forbidden sequences of length 2. Constraint (8) enforces no forbidden sequences of length 3.

To complete the model, we enforce that each day has the correct number of each type of shift.

$$\sum\nolimits_{i\in 1..n}(S_{i,j}=sh)=R_{sh,j},\quad j\in 1..w, sh\in\mathbf{A} \tag{9}$$

We can do the same for the off shifts as follows. Note that it is a redundant constraint.

$$\sum\nolimits_{i\in 1..n}(S_{i,j}=O)=o_j,\quad j\in 1..w \tag{10}$$

Note that this model appears to consist entirely of linear constraints (at least once we use 01 variables to model the decisions $S_{i,j}=sh, sh\in\mathbf{A}^+$). This is misleading, since equations (2), (4) and (6) are all contingent on variable conditions. The entire model can be easily expressed with linear constraints, and (half-)reified linear constraints.

4 Alternative model choices

The direct model (1–10) described in the previous section simply uses linear-styled constraints. However, there are alternative ways to model the shift transitions and the temporal requirements using global constraints. In addition, we can add more redundant constraints and symmetry breaking constraints to the model. In this section, we look at these choices except for the shift transition, for which we devote a separate section after this one.

4.1 Temporal requirements

Instead of using the linear constraints in (9) and (10), we can respectively use these global cardinality constraints for each week day $j\in 1..w$.

$$gcc\_low\_up([S_{i,j}|i\in 1..n],\mathbf{A},[R_{sh,j}|sh\in\mathbf{A}],[R_{sh,j}|sh\in\mathbf{A}]) \tag{11}$$

$$gcc\_low\_up([S_{i,j}|i\in 1..n],\mathbf{A}^+,[R_{sh,j}|sh\in\mathbf{A}^+],[R_{sh,j}|sh\in\mathbf{A}^+]) \tag{12}$$

They state that the number of shifts of each type in $sh\in\mathbf{A}$ (or $sh\in\mathbf{A}^+$) occurring in each day j must exactly equal the requirement $R_{sh,j}$.

4.2 Redundant constraints

In a cyclic schedule, we know that there are equal numbers of work blocks and off blocks. We exploit this knowledge to create redundant constraints for each week by ensuring the lower bounds and upper bounds of these blocks do not cross.

Let $twl = \sum_{sh \in \mathbf{A}} \sum_{j=1}^{n} R_{sh,j}$ be the total workload over the planning period. Then we can define the number of days-off ow_i at the end of the week i from the beginning of the schedule as

$$ow_i = \begin{cases} 0 & i = 0 \\ ow_{i-1} + \sum_{j \in 1..w} (S_{i,j} = O) & i \in 1..n-1 \\ n \times w - twl & i = n \end{cases}$$

Define ro_i to be the number of days-off remaining after the end of week i, and similarly rw_i to be the number of work days remaining after the end of week i

$$ro_i = n \times w - twl - ow_i, \qquad rw_i = twl - w \times i + ro_i.$$

We can determine a lower bound lo_i for the number of remaining off blocks starting from week i, and similarly an upper bound uo_i for the number of remaining off blocks as:

$$lo_i = \lceil ro_i / u_O \rceil - (S_{1,1} \neq O \wedge S_{i+1,1} = O)$$
$$uo_i = \lfloor ro_i / l_O \rfloor + (S_{1,1} = O \wedge S_{i,w} = O)$$

Note that the potential additional minus and plus one from the evaluation of the conjunction accounts for the fact that the number of work and off blocks can differ by one in the remaining schedule. For the lower bound, when the schedule starts with a work day and the week after the week i with a day-off then there might be one off block more in the remaining schedule than the number of remaining work blocks. For the upper bound, if the schedule starts with an off day and the week i ends with a day-off then there might be one off block less in the remaining schedule than the number of remaining work blocks.

Similarly, we can compute a lower bound lw_i for the number of the remaining work blocks after the end of week i, and similarly an upper bound uw_i for the number of remaining work blocks as:

$$lw_i = \lceil rw_i / u_w \rceil - (S_{1,1} = O \wedge S_{i+1,1} \neq O)$$
$$uw_i = \lfloor rw_i / l_w \rfloor + (S_{1,1} \neq O \wedge S_{i,w} \neq O)$$

Finally, we constrain these bounds to agree.

$$lo_i \leq uw_i \wedge lw_i \leq uo_i, \qquad i \in 1..n \tag{13}$$

4.3 Symmetry breaking constraints

Given a schedule S a symmetric solution can easily be obtained by shifting the schedule by any number of weeks. If there must be at least one off day at the end of the week then we impose that the last day in the schedule is an off day. Note that it happens for all instances used.

$$o_w > 0 \rightarrow S_{n,w} = O \tag{14}$$

Note that we could have chosen any day and possible shift for breaking this symmetry, but we choose this one because it aligns with our other model choice for the shift transitions described in the next section.

Another symmetry occurs when all temporal requirements are the same for each day and each shift. In this case, a symmetric schedule can be obtained by shifting the schedule by any number of days. Thus, we can enforce that the work block starts at the first day in the schedule. The same constraint can be enforced if the number of working days in the first day is greater than in the last day of the week, because there must be at least one work block starting at the first day.

$$\Big(\forall sh \in \mathbf{A}, \forall j \in 1..w-1 : R_{sh,j} = R_{sh,j+1} \vee \sum\nolimits_{sh\in\mathbf{A}} R_{sh,1} > \sum\nolimits_{sh\in\mathbf{A}} R_{sh,w}\Big) \\ \rightarrow S_{1,1} \neq O \wedge S_{n,w} = O \quad (15)$$

In comparison to (14), (15) also enforces an off day on the last day and thus it is stronger symmetry breaking constraint, but less often applicable. Note that there might be further symmetries, especially instance specific ones, but here we focus on more common symmetries.

5 Automata Based Model

The automata-based model attempts to capture as much of the problem as possible in a single `regular` [23] constraint.

In order to enforce the forbidden sequences constraints we need to keep track of the last shift taken, and if the last shift was an O shift then the previous shift before that. In order to track the lower and upper bounds for each shift type, we need to track the number of consecutive shifts of a single type (including O). In order to track the lower and upper bounds for consecutive work shifts, we need to track the number of consecutive work shifts.

We define an automata M with $Q = m + u_o + \sum_{sh\in\mathbf{A}}(u_w \times u_{sh})$ states. We bracket state names to avoid ambiguity with shift types. They represent in sequence: an artificial start state [start]; states for the first O shift in a sequence, recording the type of the previous work shift $[sO]$; states for 2 or more O shifts in sequence $(2 \leq i \leq u_o)$ $[O^i]$, states encoding that the last $1 \leq j \leq u_s$ consecutive shifts are type $s \in \mathbf{A}$ directly after a sequence of $0 \leq j < u_w$ consecutive work shifts (not O) $[w^i s^j]$. Note each state effectively records a sequence of previous shifts. Note that some of the states may be useless, since, *e.g.*, a state encoding 3 consecutive D shifts after a sequence of 4 other works shifts with $o_w = 6$ is not possible (it represents 7 consecutive work shifts).

The transition function d for the states is defined as follows (missing transitions go to an error state):

- [start]: on $sh \in \mathbf{A}$ goto $[sh]$, on O goto $[OO]$. Note that transitions assume that the previous shift was O.
- $[sO]$: on O goto $[OO]$ (assuming $u_O \geq 2$), on $sh \in A$ goto $[sh]$ unless $s\ O\ sh$ is forbidden ($(s, sh) \in F_3$) or $l_O > 1$.

– $[O^i]$, $2 \le i \le u_O$: on O goto $[O^{i+1}]$ unless $i = u_O$, on $sh \in \mathbf{A}$ goto $[sh]$ unless $i < l_O$.

– $[w^i s^j]$, $0 \le i \le u_w - 1$, $1 \le j \le u_s$: on O goto $[sO]$ unless $j < l_s$, on s goto $[w^i s^{j+1}]$ unless $j = u_s$ or $i + j \ge u_w$, on $sh \in \mathbf{A} - \{s\}$ goto $[w^{i+j} sh]$ unless $s\ sh$ is forbidden ($(s, sh) \in F_2$) or $i + j \ge u_w$ or $j < l_s$.

Each state is accepting in this automata.

An example automata with two shifts D (Day) and N (Night) with forbidden sequences ND and DON and limits $l_D = 2$, $u_D = 3$, $l_N = 1$, $u_N = 2$, $l_O = 1$, $u_O = 3$ and $l_w = 2$, $u_w = 4$ is shown in Figure 1. Unreachable states are shown dotted, and edges from them are usually omitted, except horizontal edges which do not break the total work limit. Edges for D shifts are full, N are dashed and O are dotted.

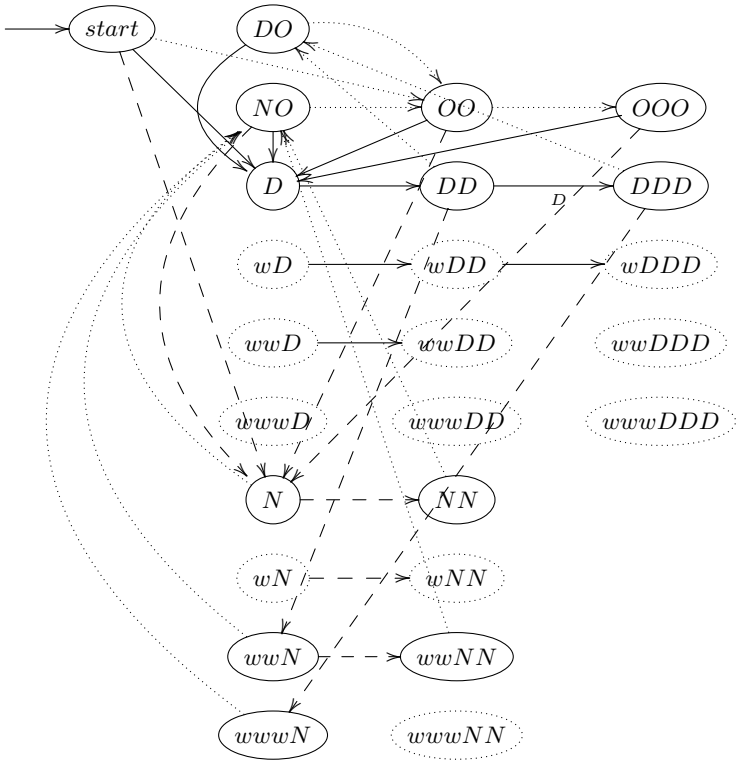

Fig. 1. The automaton capturing correct shift sequences for a problem with work shifts D and N, and forbidden sequences ND and DON. D shifts are indicated by full arrows, N shifts by dashed arrows, and O shifts by dotted arrows.

[**start**]	D	[**D**]	D	[**DD**]	N	[**wwN**]	O	$[NO]$	O	$[OO]$	N	$[NO]$	N	
$[NN]$	O	$[NO]$	D	$[D]$	D	$[DD]$	D	$[DDD]$	O	$[DO]$	O	$[OO]$	D	
[**D**]	D	[**DD**]	D	[**DDD**]	N	[**wwwN**]	O	$[NO]$	O	$[OO]$	N	$[NO]$	N	

Fig. 2. A two week sequence $DDNOONNODDDOOD$ with the first week repeated illustrating how the `regular` constraint can be in different states in the two copies of the first week.

In order to define a cyclic schedule the `regular` constraint is applied on a sequence that duplicates the first w shifts at the end. This is safe assuming that $u_w < w$, which occurs in all our examples.

The remaining constraints simply enforce the correct number of each shift type on each day. In summary the total model consists of either Equation (11) or Equation (12) together with

$$\texttt{regular}([S_{1,1}, \ldots, S_{n,w}, S_{1,1}, \ldots, S_{1,w}], Q, m+1, d, [start], 1..Q) \tag{16}$$

that is a `regular` constraint over the $n+1$ week extended shift list, using an automata defined by Q states, $m+1$ shift possibilities, transition function d, start state $[start]$ will all states being final states.

Note that the states in the first week, and the copy need not be the same. For example, given the automaton of Figure 1, Figure 2 shows a two week schedule, with the first week repeated, giving the state of the automata across the schedule. The two bold subsequences show where the states are different for the two copies of the first week. The sequence is accepted.

A simpler model would be possible if we had a `regular` constraint that included start and end states as variable arguments. We could then simply constrain the original array of shifts (with no duplication of the first week) and constrain the start and end states to be identical. Current solvers do not support such a `regular` constraint.

Note that while the start state ambiguity means that we make assumptions about the previous state when evaluating the automata on the states until we reach a first O shift, because the constraints are applied twice on the first week of shifts the proper constraints are satisfied. Note also that the `regular` constraint may actually remove solutions, since the first shift is assumed to be the first after an off, for the problems we tackle this simply removes symmetric solutions. This can be incorrect for corner cases, *e.g.*, when $\sum_{s \in \mathbf{A}} R_{s,w} = n$ (so there can be no O shift in the last day of the week). In these cases we can rotate the shift requirements for the days (effectively starting the cycle on a different day) to overcome the problem (it does not occur in any of the benchmarks).

We tried an alternate automata M' which ignores constraints until it can be sure of which state it is in, *i.e.*, a work shift followed by an O shift, or two consecutive O shifts. This proved to be terrible since it allowed erroneous schedules in the first week which then only detected as unsatisfiable when we finish labeling the last week.

6 Search Strategies

Beside the solver's default search strategy, we tested several others for the CP solver used, which are briefly described in this section.

The strategies studied consist of a variable and value selection part, which can be combined freely.

6.1 Variable selection

Variable selection is critical for reducing the search space for any combinatorial problem. We need to balance the criteria of driving quickly towards failure, with getting the most possible inference from the solver. The key decisions of the model are the schedule variables $S_{i,j}$. We define our variable selection over these variables unless stated otherwise. Ties are broken by input order.

default: Solver's default selection.
random: Randomly select a variable.
worker: Select the variables in the chronological order over the planning horizon, *i.e.*, $S_{1,1}, \ldots, S_{1,w}$, $S_{2,1}, \ldots, S_{2,w}$, $\ldots$, $S_{n,1}, \ldots, S_{n,w}$.
day: Select the variables in the following order the first day of the week from the first to the last week in the planning horizon and then the next day and so on, *i.e.*, $S_{1,1}, \ldots, S_{n,1}$, $S_{1,2}, \ldots, S_{n,2}$, $\ldots$, $S_{1,w}, \ldots, S_{n,w}$.
wd: Select the variables of the first day in the weeks in the chronological order, *i.e.*, $S_{1,1}, \ldots, S_{n,1}$, and then use the variable selection **worker**.
ff: Select the variable with the smallest domain (first fail).
first: Create new Boolean variables $b_k \leftrightarrow T_{t(k-1)} \neq T_{t(k)}$, $k \in TT$. These represent where a change of shift type occurs. Select the Boolean variables in chronological order and assign the value *true* at first. Then use the variable selection **worker**.
s1: Create new Boolean variables $b_k \leftrightarrow (T_k = O)$, $k \in TT$. These represent where an off shift occurs. Select the Boolean variables in chronological order and assign the value *true* at first. Then use the variable selection **worker**.

6.2 Value selection

All instances in the benchmark set have these shift types (D) day, (A) afternoon, and (N) night, as well as the day-off (O) shift. Hence we could consider any of the 24 different static value ordering amongst these four. We consider 4 static orderings: *DANO*, default ordering of the model `indomain_min`; *ODAN*, off shifts first; *ONAD*, reversed default ordering `indomain_max`; *NADO*, reversed ordering of shifts.

We also consider static orderings that are computed from features of the instance to be solved. Let *maxb* be the maximal number of work or off blocks, calculated as

$$maxb = \max(\lceil (\sum\nolimits_{sh \in \mathbf{A}, j \in 1..n} R_{sh,j})/l_w \rceil, \lceil (\sum\nolimits_{j \in 1..n} R_{O,j})/l_O \rceil).$$

We consider two variants of ordering shift types in terms of the tightness of the number of shifts required compared to the minimum number required to be scheduled. Both variants are in ascending order.

slack1: For each shift in $\mathbf{A}^+$, we compute the slack between maximal available space, *i.e.*, number of shift blocks times the maximal shift length, and the required workload or "days-off-load", *i.e.*, $sl^1_O = u_O \times maxb - \sum_{j=1}^{w} o_j$, and $sl^1_{sh} = \min(maxb, \sum_{j=1}^{w} R_{sh,j}/l_{sh}) \times u_{sh} - \sum_{j=1}^{w} R_{sh,j}$, $sh \in \mathbf{A}$.

slack2: This variant refines **slack1** for the work shifts. In addition, it considers when the maximal space is restricted by the high block requirement of the other work shifts, *i.e.*, $sl^2_O = sl^1_O$, and $sl^2_{sh} = \min(sl^1_{sh}, ti_{sh})$, $sh \in \mathbf{A}$ where $ti_{sh} = u_w \times (maxb - \sum_{z \in A \setminus \{sh\}} \lceil \sum_{j=1}^{w} R_{z,j}/u_z \rceil) - \sum_{j=1}^{w} R_{sh,j}$.

7 Experiments

To evaluate our methods, we took all 20 instances from a standard benchmark set[4] and further 30 hard instances from 1980 additional generated instances, for which the heuristic MC-T [19,20] either did not find a solution or required a long time for it. The instances generated consist of 9 to 51 employees, 2 to 3 shift types, 3 to 4 minimal and 5 to 7 maximal length of work blocks, 1 to 2 minimal and 2 to 4 maximal length of days-off blocks, and minimal and maximal length of periods of consecutive shifts (D: 2 to 3 and 5 to 7, A: 2 to 3 and 4 to 6, N: 2 to 3 and 4 to 5). The same forbidden sequences as for real-life examples are used. Initially the temporal requirements for shifts are distributed randomly between shifts based on the total number of working days and days-off (the number of days-off is set to $\lfloor n \times w \times 0.2857 \rfloor$). With probability 0.3 the temporal requirements during weekend are changed (half of these duties are distributed to the temporal requirements of the weekdays).

Experiments were run on Dell PowerEdge M630 machines having Intel Xeon E5-2660 V3 processors running at 2.6 GHz with 25 MB cache, unless otherwise stated. We imposed a runtime limit of one hour and a memory limit of 16 GB, unless otherwise stated. We tested Gurobi as a MIP solver and Chuffed [6] as a CP solver.[5] The development version of MiniZinc was used for modeling.

Table 2 compares the impact of the different model combinations for the temporal requirement and shift transition constraints for Gurobi and Chuffed using the solvers' default search strategy. The columns show in this order the solver, the constraints used (model), the total number of solved instances (#tot), the average number of nodes (avg. nd), the average runtime (avg. rt) (including time-outs for unsolved instances), the number of instances for which the solver found a solution (#sat), the average runtime of feasible instances (avg. rt), the number of instances for which the solver proved infeasibility (#uns), and the average runtime of infeasible instances (avg. rt). The results for Gurobi

[4] Available at http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/.

[5] We also tried Gecode as a constraint programming solver, but it was not competitive.

Table 2. Results on different constraint choices.

solver	model	#tot	avg. nd	avg. rt	#sat	avg. rt	#uns	avg. rt
Gurobi	(1–8), (9)	46	3.2k	780.6s	40	587.5s	6	1794.2s
Gurobi	(1–8), (9,10)	44	2.4k	789.2s	39	533.5s	5	2131.6s
Gurobi	(1–8), (11)	42	3.3k	891.8s	38	629.7s	4	2268.0s
Gurobi	(1–8), (12)	43	3.5k	841.0s	38	639.7s	5	1897.7s
Gurobi	(16), (9)	**50**	177	50.1s	**42**	53.6s	**8**	32.0s
Gurobi	(16), (9,10)	**50**	471	92.7s	**42**	105.9s	**8**	**23.2s**
Gurobi	(16), (11)	**50**	177	49.5s	**42**	52.8s	**8**	32.1s
Gurobi	(16), (12)	**50**	113	**45.0s**	**42**	**48.6s**	**8**	26.3s
Chuffed	(1–8), (9)	33	9.1m	1323.8s	33	890.1s	0	3600.6s
Chuffed	(1–8), (9,10)	44	2.8m	505.3s	39	342.5s	5	1360.3s
Chuffed	(1–8), (11)	37	9.9m	1070.6s	36	644.2s	1	3308.9s
Chuffed	(1–8), (12)	44	4.9m	482.8s	39	315.5s	5	1361.2s
Chuffed	(16), (9)	30	5.7m	1539.2s	30	1146.5s	0	3600.8s
Chuffed	(16), (9,10)	44	1.6m	521.1s	39	303.2s	5	1665.6s
Chuffed	(16), (11)	34	4.3m	1304.5s	34	867.2s	0	3600.4s
Chuffed	(16), (12)	42	1.8m	615.5s	37	469.2s	5	1383.7s

are clear. The shift transition constraints are the key constraints for its performance. Using the `regular` constraint (16), it solves all 50 instances regardless of the constraints for the temporal requirements. Its superiority over the direct representation results because MiniZinc transforms the `regular` constraint into a network flow for mixed-integer solvers [3] and hence almost the entire model is totally unimodular. The overall best combination is achieved with the global cardinality constraint (12).

By contrast, Chuffed is not able to solve all instances in any combination and there are two important model choices. As opposed to Gurobi, Chuffed performs better when using the direct constraints (1–8) for the shift transition constraints, even though weaker propagation is achieved. The average number of nodes indicate that a stronger propagation of the `regular` does not convert into runtime savings. This probably results from the fact that the direct constraints introduce intermediate variables which are valuable for learning, whereas the `regular` constraint introduces no intermediate variables. For Chuffed, it is also important to choose temporal constraints covering the days-off. The overall best combination are the direct constraints (1–8) for the shift transition constraints and the global cardinality constraint (12).

Table 3 shows the impact on the performance of Gurobi and Chuffed when adding the redundant and symmetry breaking constraints to the best model combination. Gurobi's performance drastically deteriorate when using the redundant constraints. This is unsurprising since these are mainly linear combinations of constraints the solver already has. Its performance significantly improves when using the symmetry breaking constraints, but only for infeasible instances, which is expected because it makes the search space smaller. For feasible instances, it does not have any impact. For Chuffed, both set of constraints are important

Table 3. Results with redundant (13) and symmetry breaking (14,15) constraints.

solver	model	(13)	(14,15)	#tot	avg. nd	avg. rt	#sat	avg. rt	#uns	avg. rt
Gurobi	(16), (12)			**50**	113	45.0s	**42**	48.6s	**8**	26.3s
Gurobi	(16), (12)		×	**50**	84	**41.4s**	**42**	48.4s	**8**	**4.5s**
Gurobi	(16), (12)	×		**50**	271	93.5s	**42**	108.2s	**8**	16.0s
Gurobi	(16), (12)	×	×	49	374	107.6s	41	127.3s	**8**	4.5s
Chuffed	(1–8), (12)			44	4.9m	482.8s	39	315.5s	5	1361.2s
Chuffed	(1–8), (12)		×	44	5.1m	489.1s	39	323.9s	5	1356.2s
Chuffed	(1–8), (12)	×		48	1.6m	190.1s	**42**	51.9s	6	915.4s
Chuffed	(1–8), (12)	×	×	48	1.8m	172.5s	**42**	**32.2s**	6	909.2s

Table 4. Results on best value and variable selections for the search strategies.

solver	search	#tot	avg. nd	avg. rt	#sat	avg. rt	#uns	avg. rt
Chuffed	**default**+*DANO*	**48**	1.8m	172.5s	**42**	32.2s	**6**	909.2s
Chuffed	**worker**+*NADO*	**48**	1.4m	169.7s	**42**	28.4s	**6**	911.1s
Chuffed	**ff**+*NADO*	**48**	1.4m	**166.7s**	**42**	**25.1s**	**6**	909.8s
Chuffed	**s1**+*DANO*	47	1.3m	236.4s	41	109.0s	**6**	**905.2s**

to increase its performance. Still it cannot solve all instances in the given runtime limit, but could find a solution for all feasible instances. Interestingly, the average runtime over all feasible instances is lower than Gurobi's best time.

Table 4 shows the best pairing of value and variable selections for the search strategies for Chuffed.[6] Chuffed alternates between the given search strategy and its default one on each restart. This is important since it allows the powerful default activity based search to be utilized.

Using a search strategy was beneficial for the performance, but either for feasible or infeasible instances, and not both together. For feasible instances, the variable selections **ff** and **worker** were the best two in combination with a value ordering *NADO*, because first fail makes the search space small and worker explores it in chronological order of the schedule allowing removal of impossible values for the next decision. The value ordering *NADO* works best, because of the structure of the instances. The night shift is the most restricted one for the shift transitions and then the afternoon. In addition, the temporal requirements tends to be the least for the night shift and then the afternoon shift. For infeasible instances, deciding work and days-off at first and then which work shift is performed using the value ordering *DANO* performed best.

Because the instances all have very similar relations between the different types of shifts, we took the 50 instances and reversed the order of the forbidden sequences to check that the value ordering *NADO* is not necessarily the best one. We set a runtime limit to 300s for this experiment. Table 5 clearly shows that the dynamic criteria **slack2** performs the best, which looks at the tightness of the temporal requirements for each shift including the days-off. In comparison

[6] We ran preliminary experiments on all possible combinations with a five minutes runtime limit.

Table 5. Results on instances with reversed forbidden sequences (runtime limit 300s).

solver	search	#tot	avg. nd	avg. rt	#sat	avg. rt	#uns	avg. rt
Chuffed	**ff+***ODAN*	**45**	304k	49.2s	**34**	33.7s	**11**	**85.4s**
Chuffed	**ff+***DANO*	**45**	303k	40.5s	**34**	21.0s	**11**	86.1s
Chuffed	**ff+***ONAD*	44	277k	45.1s	33	27.5s	**11**	86.1s
Chuffed	**ff+***NADO*	**45**	260k	40.1s	**34**	20.6s	**11**	85.6s
Chuffed	**ff+slack1**	**45**	244k	40.1s	**34**	20.4s	**11**	85.8s
Chuffed	**ff+slack2**	**45**	235k	**37.9s**	**34**	**17.0s**	**11**	86.9s

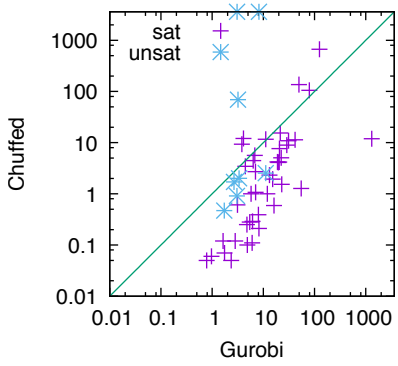

Fig. 3. Runtime comparison between Chuffed (y-axis) and Gurobi (x-axis).

Table 6. Comparison to the state of the art methods on all 2000 instances (runtime limit 200s).

solver	#fastest	#tot	avg. rt	#sat	avg. rt	#uns	avg. rt
Gurobi	31	**1988**	**10.3s**	1320	13.0s	**668**	**4.9s**
Chuffed	513	1845	19.4s	**1322**	**5.0s**	523	47.8s
MathSAT - BV	3	1470	63.8s	1198	32.1s	272	126.7s
MC-T	**781**	1212	82.7s	1212	23.3s	0	200s

to **slack2**, Chuffed is about 20% slower when using *NADO*, which confirms our previous observation that this order is well-suited for the instances in the benchmark library, due to the fact that the night shifts and then the afternoon shifts are normally the most-restrictive ones.

Figure 3 shows the runtime comparison between Chuffed and Gurobi on the 50 instances. Runtimes are given in seconds and the axis use a logarithmic scale. Points below the diagonal line express that Chuffed solved the instance quicker than Gurobi, and vice-versa for points above the line. Except for 5 feasible and 3 infeasible instances, Chuffed solved the instances in a similar speed or faster, even by order of a magnitude for many instances. However, Gurobi is more robust, especially for infeasible instances, which were solved within 10 seconds.

Table 6 compares the best Chuffed and Gurobi outcome on all 2000 instances to the state-of-the-art heuristic approach based on min-conflicts heuristic and tabu search (MC-T) [19,20] and the state-of-the-art complete SMT approach [11] using bit vectors for modeling and the SMT solver MathSAT 5.5.1[8] for solving. Note that the results of MC-T reported in this paper were obtained on a Lenovo T440s machine having Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz with 8 GB RAM. To be conservative, we consider that these machines are twice as slow as the machines on which Gurobi, Chuffed and MathSAT were executed. For this comparison we set the runtime limit to 200s and recorded the number of instances (#fastest), for that each method was the fastest, we halved the runtime of MC-T for computing #fastest. Chuffed and Gurobi significantly outperform the state-of-the-art complete methods in all aspects. The heuristic solver performs better than the SMT method on feasible instances, even though it runs on a slower machine. It is the fastest solver on almost 59% of the feasible instances followed by Chuffed with 39%. However, it cannot compete with Chuffed and Gurobi in term of the number of solved feasible instances. Chuffed and Gurobi were respectively able to solve more feasible instances within 10s and 30s than MC-T within 200s. On top of that, our methods are able to detect infeasibility. Thus, both our methods are more robust than MC-T, whereas Gurobi is the most robust one. Together, both our methods could solve all instances within 200s, except two instances.

8 Conclusion

We investigated different solver-independent models for solving the rotating workforce scheduling problem using MiniZinc [22]. Surprisingly the best model combination resulted when using `regular` constraints with Gurobi, even though the `regular` constraint is native to constraint programming. This shows the advantages of solver-independent modeling, where we do not commit to a single solver. While `regular` and network-flow models have been used previously for this problem, they made use of multiple `regular` constraints instead of one large `regular`. The advantage of using a high level modeling language was that we could generate a complex automata fully automatically that encoded almost all of the problem. Indeed a simple first version of the automata based model was constructed in under an hour. We tested our approaches on the standard benchmark set and created more challenging instances for our evaluation. Interestingly, Gurobi and Chuffed excelled on different model combinations. On the majority of instances Chuffed is the quickest solver, but Gurobi the most robust one, because of its superiority in proving infeasibility.

Acknowledgments This work was partially supported by the Asian Office of Aerospace Research and Development grant 15-4016 and by the Austrian Science Fund (FWF): P24814-N23.

References

1. Baker, K.R.: Workforce allocation in cyclical scheduling problems: A survey. Journal of the Operational Research Society 27(1), 155–167 (1976)
2. Balakrishnan, N., Wong, R.T.: A network model for the rotating workforce scheduling problem. Networks 20(1), 25–42 (1990)
3. Belov, G., Stuckey, P.J., Tack, G., Wallace, M.G.: Improved linearization of constraint programming models. In: Rueher, M. (ed.) Principles and Practice of Constraint Programming – CP 2016. pp. 49–65. Springer International Publishing, Cham (2016)
4. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research 226(3), 367–385 (2013)
5. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. Journal of scheduling 7(6), 441–499 (2004)
6. Chu, G.: Improving Combinatorial Optimization. Ph.D. thesis, The University of Melbourne (2011), http://hdl.handle.net/11343/36679
7. Chuin Lau, H.: On the complexity of manpower shift scheduling. Computers & operations research 23(1), 93–102 (1996)
8. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: A modular approach to maxsat modulo theories. In: Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings. pp. 150–165 (2013)
9. Côté, M.C., Gendron, B., Quimper, C.G., Rousseau, L.M.: Formal languages for integer programming modeling of shift scheduling problems. Constraints 16(1), 54–76 (2011)
10. Côté, M.C., Gendron, B., Rousseau, L.M.: Grammar-based integer programming models for multiactivity shift scheduling. Management Science 57(1), 151–163 (2011)
11. Erkinger, C., Musliu, N.: Personnel scheduling as satisfiability modulo theories. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 614–621 (2017), https://doi.org/10.24963/ijcai.2017/86
12. Falcón, R., Barrena, E., Canca, D., Laporte, G.: Counting and enumerating feasible rotating schedules by means of Gröbner bases. Mathematics and Computers in Simulation 125, 139–151 (2016)
13. Gärtner, J., Musliu, N., Slany, W.: Rota: a research project on algorithms for workforce scheduling and shift design optimization. AI Communications 14(2), 83–92 (2001)
14. Hashemi Doulabi, S.H., Rousseau, L.M., Pesant, G.: A constraint-programming-based branch-and-price-and-cut approach for operating room planning and scheduling. INFORMS Journal on Computing 28(3), 432–448 (2016)
15. Kadioglu, S., Sellmann, M.: Efficient context-free grammar constraints. In: AAAI. pp. 310–316 (2008)
16. Laporte, G.: The art and science of designing rotating schedules. Journal of the Operational Research Society 50, 1011–1017 (9 1999)
17. Laporte, G., Nobert, Y., Biron, J.: Rotating schedules. European Journal of Operational Research 4(1), 24–30 (1980)
18. Laporte, G., Pesant, G.: A general multi-shift scheduling system. Journal of the Operational Research Society 55(11), 1208–1217 (2004)

19. Musliu, N.: Combination of local search strategies for rotating workforce scheduling problem. In: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005. pp. 1529–1530 (2005), http://ijcai.org/Proceedings/05/Papers/post-0448.pdf
20. Musliu, N.: Heuristic methods for automatic rotating workforce scheduling. International Journal of Computational Intelligence Research 2(4), 309–326 (2006)
21. Musliu, N., Gärtner, J., Slany, W.: Efficient generation of rotating workforce schedules. Discrete Applied Mathematics 118(1-2), 85–98 (2002)
22. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer Berlin Heidelberg (2007)
23. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M.G. (ed.) Principles and Practice of Constraint Programming – CP 2004. pp. 482–495. No. 3258 in LNCS, Springer, Berlin, Heidelberg (2004)
24. Quimper, C.G., Rousseau, L.M.: A large neighbourhood search approach to the multi-activity shift scheduling problem. Journal of Heuristics 16(3), 373–392 (2010)
25. Restrepo, M.I., Gendron, B., Rousseau, L.M.: Branch-and-price for personalized multiactivity tour scheduling. INFORMS Journal on Computing 28(2), 334–350 (2016)
26. Salvagnin, D., Walsh, T.: A hybrid MIP/CP approach for multi-activity shift scheduling. In: Principles and practice of constraint programming – CP 2012. pp. 633–646. Springer (2012)
27. Triska, M., Musliu, N.: A constraint programming application for rotating workforce scheduling. In: Developing Concepts in Applied Intelligence, Studies in Computational Intelligence, vol. 363, pp. 83–88. Springer Berlin / Heidelberg (2011)