Feasibility of Building Better Traincrew Rosters with Complete Solvers

Alan M. Frisch[1] and Miquel Palahí[1] *

Artificial Intelligence Group, Dept. of Computer Science, Univ. of York, UK
Alan.Frisch@york.ac.uk whiteshadow.mps@gmail.com

1 Introduction

The commercial benefits of a well-designed base roster for passenger traincrew can be enormous, but producing efficient base rosters is an expensive and time-consuming task. With existing processes and tools the efficiency of the finished roster is largely determined by the skill of the roster clerks involved. Existing tools, such as TRACSRoster[1] and CREWS,[2] use local search and frequently produce rosters that need to be manually repaired or improved. There is a market demand for better tools to make this process more effective and reliable.

This work presented here studied the feasibility of re-engineering TRACSRoster, the Tracsis PLC traincrew rostering product, to produce better rosters and to be easier to modify and maintain. TRACSRoster uses local search to find rosters with no guarantee of quality. This paper considered the feasibility of basing TRACSRoster on state-of-the-art complete solvers that perform exact optimisation. The paper presents and evaluates models for both SAT Modulo Theories and finite domain constraint programming.

We have identified three techniques that enable a SMT to perform much better on the train crew rostering problem. Nonetheless, the problem remains very difficult for SMT. While we can solve some smaller instances to optimality, the large instance remains unsolved.

Problem instances and models associated with this research can be found at http://www.cs.york.ac.uk/aig/constraints/SMT/.

2 The Traincrew Rostering Problem

The Traincrew Rostering Problem is a constrained optimisation problem that involves finding an optimal roster, or—more realistically—one that is good enough

* We are extremely grateful to David Turner of Tracsis who suggested we try this problem, patiently explained the ins and outs of traincrew rostering, provided real data and gave us access to TRACSRoster for testing. We thank the Yorkshire Innovation Fund for funding this project. For their help in setting up this project we thank Rukmal Abeysekera and Katie Wytwyckyj, both of the University of York Research and Enterprise office.

[1] TRACSRoster is a product of Tracsis PLC. For further information see http://www.tracsis.com/software/tracsroster .

[2] CREWS is a product of SISCOG – Sistemas Cognitivos, SA. For further information see http://www.siscog.eu/subarea.asp?idSubArea=36&idArea=2 .

to be approved by the crew's union. The crew on a train includes, possibly among others, drivers, guards, train managers and catering staff. Each of these types of crew members are rostered separately, but the form of the problem is the same across all crew types. Drivers are the most challenging crew type to roster as the most constraints apply to them. The problem instances we consider are all for driver rostering. Therefore, the rest of this paper discusses drivers, but the reader should bear in mind that scheduling other crew types is essentially the same problem.

This remainder of this section describes the traincrew rostering problem in three stages: a basic decision problem that entails finding a feasible solution; the full decision problem that also entails finding a feasible solution; and an optimisation problem of finding the feasible solution that is optimal.

2.1 The Basic Decision Problem

On each day of the week a set of jobs must be performed. Each job, called a "turn," comprises a work shift to be performed by a single driver. As an example, a Monday turn could comprise driving an 8:00 train from the depot to station B, taking a lunch break, driving a 13:30 train to station C and then taking a taxi back to the depot arriving at 16:50. For the purpose of rostering, the particular work to be performed within the turn is irrelevant; all that matters is that this turn involves work that runs from 8:00 to 16:50 on Monday. A turn such as this is called a working turn. There are two other types of turns: a training turn in which the driver undertakes training and a spare turn in which the driver must be available to substitute for an absent driver. For the decision problem all three types of turn are treated the same; however they can be treated differently by the objective function in the optimisation problem.

An instance of the problem specifies a finite set T of turns and a finite set D of drivers. Define a "slot" as a pair comprising a driver and a day of the week. Hence a problem instance has $7 * |D|$ slots. A feasible solution assigns to each slot either a single turn or a rest day and each turn must be assigned to exactly one slot on the appropriate day.

As an example, consider an instance with 17 turns, $T = \{t_1, t_2, \ldots, t_{17}\}$, and 3 drivers, $D = \{d_1, d_2, d_3\}$. Suppose that t_1, t_2, t_3 are Monday turns; t_4, t_5, t_6 are Tuesday turns; t_7, t_8, t_9 are Wednesday turns; t_{10}, t_{11}, t_{12} are Thursday turns; t_{13}, t_{14}, t_{15} are Friday turns; t_{16} is a Saturday turn; and t_{17} is a Sunday turn. Then one feasible solution is displayed in Figure 1, where drivers d_1, d_2 and d_3 operate Line 1, Line 2 and Line 3, respectively.

	Mon	Tues	Wed	Thur	Fri	Sat	Sun
Line 1	t_1	t_4	t_7	t_{10}	t_{13}	rest	rest
Line 2	t_2	t_5	t_8	t_{11}	t_{14}	t_{16}	rest
Line 3	t_3	t_6	t_9	t_{12}	t_{15}	rest	t_{17}

Fig. 1. A feasible roster.

Figure 1 shows one week of work for each of three drivers. However, drivers are scheduled for 6 months at a time. This roster is extended to a 6-month schedule by rotating the drivers through the lines. Thus, the second week of the schedule is obtained by rotating d_1 to Line 2, d_2 to Line 3 and d_3 to Line 1. In general the schedule for each week is obtained by rotating the schedule from the previous week. Any feasible solution to an instance with n drivers can be displayed as a table with n lines in which each slot is labeled with either a turn or "rest" and in which each turn appears in exactly one slot.

2.2 The Full Decision Problem

A sequence of lines through which the drivers rotate each week is called a link. Because the schedule rotates, each driver eventually operates every turn in the link and so each driver must be qualified to operate each turn. In all but the simplest problem instances many drivers are not qualified to operate every turn. In such cases a roster is formed with multiple links. There are also other reasons for forming the lines into multiple lines.

To obtain multiple links, each instance specifies how many links must be rostered, and how many lines are in each link and which qualifications are required of all drivers operating that link. So every turn is associated with a set of qualifications (those which are required to operate the turn) and every link is associated with a set of qualifications (those which are required by all drivers in the link). We say that the qualifications of a slot are those of the link it is in. Now a feasible roster must meet one additional constraint: a turn t with qualifications Q must be assigned to a slot s whose qualifications are a superset of Q. We say that s and t are *compatible*.

An instance comprises:

- T, a finite set of turns, each of which has a day of the week, a start time, an end time and a set of qualifications;
- $Links$, a finite set of links, each of which has a number of lines and a set of qualifications.

Let a slot be a triple comprising a link, a line number and a day of the week. A feasible solution to an instance of the train rostering problem assigns every turn t to a single slot s such that

- No two turns are assigned to the same slot;
- The day of t is the same as the day of s; and
- The qualifications of t are a subset of the qualifications of s.

2.3 The Optimisation Problem

The optimisation problem extends the decision problem by assigning to every feasible solution an objective value that must be optimised. We have attempted to capture the objective used by TRACSRoster, which itself is ad hoc and repeatedly changing in each attempt to produce a roster that is acceptable to the drivers' unions. It is worth bearing this in mind if some aspects of the objective appear

arbitrary or ad hoc. Furthermore, we can only approximate the objective used by TRACSRoster because we do not have access to all the details of its operation.

The objective is obtained by adding to the decision problem a set of soft constraints, each associated with a penalty that is imposed if the constraint is violated by a feasible solution. The objective is to minimise the sum of all the imposed penalties. Each instance of the rostering problem can use different soft constraints. Indeed, even within a problem instance different links can use different soft constraints.

TRACSRoster supports dozens of types of soft constraints, a subset of which are used in each instance. Here are examples of some of the soft constraints that arise in the instances we have worked on so far.

- The total amount of time worked in each line has an upper and a lower bound.
- The duration between turns worked on any two consecutive days has a lower bound.
- Within each line the start times of turns differs by at most a certain amount.

One soft constraint, called a pattern constraint, is of particular interest. The pattern constraint for a link designates each slot in the link as either a work slot, a rest slot, or an "any" slot. A work slot can be labelled "duty," which indicates that the slot should be filled with any kind of duty, or "NS duty," which indicates that the slot should be filled with a non-spare duty—that is, the duty is to perform a pre-determined turn. As an example, Fig. 5 shows the pattern used in problem Instance B, which will be introduced later. It is a three-line pattern, but patterns

	Mon	Tues	Wed	Thurs	Fri	Sat	Sun
line 1	rest	rest	rest	NS duty	duty	duty	duty
line 2	any	duty	duty	rest	rest	NS duty	duty
line 3	any	duty	duty	duty	duty	rest	rest

Fig. 2. A three-line pattern.

can contain any number of lines. If the link and pattern have the same number of lines, then the pattern is overlaid directly on the link. If the link has fewer lines than the pattern, then the pattern is truncated to the same size of link and then overlaid. If, as is typically the case, the link has more lines than the pattern then the pattern is repeated until it fills the link. The general rule is that line k of the link is overlaid with line $((k+2) \bmod p)+1$ of the pattern, where p is the number of lines in the pattern.

If patterns are used on multiple links in the roster, they are applied to each link independently.

A roster is incurs a penalty for each slot in the roster that does not respect the pattern. Each violation incurs the same penalty and a slot designated "any" by the pattern never incurs a penalty.

3 Problem Instances

Our experiments were run on four real problem instances (anonymised in this paper) whose features are listed in Figure 3. For each problem instance the columns display the total number of turns, soft constraints, links and lines; the last column shows how the lines are distributed among the links. Within each of these particular instances the same same soft constraints apply to every link.

Instance B uses the three-line pattern shown in Fig. 5 and Instance D uses a six-line pattern. Within both these instances the pattern applies to every link that has three or more lines. Instances A and C use no patterns.

	Turns	Soft	Lines	Links	Lines in each link
Instance A	49	6	13	1	13
Instance B	107	16	21	4	6, 6, 7, 2
Instance C	123	1	29	2	27, 2
Instance D	661	9	159	11	41, 3, 48, 18, 3, 1, 1, 24, 1, 18, 1

Fig. 3. Characteristics of the four instances.

4 Solvers Used

We encoded all SMT models of the rostering problem in the language of WSimply [1], a tool which translates a model in its input language to the language of SMT with the QF-LIA theory (quantifier-free linear integer aritmetic) as specified by the SMTLib standard. To solve the SMT models we used Yices 1 with its QF-LIA solver [2] because it has been shown to be particularly effective across a wide range of combinatorial problems [3].

We encoded all CP models of the rostering problem in the language of MiniZinc 1.6 and used CPX as the backend solver.

All experiments have been run on a Intel® Core™CPU@2.8GHz, with 12GB of RAM.

5 Modelling the Full Decision Problem and Results

Our methodology is to first find an effective model to solve the full decision problem and then extend that model to handle the optimisation problem. This section of the paper presents and evaluates the effectiveness of a range of models for the full decision problem. We consider both SMT models and CP models.

Let S be the set of slots, T be the set of turns, L be the set of lines and D be the set of days of the week. Let $c(s)$ be the set of turns compatible with slot s and $c^+(s)$ be $c(s) \cup \{0\}$, where we use 0 to assign to any slot that is rostered as a rest day. Let $c'(t)$ be the set of slots compatible with turn t.

The models are presented using some abstract constraints that are subsequently translated to concrete constraints in a variety of ways. The abstract constraints are as follows, where B is a set of Boolean expressions:

$EO(B)$ is true if and only if exactly one expression in B is true.

$AMO(B)$ is true if and only if at most one expression in B is true.

$ALO(B)$ is true if and only if at least one expression in B is true.

$BDD(pb)$ is a Boolean encoding of the pseudo-Boolean constraint pb.

In particular, BDD encodes pb with the method of Abío et al. [4], which translates pb to a binary decision diagram (BDD) and then translates the BDD to a Boolean constraint. The method only works on pseudo-Boolean constraints of the form $\sum_i c_i \cdot b_i \leq a$, where each c_i is a positive integer, a is a non-negative integer, and each b_i is a Boolean variable.

5.1 Integer Models

For each slot $s \in S$ the integer model has as an integer variable I_s whose domain is $c^+(s)$. The only constraints in the model are to ensure that every turn in T is assigned to exactly one variable in S.

$$EO(\{I_s = t \mid s \in c'(t)\}) \quad (t \in T) \tag{1}$$

Integer Model in SMT SMT with the Quantifier-Free Linear Integer Arithmetic (QF-LIA) theory provides unbounded integer variables, so we must impose constraints to implement the finite domains. We used the following constraints:

$$0 \leq I_s \wedge I_s \leq \max(c^+(s)) \quad (s \in S) \tag{2}$$

$$\bigvee_{t \in c^+(s)} I_s = t \quad (s \in S) \tag{3}$$

Notice that (3) implies (2); however, it is advantageous to include this implied constraint as it provides information about the value of I_s to the LIA solver.

To implement the abstract constraints of (1) we decompose each EO constraint into a conjunction of an AMO constraint and an ALO constraint. We used two alternative versions of ALO and two alternative versions of AMO.

$$ALO_1(X) \stackrel{\text{def}}{=} \bigvee_{x \in X} x$$

$$ALO_2(X) \stackrel{\text{def}}{=} BDD(\sum_{x \in X} \neg x \leq |X| - 1)$$

$$AMO_1(X) \stackrel{\text{def}}{=} \sum_{x \in X} \text{if } x \text{ then 1 else 0} \leq 1$$

$$AMO_2(X) \stackrel{\text{def}}{=} BDD(\sum_{x \in X} x \leq 1)$$

In total we have four integer SMT models.

Integer Model in CP Finite domain constraint solvers directly support domains for the integer variables. So all that is needed is an implementation of the constraints of (1). We used two alternatives, one based on a global "exactly" constraint, the other based on a summation constraint.

$$\text{exactly}(1, [I_s \mid s \in c'(t)], t) \quad (t \in T) \tag{4}$$

$$\sum_{s \in c'(t)} \text{bool2int}(I_s = t) \;=\; 1 \quad (t \in T) \tag{5}$$

In total we have two integer CP models.

5.2 Boolean Models

For every slot $s \in S$ and for every turn $t \in c^+(s)$ this model has a Boolean variable $B_{s,t}$. The intended interpretation is that $B_{s,t}$ is true if and only if turn t is to take place in slot s. Constraints are needed to ensure that for every slot $s \in S$ there is exactly one $t \in c^+(s)$ such that $B_{s,t}$ is true.

$$EO(\{B_{s,t} \mid t \in c^+(s)\}) \;\; (s \in S) \tag{6}$$

Finally, the following constraints ensure that for each turn $t \in T$ there is exactly one slot $s \in c'(t)$ such that $B_{s,t}$ is true. These constraints are the Boolean correlate of (1).

$$EO(\{B_{s,t} \mid s \in c'(t)\}) \;\; (t \in T) \tag{7}$$

Boolean Model in SMT We implemented the constraints of (6) by AMO and ALO constraints. The ALO constraints are

$$\bigvee_{t \in c^+(s)} B_{s,t} \quad (s \in S) \tag{8}$$

We tried a variety of ways to implement the AMO constraints and found the most effective to be channeling to the original integer representation of the slot value. In particular, for each slot $s \in S$ we introduce a variable I_s whose domain is $c^+(s)$. Introducing these additional integer slot variables is also worthwhile as they will prove useful in modelling the optimisation problem.

The channeling constraints are

$$B_{s,t} \iff I_s = t \;\; (s \in S,\; t \in c^+(s)) \tag{9}$$

and we also add the constraints of (2). Though (2) is implied by the conjunction of (8) and (9), it is useful because it gives the LIA solver partial information about the domain of each I_s.

For implementing (7), we decomposed each EO constraint into a conjunction of an AMO constraint and an ALO constraint. We tried each of ALO_1 and ALO_2 in combination with each of AMO_1 and AMO_2.

In total we have four Boolean SMT models.

Boolean Model in CP The abstract constraints of (6) are implemented the same way as in the Boolean SMT model: introducing the integer slot variables, the channelling the constraints of (9), the ALO constraints of (8) and the implied constraints of (2).

The abstract constraints of (7) are implemented with the following constraints:

$$\sum_{s \in c'(t)} \text{bool2int}(B_{s,t}) \;=\; 1 \quad (t \in T) \tag{10}$$

We have only one Boolean Model in CP.

5.3 Using Domain Mapping to Implement the Models

The integer models of Sec. 5.1 treat each turn as an integer and each slot s as an integer variable whose domain is the set of turns $c^+(s)$. However we have not identified what integers correspond to what turns. How this correspondence is done has important consequences on how well the integer models perform.

Suppose the turns are implemented by consecutive integers starting at 1 and that this is done systematically day by day. For example, suppose that each day has exactly 10 turns with 1..10 being Monday turns, 11..10 are Tuesday turns and so on. So the domain of the Tuesday slots is $\{0, 11, 12, \ldots, 20\}$, which is not a consecutive range of integers. This is, captured by (2), which just bounds Tuesday slot to 0..20. The constraints of (3) are needed to prohibit the integer variable from taking values 1..10.

The complexity of non-consecutive integer domains is increased in an instance that has multiple links with different skill requirements. In the above example, it could be that turns 11..15 are compatible with link 1, 16..20 are compatible with link 2, $\{11, 13, 15, 17\}$ are compatible with link 3 and $\{12, 14, 16, 18\}$ are compatible with link 4.

The shortcoming of the presented SMT models is that the LIA solver has access to only (2) and thus operates without complete information about the domain. The non-consecutive integer domains could also impede that performance of a CP solver, particularly if it uses bounds consistency.

The use of non-consecutive integer domains has an additional secondary effect on the Boolean models. These models have one Boolean variable for each compatible slot and turn. However, a shortcoming arises because these Boolean models are generated by WSimply. WSimply, like other modelling languages including MiniZinc, ESSENCE, ESSENCE′ and OPL, generates collections of variables by using an array. In this case, it would be a two-dimensional Boolean array indexed by S and T. Hence, WSimply produces unnecessary Boolean variables for slots and turns that are not compatible. We set each of these unnecessary variables to FALSE as they represent rostering assignments that cannot be made.

To avoid these shortcomings of non-consecutive integer domains, we introduce a new technique, which we call *domain mapping*, that transforms our models into ones in which the integer domains are consecutive. In particular, the domain of each slot s is $0..|c(s)|$, where 0 continues to represent a rest day and each turn $t \in c(s)$ is represented by a distinct integer, denoted t_s, in $1..|c(s)|$.

This change must reflected in the implementation of each constraint.

First consider the constraints of the integer SMT and CP models. Constraint (1) now becomes

$$EO(\{I_s = t_s \mid s \in c'(t)\}) \quad (t \in T). \tag{11}$$

Constraints (2) and (3) are now replaced with a single constraint

$$0 \leq I_s \ \wedge \ I_s \leq |c(s)| \quad (s \in S). \tag{12}$$

Constraint (4) is incompatible with the use of domain mapping, so we must use constraint (5) which now becomes

$$\sum_{s \in c'(t)} \text{bool2int}(I_s = t_s) \ = \ 1 \quad (t \in T). \tag{13}$$

The Boolean models also employ the integer variables so the domain mapping can be applied there in the same way.

5.4 Experimental Evaluation of the Models

We focus this evaluation on Instance D as its decision problem is much harder than the others. If we can't quickly find feasible solutions then there is very little hope of having success with the optimisation problem.

We tried to solve a total of 21 models of Instance D: the 11 models described in Sections 5.1 and 5.2, 10 of which are then modified with domain mapping as described in Section 5.3. As described in Sec. 4 we used MiniZinc/CPX for the CP models and Yices 1 QF-LIA for the SMT models.

Of the 21 models of Instance D only two were able to find a satisfying assignment within 600 seconds (translation time plus run time). The successful models were both Boolean SMT models with domain mapping. The two successful models differ in terms of which AMO encoding they used, as shown in Figure 4.

	ALO_1	ALO_2
AMO_1	> 600	> 600
AMO_2	**27.6 (1.7)**	41.6 (6.2)

Fig. 4. Solve time of the decision problem for Instance D using the Boolean SMT model with domain mapping. The total translation time plus solve time in seconds is given. In parentheses the the solve time in seconds is given.

6 Modelling the Optimisation Problem and Results

The previous section identified the best approach to solving the full decision problem: the Boolean SMT model using ALO_1 and AMO_2. This section deals with extending that model to handle the train crew optimisation problem.

We consider the most important soft constraint to be the pattern constraint, which expresses a preference for rosters that respect a given pattern of work and rest days. It is important because in some instances it has the highest weight and in many instances the pattern aligns with other constraints, such as a preference for rosters in which there are not too many consecutive work days without a rest day.

This section proceeds by considering first the modelling of the pattern soft constraint then the modelling of all the other soft constraints. As SMT is inherently a decision problem it is necessary special methods are needed to use it for optimisation problems. The issue is considered next. Lastly the section presents experiments that evaluate the performance of the models and solving methods considered.

6.1 Modelling the Pattern Constraint as a Hard Constraint

The hard constraints only distinguish between days off duty (called rest days) and days on duty. The soft constraints distinguish three kinds of days on duty: work days during which a driver is to drive, training days and spare days in which a driver is to be available to fill in for a scheduled driver who is off work.

The most important soft constraint, in terms of having the highest weight, is the preference for rosters that respect a given pattern of work and rest days.

Because of its importance we have decided to try an approach in which we replace the soft pattern constraint with hard constraints that are satisfied by a roster if and only if that roster is an optimal solution of the soft pattern constraint.

Each day of the week is treated independently, so just consider one day. For that day let:

- nt be the number of turns,
- W be the set of working slots in the pattern and $nw = |W|$,
- R be the set of rest slots in the pattern and $nr = |R|$, and
- A be the set of anything slots in the pattern and $na = |A|$.

Exactly one of seven cases must hold and this specifies what constraints to add to the model in each case.

Case	Constraints to Impose	Violations
$nt < nw$	$s = 0$ for all $s \in (R \cup A)$ exactly nt slots in W are non-zero	$nw - t$
$nt = nw$	$s = 0$ for all $s \in (R \cup A)$ $s \neq 0$ for every $s \in W$	0
$nw < nt < nw + na$	$s = 0$ for all $s \in R$ $s \neq 0$ for all $s \in W$ exactly $nt - nw$ slots in A are non-zero	0
$nt = nw + na$	$s \neq 0$ for all $s \in W \cup A$ $s = 0$ for all $s \in R$	0
$nw + na < nt$ $< nw + na + nr$	$s \neq 0$ for all $s \in (W \cup A)$ exactly $nt - nw - na$ slots in R are non-zero	nr
$nt = nw + na + nr$	$s \neq 0$ for all $s \in (W \cup A \cup R)$	nr
$nt > nw + na + nr$	False (instance is unsatisfiable)	

6.2 Modelling the Remaining Soft Constraints

In this section we describe the remaining soft constraints used in the experimental section.

First of all we introduce some auxiliary variables, which are needed for the soft constraints.

- $startTime_s$: starting time of slot s
- $endTime_s$: end time of slot s
- $workingDuration_s$: work duration of slot s
- $weekWorkDuration_{ln}$: sum of work duration of the slots of line ln
- $freeWeekend_{ln}$: a Boolean indicating if the weekend of line ln is free
- $conseqWork_s$: number of consecutive working turns including slot s
- $conseqFree_s$: number of consecutive rest turns including slot s

We impose constraints that force these variables to take the appropriate values according to the variables in the decision model.

We note that the values of variables $startTime_s$, $endTime_s$, $workingDuration_s$ and $weekWorkDuration_{ln}$ are in minutes.

Next we describe nine rules which generate the soft constraints.

1. Any two successive turns where the first turn is an operational duty turn and the second one is an operational duty turn should have a rest period at least of $time$.

$$(\neg B_{s,0} \wedge \neg B_{next(s),0})\ Implies$$

$$(startTime_{next(s)} + 1440 - endTime_s \geq time) \quad (s \in S)$$

   where $next(s)$ is the slot that follows slot s within a link. The Monday slot of line l follows the Sunday slot of line $l-1$ and the Monday slot of line 1 follows the Sunday slot of the last line.
2. Any two successive turns where the first turn is an operational duty turn and the second one is an spare duty turn should have a rest period at least of $time$.

$$(\neg B_{s,0} \wedge I_{next(s)} \geq i \wedge I_{next(s)} \leq j)\ Implies$$

$$(startTime_{next(s)} + 1440 - endTime_s \geq time) \quad (s \in S)$$

   where $i..j$ is the range of spare turn indices of $next(s)$.
3. Any two successive turns where the first turn is a spare duty turn and the second one is an operational duty turn should have a rest period at least of $time$.

$$(I_s \geq i \wedge I_s \leq j \wedge \neg B_{next(s),0})\ Implies$$

$$(startTime_{next(s)} + 1440 - endTime_s \geq time) \quad (s \in S)$$

   where $i..j$ is the range of spare turn indices of s.

4. Any two successive turns where the first turn is a work turn and the second one is a work turn should have a rest period at least of *time*.

$$(I_s \geq i_s \wedge I_s \leq j_s \wedge I_{next(s)} \geq i_{next(s)} \wedge I_{next(s)} \leq j_{next(s)})\ Implies$$

$$(startTime_{next(s)} + 1440 - endTime_s \geq time) \quad (s \in S)$$

where $i_s..j_s$ and $i_{next(s)}..j_{next(s)}$ are the ranges of work turn indices of s and $next(s)$ respectively.

5. At least n weekends of rest are required in the link.

$$\sum_{ln \in 1..nlines} freeWeekend_{ln} \geq n$$

where *nlines* is the number of lines in the link.

6. In any line, the total number of duty hours should be at least *min* and at most *max*.

$$min \leq weekWorkDuration_{ln} \leq max \quad (ln \in 1..nLines)$$

7. Blocks of work turns should contain at least *min* but no more than *max* turns.

$$(conseqWork_s = min - 1)\ Implies\ (conseqWork_{next(s)} = min) \quad (s \in S)$$

$$(conseqWork_s = max)\ Implies\ (conseqWork_{next(s)} = 0) \quad (s \in S)$$

8. Each blocks of rest days should be least *min* but no more than *max* days long.

$$(conseqFree_s = min - 1)\ Implies\ (conseqFree_{next(s)} = min) \quad (s \in S)$$

$$(conseqFree_s = max)\ Implies\ (conseqFree_{next(s)} = 0) \quad (s \in S)$$

9. Allow 24:00 hours per rest day plus *time* rest for blocks of *min* to *max* rest days.

$$(conseqFree_s = val) \wedge (conseqFree_s = 0)\ Implies$$

$$startTime_{next(s)} + 1440 + 1440 * val - endTime_{previous(s,val+1)} \geq time$$

$$(s \in S, val \in min..max)$$

where $previous(s, n)$ is the n-th previous slot of s.

We want to remark that each rule generates a list of soft constraints. All the soft constraints generated from the same rule have the same penalty, which is an input parameter specified by the instance.

6.3 Solving Optimisation Problems with SMT

As the most successful method of solving the hard constraints is to use the Boolean model for SMT, we have chosen to extend that to the optimisation problem.

SMT is inherently for decision problems but there are various ways to put a wrapper around a solver to enable the solution of optimisation problems. We tried three different wrappers:

Binary First use Yices 1 to solve the decision problem. Compute the objective value, u, of that solution. Then use binary search over the range $0..u$ to find the smallest value, opt ($0 \leq opt \leq u$) such that the decision problem with the constraint that the objective does not exceed opt is satisfiable. This was implemented through the Yices 1 API, which enabled learned clauses to be retained through multiple runs of the binary search.

BDD We also tried the above approach except that the constraint that the objective does not exceed opt was compiled by encoding the constraint as a BDD and then translating that BDD to a Boolean constraint [5].

Core We treated the rostering problem as a weighted SMT problem, using the objective function to determine the weights. We solved the resulting weighted SMT problem with the unsatisfiable-core method of Ansotegui et al. [6], which we implemented as a wrapper around Yices 1.

6.4 Experimental Evaluation

First of all, in Figure 5 we present the time spent by the three SMT solving methods when solving instances B and D using the full decisional model plus the pattern constraint. We show that representing the pattern as hard constraints allows us to solve the model, in contrast to representing the pattern as soft constraints which cannot be solved in less than 600 seconds. Note that we only have one column when we represent the pattern as hard constraints because there is no optimization and, hence, it is the same time for all three methods.

	Instance D				Instance B			
	Pattern HC	Pattern SC			Pattern HC	Pattern SC		
		BDD	Binary	Core		BDD	Binary	Core
Total time	29.5	> 600	> 600	> 600	1.08	> 600	> 600	> 600
Solve time	2.87	> 600	> 600	> 600	0.026	> 600	> 600	> 600

Fig. 5. Solving and total times (in seconds) of the three SMT solving methods when representing the pattern as hard constraints (HC) and as soft constraints (SC) for instances B and D.

Finally, in Figure 6 we show the performance of the three SMT solving methods solving different models of instances A, B, C and D. Instances A and C are solved using all the rules of their respective optimization models, while instances B and D

are solved using only a subset of the rules of their respective optimization model. The patterns are all represented using hard constraints, and Rn refers to rule n defined in Section 6.2.

The table of results shows that for solving these instances the core-based method is the best solving method followed by the BDD. On one hand, we find the optimal solution of instances A and C in 2.2 and 5.3 seconds respectively. On the other hand, we find the optimal solution for a subset of rules of instances B and D, the pattern and three rules for the former and the pattern and one rule for the latter.

	BDD	Binary	Core
Instance A: R4+R5+R6+R7+R8+R9	2.06 (0.65)	3.35 (1.91)	1.64 (0.25)
Instance C: R1	5.29 (1.09)	5.21 (1.01)	4.52 (0.36)
Instance D: pattern	29.5 (2.87)		
Instance D: pattern+R1	213.46 (99.49)	> 600	119.42 (5.91)
Instance D: pattern+R1+R2	> 600	> 600	> 600
Instance B: pattern	1.08 (26 ms)		
Instance B: pattern+R1+R2+R3	4.98 (0.81)	9.85 (5.66)	4.43 (0.25)

Fig. 6. Total time and solving time (within parentheses) in seconds spent by the three solving methods on solving different models of instances A, B, C and D.

7 Conclusions and Future Directions

Our work has found that for small problem instances our approach can rapidly find optimal solutions. However, the difficulty of the instances grows rapidly as the number of lines or number of soft constraints increase. Our expectations were that SMT approaches would scale better than this and understanding this could prove useful to improving the performance. We have made some attempts to identify the difficult parts of the problem, but have not yet succeeded and we have some ideas for improving our models.

Future work includes understanding why our current approach is not performing better and trying to solve the problem with other solvers. We also have some ideas for improving our models. Finally, we hope that this paper could stimulate others to attack the problem.

The traincrew rostering problem may prove to be best suited for solution by a local search method and we think it is quite possible that further work could devise a local search method that significantly improves upon the method currently used by TRACSRoster.

References

1. Ansótegui, C., Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving weighted CSPs with meta-constraints by reformulation into satisfiability modulo theories. Constraints **18**(2) (2013) 236–268
2. Dutertre, B., de Moura, L.M.: The Yices SMT solver. Tool paper at http://yices.csl.sri.com/tool-paper.pdf (August 2006)
3. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3) (2012) 273–303
4. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A New Look at BDDs for Pseudo-Boolean Constraints. Journal of Artificial Intelligence Research (JAIR) **45** (2012) 443–480
5. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving intensional weighted CSPs by incremental optimization with BDDs. In O'Sullivan, B., ed.: Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings. Volume 8656 of Lecture Notes in Computer Science., Springer (2014) 207–223
6. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing. Volume 5584 of LNCS., Springer (2009) 427–440
7. Kroon, L., Huisman, D., Abbink, E., Fioole, P.J., Fischetti, M., Maróti, G., Schrijver, A., Steenbeek, A., Ybema, R.: The new Dutch timetable: The OR revolution. Interfaces **39**(1) (2009) 6–17