Air Traffic Controller Shift Scheduling by Reduction to CSP, SAT and SAT-related Problems*

Mirko Stojadinović

Faculty of Mathematics
University of Belgrade
mirkos@matf.bg.ac.rs

Abstract. [1] In this paper we present our experience in solving Air Traffic Controller Shift Scheduling Problem. We give a formal definition of this optimization problem and introduce three encodings. The encodings make possible to formulate a very wide set of different scheduling requirements. The problem is solved by using SAT, MaxSAT, PB, SMT, CSP and ILP solvers. In combination with these solvers, three different optimization techniques are presented, a basic technique and its two modifications. The modifications use local search to modify some parts of the initial solution. Results indicate that SAT-related approaches outperform other solving methods used and that one of the introduced techniques which uses local search can significantly outperform the basic technique. We have successfully used these approaches to make shift schedules for one air traffic control center.

1 Introduction

In the last few decades, personnel scheduling problems have been extensively studied (e.g., nurse scheduling problem [7], course timetabling [9]). Given the input parameters (e.g., the number of available workers, their skills and skills needed for working positions) and constraints (e.g., maximum number of consecutive working days for each worker), a schedule satisfying specified constraints needs to be generated.

In this paper we consider a type of scheduling problem called *Air Traffic Controller (ATCo) Shift Scheduling Problem* (*ATCoSSP*). The objective is to make a shift schedule, so that at each working hour every position is filled by a sufficient number of controllers with adequate skills. Because of the nature and the importance of their job, controllers need to be fully concentrated while they are on position. Therefore, their schedule must satisfy a stricter set of constraints compared to other personnel scheduling problems. To the best of our knowledge, there are no papers which describe encodings or instances of the problem, nor

* This work was partially supported by the Serbian Ministry of Science grant 174021.
[1] The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-10428-7_63

its solving methods in detail. We focus on solving this problem by using different exact methods: *CSP (COP)*, *SAT*, *Partial MaxSAT*, *SMT*, *ILP* and *PB*.

Constraint satisfaction problems (CSP) and *constraint optimization problems (COP)* [1] are wide classes of problems that include many problems relevant for real world applications (e.g., scheduling, timetabling, sequencing, routing, rostering, planning). Many different approaches for solving CSPs and COPs exist (e.g., constraint programming, mathematical programming, systematic search algorithms, forward checking, answer set programming). *Global constraints* [3] describe relations between a non-fixed number of variables and their purpose is to improve the readability and the efficiency of CSP solving.

Propositional satisfiability problem (SAT) [6] is the problem of deciding if there is a truth assignment under which a given propositional formula (in conjunctive normal form) evaluates to true. It is a canonical NP-complete problem [10] and it holds a central position in the field of computational complexity. *Partial MaxSAT problem* [6] is an optimization version of SAT which consists of finding an assignment that satisfies all *hard clauses* and maximizes the number of satisfied *soft clauses*. *Satisfiability modulo theories (SMT)* [6] is a research field concerned with the satisfiability of formulae with respect to some decidable background theory (or combination of them). Some of these theories are *Linear Integer Arithmetic*, *Integer Difference Logic*, *Linear Real Arithmetic*, etc. *Integer linear programming (ILP) problem* deals with minimizing a linear function while satisfying a set of linear constraints over integer variables. *Pseudo-boolean (PB) problem* [6] is a restriction of ILP problem where the domains of variables are restricted to $\{0, 1\}$. It can be considered as a generalization of SAT problem.

Contributions of this work are the following.

- A formal definition of *ATCoSSP* is given (to the best of our knowledge only informal descriptions are available in literature).
- Three encodings of the problem are introduced. The first two encodings formulate problem as a COP: the first encoding uses linear arithmetic constraints and the well-known global constraint *count*, whereas the second uses linear arithmetic constraints only. The third encoding formulates the problem as a PB problem.
- Different solving methods are compared and a variety of solvers are used. Experimental results indicate that, due to the nature of the constraints, non-SAT-related solvers cannot be on a par with SAT-related solvers when solving this problem is in question. Results show that solver Sugar [29] using reduction to SAT outperforms all other solvers. Sugar efficiently handles arithmetic constraints and can process large number of problem constraints.
- Three optimization techniques are defined and applied to *ATCoSSP*. In each of them we run the solver on instances that differ only in values of optimization variable. The first technique uses a variant of binary search to determine the next value of this variable. The second is intended to improve the solutions of the first approach by using local search specifically adapted to this problem. The third is even more adapted to this problem. It uses a two-step approach that can significantly speed up the solution process by overcoming

the main difficulty: a great number of variables and constraints. It finds an initial solution and iteratively searches for a better solution by fixing some parts of the initial solution and then optimizing its other parts.
– As stated by Burke et al. [7], the drawback of many developed scheduling algorithms is that they were not applied in practice. We have applied the solving methods described in this paper to make shift schedules for one air traffic control center. Instances of the problem are made available online.

Related work. The overview of available results related to *ATCoSSP* is presented by Arnvig et al. [2]. Some software tools for generating *ATCo* schedules already exist [16]. The advantages and the disadvantages of using these tools have already been recognized [31]. Some of them are in-house tools, and the details about them are not available. The others are more general tools (e.g., *Shift Scheduler Continuous*[2]) that can be used only with restricted versions of *ATCoSSP* (e.g., controllers are divided into teams and this is usually the case at big airports).

Scheduling problems have been extensively studied in literature. One of these problems is *Nurse Scheduling Problem* (NSP) [7]. Although there are many similarities between this problem and the problem we discuss, the important difference is that (in its most frequent form) NSP does not include scheduling on an hourly basis. *Course Timetabling* (CTT) [9] is well studied problem that does scheduling on an hourly basis. *ATCoSSP* differs from both cited problems as schedule requirements are much stricter. This implies a great number of hard and a small number of soft constraints, and finding any solution is often very difficult. Most often, known heuristic methods which successfully solve NSP and CTT assume easy way of finding initial solution. The described differences make it very hard to adapt known heuristic methods to *ATCoSSP*.

Overview of the paper. In Section 2 we give the problem description. In Section 3 we describe encodings of the problem and the optimization techniques used. We present our implementation and experimental evaluation in Section 4. Section 5 contains details about the real world application of our implementation. In Section 6 we draw some final conclusions and present ideas for further work.

2 Problem Description

ATCoSSP is a problem of assigning shifts to controllers in a considered period (usually a month or a year) with respect to some requirements. There are many documents that describe these requirements (e.g., [2], [16], [17]). The period consists of a number of days and the days consist of time slots. Each day a controller takes exactly one of three possible types of shifts. During *working shifts* a controller works in an ATCo facility on a given day from the first until the last time slot of that shift (including both these time slots) and rests in the remaining time slots. Working shifts may have different lengths and depending on the first time slot, we distinguish between morning, day, afternoon and night

[2] `http://www.bizpeponline.com/Helpssce.html`

shifts. It is assumed that the time slots of working shifts are known in advance. If a controller does not take working shift on some day, we say the controller takes a *rest shift* on that day (it is equivalent to weekend day for the majority of professions as teachers, lawyers, etc.). Each controller is allowed a number of paid vacation days and we say this person takes a *vacation shift* on these days. Vacations for the period are approved or disapproved in advance by the officials. For each controller, a number of *working hours* in the considered period must be greater than some value *min* (in order to get a full wage) and smaller than some value *max* (to avoid fatigue). Each working shift implies a number of working hours equal to the duration of that shift. A rest shift is not counted as working hours. A vacation shift implies some predetermined number of working hours.

Each controller must not take more than a specified number of consecutive working/rest shifts (usually 2 or 3). Only some controllers have the licence to be the heads of the working shift. On each day in each time slot when a facility works, at least one of these controllers has to be in the facility[3]. Each controller must take at least a minimum number of rest shifts per month. All controllers need rest between working shifts, and regulations usually specify a minimum number of rest time slots between working shifts (e.g., 12 hours).

Assigning controllers to positions within their working shifts is also a part of the problem. There are different types of positions in ATCo facilities (e.g., tower, terminal, en route) and depending on a facility size some or all of the positions are present. In any time slot of a working shift a controller can either be on position or can have a break. In any time slot a controller can be assigned to maximum one position. In two consecutive time slots a controller can be on two different positions. A controller must not be on position longer than the specified number m of consecutive time slots. Based on expected air traffic intensity, for each day in each time slot of working hours of a facility (some facilities work 24 hours a day while others do not) a number of controllers is needed for each position. A controller needs certain skills in order to obtain a licence to work on some position. It is assumed that the licences of controllers and the number of needed controllers for each time slot are known in advance[4].

The description so far has been focused only on *hard constraints* which are essential for shift schedule correctness and thus have to be satisfied. *Soft constraints* represent staff wishes (or preferences). Controllers may prefer different working shifts (e.g., they may prefer morning shifts), they may prefer to take consecutive working shifts as rarely as possible, etc. The reasons for including the staff to make schedules and some of most usual preferences are described by Arnvig et al. [2], subsection 6.5.

[3] This requirement can be expressed in terms of shifts instead of time slots.

[4] Actually, determining the number of controllers for each position is a problem itself, and it has been extensively studied in literature (e.g., [17], [31]).

3 Encodings of the Problem

Time slots can be of any fixed specified length but we assume the length of time slot is 1 hour. Shift schedules are generated for a period of one month and for each month, a new shift schedule needs to be generated. There are many reasons for this: expected monthly traffic intensity changes, different controllers take vacation days in different months, etc. We assume that there are only two types of positions: tower and terminal, and that there are no night shifts. These assumptions are not a limitation since the encodings can be easily extended to support more types of positions and night shifts.

Let us assume that the days are $1, \ldots, n_d$, the controllers are $1, \ldots, n_c$ and the shifts are $1, \ldots, n_s$. In order to make the encodings compact and more efficient, we assume that working shifts are $1, \ldots, n_s - 2$, that $res = n_s - 1$ is rest shift and $vac = n_s$ is vacation shift. Time slots take values $0, \ldots, 23$ and for each shift s, the first (s_f) and the last (s_l) working hour of that shift are fixed.

We experimented with different encodings and constraints. In the following 3 subsections, we describe 3 encodings which showed good results. Only the values of variables that determine controllers shifts for each day and controllers positions for each time slot are used when making tabular schedule for employees. Other variables are auxiliary and they are used to improve the readability and the efficiency of the encodings. The fact that vacation shifts and vacation working hours are fixed for the period is used to make the encodings more compact. Due to space limit, we omit the descriptions of some constraints[5]. In subsection 3.4 we describe how optimization instances are solved.

3.1 The first encoding

Linear arithmetic constraints and global constraint *count* [3] are imposed on integer variables. The *count* constraint requires that the number of occurrences of the value of the expression e in the set of expressions $e_1, \ldots, e_k$ is in some arithmetic relation ($=, \neq, \leq, <, \geq, >$) with the expression n. E.g., $count(\{x_1, x_2, x_3, x_4\}, 5) > 3$ (where $e = 5$, $e_i = x_i$, the relation is $>$, and $n = 3$) specifies that the value 5 occurs more than 3 times in the set of variables $\{x_1, x_2, x_3, x_4\}$.

Integer variables.

- $dc_{d,c}$: on the day d the controller[6] c can be assigned any from the possible shifts $1, \ldots, n_s$. Note that the fact that the working shifts are $1, \ldots, n_s - 2$ allows us to state that the controller c is working in a facility on the day d by imposing constraint $dc_{d,c} \leq n_s - 2$.

[5] The omitted variable relationships and the constraints of the encodings are available online from: `http://jason.matf.bg.ac.rs/~mirkos/Atco.html`

[6] Most of the constraints have to be true for all controllers, but we use some fixed controller c in the descriptions. Similarly for days, hours and shifts.

- $dhc_{d,h,c}$: in the hour h of the day d the controller c can be assigned different tasks: c can be on position on tower ($TOW = 0$) or terminal ($TER = 1$), c can have break hour in a facility ($B = 2$), vacation hour ($V = 3$) or a rest time ($R = 4$). Note that this allows us to state that c is having working hour in the facility in the hour h of the day d by imposing constraint $dhc_{d,h,c} \leq B$.
- $h_{d,c}$: on the day d the controller c is counted a certain number of working hours $(0, \ldots, 12)$.

Variable relationships.

- If the controller c takes the working shift s on the day d, then c works on working hours of that shift and rests during other hours of the day. So, for any $j \in \{s_f, \ldots, s_l\}$: $dc_{d,c} = s \rightarrow dhc_{d,j,c} \leq B$. For any $j \in \{0, \ldots, 23\} \setminus \{s_f, \ldots, s_l\}$: $dc_{d,c} = s \rightarrow dhc_{d,j,c} = R$.
- If the controller c takes the non-rest shift s with working hours $s_f, \ldots, s_l$ on the day d, then this implies c is working $s_l - s_f + 1$ hours on that day: $dc_{d,c} = s \rightarrow h_{d,c} = s_l - s_f + 1$. In case of the rest shift: $dc_{d,c} = res \rightarrow h_{d,c} = 0$.

Hard constraints.

- *Assigning shifts:* On the day d the controller c takes one of the possible shifts (already imposed as variable $dc_{d,c}$ takes one from the values $1, \ldots, n_s$).
- *Consecutive working shifts:* The controller c must take at least one rest shift in cws days in a row starting from the day d, if c does not take vacation shift on any of these days: $count(\{dc_{d,c}, \ldots, dc_{d+cws,c}\},\ res) \geq 1$.
- *Maximum working hours:* The controller c must not work more than the specified max working hours in a month: $\sum_{d=1}^{n_d} h_{d,c} \leq max$.
- *Positions filled:* If in the hour h of the day d at least k controllers are needed for tower position, then the following constraint is imposed: $count(\{dhc_{d,h,1}, \ldots, dhc_{d,h,n_c}\},\ TOW) \geq k$. Analogously for terminal position[7].
- *Consecutive time slots on position:* On the day d starting from the hour h the controller c is on position not more than the specified number m of hours in a row: $dhc_{d,h,c} \geq B \vee \ldots \vee dhc_{d,h+m,c} \geq B$.

Soft constraints. Each wish of a controller is expressed as a constraint that is true iff the wish is not satisfied. Each of these constraints is made equivalent to a fresh integer variable with the domain $\{0, 1\}$. If all these variables take value 0, then all wishes are satisfied. For the fixed controller c a new fresh variable is denoted by $x_{c,i}$, where index i takes the smallest unused non-negative number.

- *Shift preferences:* If the controller c prefers working shifts $s_1, \ldots, s_z$, then each shift s that is different from these shifts, rest or vacation shift is considered undesirable on any day d: $x_{c,i} \leftrightarrow dc_{d,c} = s$. E.g., if the month has 30 days and if the smallest unused non-negative index of the variables associated with the controller c is j, then for undesired shift s variables $x_{c,j}, \ldots, x_{c,j+29}$ are introduced.

[7] Actually, several *count* constraints are replaced by one *global cardinality* constraint [3] in order to obtain stronger filtering, but we omit the details.

– *Minimize consecutive working shifts:* The controller c prefers to take consecutive working shifts as rarely as possible. For each day d: $x_{c,i} \leftrightarrow dc_{d,c} \leq n_s - 2 \wedge dc_{d+1,c} \leq n_s - 2$.

3.2 The second encoding

As the syntax of some solvers does not allow the usage of global constraints, we adapt the first encoding not to use these constraints. *Integer variables*, *variable relationships* and *soft constraints* are the same as in the first encoding.

Hard constraints. As most of the hard constraints are the same as in the first encoding, we only describe differently encoded constraints. New variables and constraints specifying their relationships are introduced for if-then-else expressions, so this encoding can be of much greater size than the first one.

– *Consecutive working shifts:* The controller c must take at least one rest shift in cws days in a row starting from the day d, if c does not take vacation shift on any of these days: $dc_{d,c} = res \vee \ldots \vee dc_{d+cws,c} = res$.
– *Positions filled:* If in the hour h of the day d at least k controllers are needed for tower position, then the following constraint is imposed: $\sum_{c=1}^{n_c}(\text{if } (dhc_{d,h,c} = TOW) \text{ then 1 else 0}) \geq k$. Analogously for terminal position.

3.3 The third encoding

If $l_1, \ldots, l_n$ are Boolean literals, then the formula $l_1 + \ldots + l_n \ \# \ k, \quad k \in \mathbb{N}, \quad \# \in \{\leq, <, \geq, >, =\}$ is called *Boolean cardinality constraint* (BCC) [27]. In our presentation of the constraints we use equivalences, implications and clauses as often as possible in order to improve the readability of the paper, but the third encoding actually uses BCCs only. Each equivalence can be converted to 2 implications (from left to right and vice versa). The implication $a \rightarrow b$ can be directly translated to a clause $\neg a \vee b$ and more complicated implications can be translated to clauses by using De Morgans laws and distributivity rules[8]. Note that each clause $l_1 \vee \ldots \vee l_n$ is actually BCC $l_1 + \ldots + l_n \geq 1$.

Propositional variables.

– $dcs_{d,c,s}$: on the day d the controller c takes the shift s.
– $dc_{d,c}$: on the day d the controller c takes a working shift.
– $dhc_{d,h,c}$: the hour h of the day d for the controller c is a working hour (in a facility or on vacation). If false, c is having rest hour.
– $tow_{d,h,c}/ter_{d,h,c}/pos_{d,h,c}$: in the hour h of the day d the controller c is on position on tower/on position on terminal/on any position.
– $b_{d,h,c}/v_{d,h,c}$: in the hour h of the day d the controller c has break hour in a facility/has vacation working hour.

[8] There is no risk of exponential blow-up as implications in this encoding have small number of literals.

Variable relationships.

- If the controller c takes the working shift s on the day d, then c works on working hours of that shift and does not work during other hours of the day. So, for any $j \in \{s_f, \ldots, s_l\}$: $dcs_{d,c,s} \rightarrow dhc_{d,j,c}$. For any $j \in \{0, \ldots, 23\} \setminus \{s_f, \ldots, s_l\}$: $dcs_{d,c,s} \rightarrow \neg dhc_{d,j,c}$.
- The controller c works in the hour h of the day d iff c is on position on tower or on position on terminal or has break hour in a facility or has vacation working hour: $dhc_{d,h,c} \leftrightarrow tow_{d,h,c} \vee ter_{d,h,c} \vee b_{d,h,c} \vee v_{d,h,c}$.
- The controller c is on position in the hour h of the day d iff c is on position on tower or on terminal: $pos_{d,h,c} \leftrightarrow tow_{d,h,c} \vee ter_{d,h,c}$.

Hard constraints.

- *Assigning shifts:* On the day d the controller c takes one of the possible shifts (any of working shifts, rest or vacation shift): $dcs_{d,c,1} + \ldots + dcs_{d,c,n_s} = 1$.
- *Consecutive working shifts:* The controller c must not work in a facility more than cws days in a row starting from the day d, if c does not take vacation shift on any of these days: $\neg dc_{d,c} \vee \ldots \vee \neg dc_{d+cws,c}$.
- *Maximum working hours:* The controller c must not work more than the specified max working hours in a month: $\sum_{d=1}^{n_d} \sum_{h=0}^{23} dhc_{d,h,c} \leq max$.
- *Positions filled:* If in the hour h of the day d at least k controllers are needed for tower position, then the following constraint is imposed: $\sum_{c=1}^{n_c} tow_{d,h,c} \geq k$. Analogously for terminal position.
- *Consecutive time slots on position:* On the day d starting from the hour h the controller c is on position not more than the specified number m of hours in a row: $\neg pos_{d,h,c} \vee \ldots \vee \neg pos_{d,h+m,c}$.

Soft constraints. Fresh Boolean variables are introduced in the same way as integer variables with the domain $\{0, 1\}$ in the first encoding.

- *Shift preferences:* If the controller c prefers working shifts $s_1, \ldots, s_z$, then any other shift s that is different from these shifts, rest or vacation shift is considered undesirable on any day d: $x_{c,i} \leftrightarrow dcs_{d,c,s} = 1$.
- *Minimize consecutive working shifts:* The controller c prefers to take consecutive working shifts as rarely as possible. For each day d: $x_{c,i} \leftrightarrow dc_{d,c} \wedge dc_{d+1,c}$.

3.4 Search for optimum

Controllers indicate importance for their wishes and this is expressed by associating integer weight with each wish they have. In order to make the schedule fair, the weights are scaled so that for each controller the sum of weights of all wishes is equal to some fixed value. If the controller c specifies m_c wishes expressed by the Boolean variables $x_{c,i}$ (introduced in the description of soft constraints), and if associated weights are scaled to the values $w_{c,i}$, then *controllers penalty* is defined: $c_{penalty} = \sum_{i=1}^{m_c} w_{c,i} \cdot x_{c,i}$. For each controller c, the constraint of the form $c_{penalty} \leq cost$ is imposed (for the fixed value of some integer variable $cost$). The

goal is to find a minimum non-negative value for this variable (the maximum of all controllers penalties is to be minimized). Note that all $x_{c,i}$ have the domain $\{0, 1\}$, so the upper constraint can be encoded either as a linear expression (for the first two encodings) or BCC (for the third encoding).

Three optimization techniques are used. For all of them, instances for different values of *cost* (with bounds $cost_l$ and $cost_r$) are generated and solved by new runs of the associated solver. The solving process starts from the beginning for each new value of *cost* on the instance which differs from the previous instance only in this value. We are aware that there are approaches that can improve efficiency by using incremental solving [14, 30], but the improvements should not be significant due to results of conducted experiments (Subsection 4.2). Linear search, binary search and many other algorithms [6] can be used to find an optimal value of *cost* variable. In all three techniques we use (asymmetric) binary search algorithm combined with some additional techniques.

The first technique (*bsBasic*). In this approach pure (asymmetric) binary search is used. If a solution is found and the maximum controllers penalty is some value *sol* ($sol = \max_{c=1}^{n_c} c_{penalty}$, where $c_{penalty}$ is calculated for the found values of $x_{c,i}$), then $cost_r = sol - 1$. If no solution exists, then $cost_l = cost + 1$. Next instance considers the value $cost = cost_l + k \cdot (cost_r - cost_l)$. For $k = 1/2$ this is symmetric binary search, and for $k \geq 1/2$ the satisfiable instances are favored (they are usually easier). The search is ended and an optimum is found when $cost_l$ becomes greater than $cost_r$.

The second technique (*bsShExc*). Having found a solution with the maximum penalty *sol*, a local search is used in order to improve the solution (to reduce *sol*). The local search is based on shift exchanges. Two shifts can be exchanged between two controllers if they are of the same length and if the exchange does not violate any of the hard constraints. In the example schedule given in Table 1, if Alice and Charlie have licences for the same positions and Alice prefers working in the morning, then their working shifts on the day 4 are promising candidates for exchange. The shifts are exchanged whenever the greater of two controllers penalties is not increased. This assures that *sol* cannot be increased. In order to escape from local minimum, after a number of iterations a certain number of random shift exchanges is performed (thus maybe increasing *sol*). After a number of local search iterations, the binary search continues.

Name	Day 1	Day 2	Day 3	Day 4
Alice	2 (08-16)	4 (rest)	2 (08-16)	3 (12-20)
Bob	4 (rest)	3 (12-20)	3 (12-20)	4 (rest)
Charlie	1 (04-12)	4 (rest)	1 (04-12)	2 (08-16)
Dave	4 (rest)	3 (12-20)	5 (vac.)	5 (vac.)

Table 1. Small example of schedule for only 4 days (no position schedule presented). Shifts are 1 (04-12), 2 (08-16), 3 (12-20), 4 (rest), 5 (vacation).

The third technique (*bsNoPos*). The third and the second technique are similar. The difference is in the way the local search is performed. This technique assumes that all controllers have the licence to work on all positions. We aim to get much smaller encodings by replacing position requirements with working shift requirements. A number of controllers is needed for each position in each time slot, as noted in Section 3. In the initially found solution, a sufficient number of controllers is assigned to each position in each time slot and at the same time a number of controllers is assigned to each working shift on each day (e.g., on the day 2 in the example given in Table 1, 2 controllers are assigned to the shift 3 and no controllers are assigned to the shift 1).

The goal is to reduce the number of assigned controllers to each working shift and to get less filled shift schedule than the one initially found. Let us first assume that for some days d_1 and d_2 the same number of controllers is needed for each position in each time slot of these days (we assume this is the case with days 1 and 3 in the example). The second assumption is that for each working shift, the number of assigned controllers to that shift on day d_1 is less or equal than the one on day d_2. The third assumption is that for at least one working shift, the number of assigned controllers to that shift on day d_1 is strictly less than the one on day d_2 (days 1 and 3 fulfill both the second and the third requirement). If these three assumptions are met, then the number of needed controllers for each working shift of d_2 is made equal to the corresponding number of d_1, thus decreasing the number of needed controllers for working shifts of the day d_2 (no controller is needed for the shift 3 on day 3). The positions assignment for d_1 is copied to position assignment for d_2. We repeat this until there is not a day for which the number of needed controllers for any working shift can be reduced.

The search is now continued in the encodings where instead of positions variables and constraints, the constraints specifying the number of needed controllers for each working shift on each day are imposed (e.g., in the first encoding for day 2: $count(\{dc_{2,1}, \ldots, dc_{2,4}\},\ 3) \geq 2$). Each time slot of these shifts is already associated with a position. This significantly reduces the encodings size (we denote the original encoding as *complete* and this encoding as *reduced* encoding). When an optimum on the *reduced* encoding is found, it is not necessary the optimum of the initial problem. The binary search then continues on *complete* encoding.

4 Experimental Evaluation

All the tests were performed on a multiprocessor machine with AMD Opteron(tm) CPU 6168 on 1.9Ghz with 2GB of RAM per CPU, running Linux. Value $k = 4/5$ is used for optimization as it showed good results in initial experiments. The timeout is 600 minutes (10 hours) for each instance, including both encoding and solving time (the first is negligible in comparison to the second).

Instances. We experimented with making monthly shift schedules for 2 different months for an airport in Vršac, Serbia, that employs 12 controllers. Officials specified different input parameters as they wanted to choose among several

	Variables		Constraints		Domain	
Encoding	*complete*	*reduced*	*complete*	*reduced*	*complete*	*reduced*
1	10215	3951.7	81295.3	13400	4.79	2.79
2	35222.2	4467	131840	14430	2.82	2.70
3	55447.9	12964	160314	65293.1	2	2

Table 2. Average numbers of variables and constraints, and average domain size on all instances.

schedules. Parameters differed in number of working shifts (from 3 to 5) and their working hours (from 8 to 12), in number of controllers allowed to take vacation, and in few other parameters. For each month 9 instances were generated. Officials selected one of the solutions to be an official schedule[9]. In this way, 18 real-world instances were generated[10]. Additionally, we generated 4 harder instances in order to estimate the scalability of different solving methods. These instances include more controllers (17-25), longer periods (60-120 days) and more controllers are needed for certain positions in certain time slots.

Table 2 shows the average number of variables, constraints and average domain size for each of the encodings. The numbers are presented both for the *complete* and *reduced* encodings used during local search in *bsNoPos* technique. The numbers indicate that the size of *reduced* encoding is significantly smaller compared to *complete* encoding. This is due to a large number of variables and constraints directly connected to position requirements. Both these variables and constraints need to be introduced for each controller, day and time slot, so we advocate that the sizes of *complete* encodings can not be significantly reduced. Although the first two encodings differ only in the way some of the constraints are encoded, the first encoding is much more compact as it uses global constraints.

4.1 Solving methods and preliminary experiments

Solving methods. Four exact solving methods were used to make schedules.

The first method uses state-of-the-art non-SAT-oriented CSP solvers for solving generated CSP and COP instances in two formats. The first two encodings specifications can be directly translated to these formats. XCSP [25] format is used by solvers *Mistral* 1.545 [19] and *Abscon* 112v4 [23]. MiniZinc [24] input format is used by solvers mzn-g12cpx, mzn-g12fd, mzn-g12lazy, mzn-g12mip included in MiniZinc 1.6 distribution and mzn-gecode from Gecode-4.2.0 [26] distribution.

The second method reduces CSP and COP instances in the described formats to satisfiability problems SAT, Partial MaxSAT and SMT (in Linear Integer Arithmetic theory). In this approach, input instances are translated to instances

[9] The most important parameter in selection was the number of controllers allowed to go on vacation - the intention was to maximize this number.

[10] The source code of our implementation and the instances used in experiments (but without third-party solvers, due to specific licensing) are available online from: `http://jason.matf.bg.ac.rs/~mirkos/Atco.html`

of satisfiability problems in standardized input formats (e.g., DIMACS[11] for SAT, WCNF[12] for Partial MaxSAT, SMT-LIB[13] for SMT). Modern efficient satisfiability solvers are used for finding solutions that are then converted back to the solutions of the original CSPs and COPs. Systems Sugar v2-0-3 [29] and Sat4j 2.3.4 [4] are used for reduction to SAT. We performed the reduction to SMT and solvers Z3 v4.2 [11] and Yices 2.2 [12] are used for solving generated instances. Partial MaxSAT solvers QMaxSAT 0.4 [21] and MSUnCore-20130422 [22] are used for solving (slightly modified) instances generated by Sugar.

The third method solves the problem instances of the third encoding directly encoded in satisfiability input formats (PBS[14] for PB and already mentioned DIMACS format for SAT) by corresponding satisfiability solvers. We used SAT solvers *clasp* 2.1.3 [18], MiniSat 2.2 [13] and Lingeling aqw-27d9fd4-130429 [5], and PB solvers *clasp* 2.1.3 and MiniSat+ 1.0 [15]. In all further experiments in case of SAT and Partial MaxSAT solvers reduction to clauses is done using sequential counters (as they outperformed cardinality networks, that we used in experiments) implemented in system *meSAT* [28]. This system implements 5 encodings of CSP problems to SAT and it uses different encodings of BCCs to SAT. When number of variables is greater than 20, at-most-one constraint (special type of BCC where # is $\leq$ and k is 1) is encoded in a way described by Chen [8]. Otherwise, it is encoded in a way described by Klieber [20].

The fourth method uses ILP solver IBM ILOG CPLEX Optimization Studio[15] with the second encoding specification translated to its input format.

Preliminary experiments. These were conducted on 5 randomly selected instances with the goal to eliminate less efficient solvers. All solvers except Partial MaxSAT solvers were used with the described optimization techniques, as they outperformed the built-in optimization algorithms. All solvers were used in their default configurations.

4.2 Experimental results

In this subsection we present the results only for the solvers that achieved the best results in preliminary experiments and for the *interesting instances*, the ones for which not all of the best solvers found optimum within given timeout.

Comparison of techniques. In *bsShExc* approach we used 10^6 iterations. After each 10^4 iterations random shift exchanges were performed. However, there was no improvement in the value of *sol*, so we excluded this approach from the experimental results. Table 3 summarizes the results of comparison between the two remaining techniques using the best solvers. The average objective value achieved on interesting instances is given. The results show that in case of the best solvers (Sugar and Yices) *bsNoPos* technique outperforms *bsBasic* technique.

[11] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.dvi`
[12] `http://maxsat.ia.udl.cat/requirements`
[13] `http://www.smt-lib.org`
[14] `http://www.cril.univ-artois.fr/PB12/format.pdf`
[15] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

Encoding	1	2	2	3	3
Method	Sugar	Sugar	Yices	*clasp*	MiniSat
bsBasic	31.5	29.3	50.8	53.2	66.3
bsNoPos	23.1	22.5	33.2	72.7	73.7

Table 3. Comparison of techniques on interesting instances (timeout 600 minutes): cells contain average objective value achieved, the smaller the number, the better the result is; rows represent techniques; *clasp* is used for solving PB instances.

	Encoding	1	2			3	
Instance	Best	Sugar *bsNoPos*	Sugar *bsNoPos*	Yices *bsNoPos*	QMaxSAT built-in opt.	*clasp* *bsBasic*	MiniSat *bsBasic*
2	52	61 (1)	55 (1)	**52 (1)**	160 (0)	54 (51)	52 (35)
3	52	55 (3)	60 (1)	54 (8)	160 (0)	58 (0)	**52 (129)**
4	28	34 (8)	**28 (41)**	36 (413)	160 (0)	36 (99)	38 (538)
5	*6*	**6 (54)**	10 (455)	12 (457)	160 (0)	20 (95)	10 (587)
6	*20*	**20 (28)**	28 (272)	20 (77)	20 (45)	20 (70)	20 (342)
10	*16*	16 (137)	20 (6)	22 (15)	26 (11)	22 (2)	**16 (105)**
14	*0*	6 (78)	0 (36)	0 (37)	0 (223)	**0 (21)**	0 (447)
15	*4*	20 (587)	18 (125)	**4 (418)**	160 (0)	42 (109)	66 (577)
18	*12*	13 (88)	20 (18)	14 (414)	**12 (77)**	15 (242)	18 (411)
19	*8*	**8 (25)**	8 (40)	8 (329)	160 (0)	10 (522)	110 (563)
20	8	14 (220)	15 (67)	**8 (329)**	160 (0)	95 (109)	160 (0)
21	12	13 (573)	**12 (300)**	42 (145)	160 (0)	160 (0)	160 (0)
22	18	34 (89)	**18 (125)**	160 (0)	160 (0)	160 (0)	160 (0)
average	18.2	23.1	22.5	33.2	115.2	53.2	66.3

Table 4. Results on interesting instances (timeout 600 minutes): each cell contains objective value; the time needed to achieve this value is given in parenthesis; rows represent different instances; *clasp* stands for using *clasp* on PB instances.

Detailed results. Table 4 shows the results of the best solver/technique combinations. Instances 19-22 are the additionally generated harder instances. If the best achieved objective value from all the methods used (column *Best*) equals the optimum, the content of the cell is printed in italic font. For cells containing 2 numbers, the first number is the value of objective variable achieved by the method in the given timeout. Number 160 denotes that the solver did not find any solution on a given instance (160 is used as it is greater than all the objective values obtained). The number in parenthesis is the time (in minutes) needed to achieve the objective value. The cells which represent the best method for each instance are printed bold. In the last row we present the arithmetic mean (average objective value) from the obtained objective values.

Results show that among the best solvers there were no non-SAT-oriented solvers (mzn-g12cpx and *Abscon* were the only ones that managed to solve some of the instances but could not refine any of the found solutions). Interestingly, CPLEX could not solve any instance. By removing constraint *Positions filled*, CPLEX managed to solve some of the instances so probably these constraints are the reason for inefficiency. We attribute success of SAT-oriented solvers to small

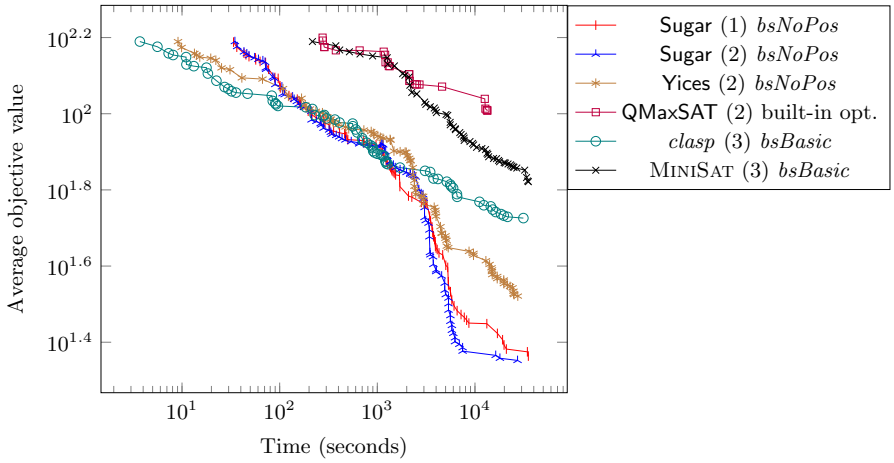

Fig. 1. Average objective value achieved in time - each mark on the curves represents one decrease in value; the encoding for each solver is given in parentheses.

domains of variables and large number of connection constraints (they represent about 75% of the generated constraints). The best results in average are achieved by reduction to SAT using Sugar. However, there are cases when other solvers achieved better performance. When we look at the hard instances (19-22), we can see that Sugar significantly outperforms other solvers. As BCC encodings of hard instances for the third encoding are very large, MiniSat and *clasp* are less successful with these instances. Sugar efficiently handles arithmetic constraints by using order encoding. This solver can process large number of constraints of the problem as it uses efficient built-in propagation. These are the reasons why it outperforms other solvers. Its success cannot be attributed to the underlying SAT solver, as MiniSat achieved only slightly better performance than Lingeling.

Figure 1 shows the change of average objective value achieved on the interesting instances during time (as if tests were run in parallel) for each of the solving methods presented in Table 4. It is taken that objective value achieved in the beginning is 160 for all solving methods and all instances. Solver Sugar shows the best performance and it achieves similar performance on the first and the second encoding. This can be attributed to similarity of these encodings, with the difference in encoding some of the constraints only. Reduction to PB achieves good results in the beginning and is the best solving method for finding quick solutions. However, it does not scale well in time. The average number of solver runs is 19 when an optimum was found, otherwise it is 3. In presentation of Sugar++ [30] (a version of Sugar using incremental solving) in Third International CSP Solver Competition[16], the authors indicate that the solving time of incremental search can be significantly increased when the number of solver runs is small (less than 6). For these reasons, we advocate that incremen-

[16] `http://www.cril.univ-artois.fr/CPAI08/`

tal search could not significantly improve the objective values found, although it could reduce the time in cases when optima were found.

5 Real World Applications

The schedules for the airport in Vršac were generated manually prior to using techniques described in this paper. The need for automated generation of schedules is seasonal. The number of needed working hours in summer months increases and the number of available controllers decreases, as this is the time when most of employees go on vacation. Therefore, it becomes too hard to generate schedules manually for these months, and the airport staff is trying to find the way to automate this process. Last year scheduling was offered as a service, not as a tool, as we have not yet developed GUI-based tool to enter input (but we plan to address this issue in our further work). We generated schedules for summer months of 2013. It took us about a month to develop the application that reads the input, automatically generates instances, solves them and outputs tabular schedules (their correctness is automatically checked). The automated generation of schedules is also planned for summer months of 2014.

We generated two schedules manually in order to compare them with the automatically generated ones. On average, we achieved objective value 50 in 3 hours by manually generating and objective value 28 in 11 minutes by automated generation (Sugar (2) *bsNoPos*). Many wishes in soft constraints became possible to satisfy by using the automation. This lead to higher satisfaction of the staff. Since automatically generated schedules reduce the overall controllers workload, the controllers are less subject to fatigue, so the overall safety is improved. Also, the manual scheduling was very error-prone. No improvement of solution could be made by using simple shift exchanges, as suggested by the results of *bsShExc* technique. The solution can be improved if we consider the exchange of a larger number of controllers and shifts of different lengths. But in our experience, this is almost impossible to achieve manually.

6 Conclusions and Further Work

In this paper, we have presented the air traffic controller shift scheduling problem. We have described three encodings of the problem in detail and presented three optimization techniques for solving the problem. A variety of solvers have been used for this problem. We have used the described solving methods to design shift schedules for one air traffic control center.

To our knowledge, the presented encodings are the most compact way to introduce position variables and constraints as they are needed for each controller, for each day and for each time slot in all three encodings. These variables and constraints represent the largest part of generated instances. Non-SAT-related approaches are inefficient in processing these instances. SAT-related approaches are significantly more efficient as they compactly encode the variables with small domains and directly encode connection constraints. Experimental results show

that the technique *bsNoPos* that fixes assigned positions in local search improves efficiency of the best solver (Sugar). Generally, main lessons learned from the use of CP are that different CP techniques should be tried and that it is beneficial to hybridize CP with other techniques (e.g., local search).

Our experience shows that stated requirements can be very diverse and can change over time. Existing software cannot express these requirements. The encodings we have developed offer a rich modeling language and a possibility to formulate a very diverse set of requirements.

We plan to address unexpected condition changes (e.g., sick leave) in future. Also, the promising directions for further work are improvements of *bsShExc* technique and the usage of other types of local search.

Acknowledgments We thank Filip Marić, Milan Banković, Mladen Nikolić and anonymous reviewers for valuable comments on the first versions of this manuscript.

References

1. Krzysztof R. Apt. *Principles of constraint programming.* Cambridge University Press, 2003.
2. M Arnvig, B Beermann, B Köper, M Maziul, U Mellett, C Niesing, and J Vogt. Managing shiftwork in european atm. *Literature Review. European Organisation for the safety of air navigation*, 2006.
3. Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalog. Technical report, SICS, 2005.
4. Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
5. Armin Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. *FMV Report Series Technical Report*, 10(1), 2010.
6. Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
7. Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. Scheduling*, 7(6):441–499, 2004.
8. Jingchao Chen. A new sat encoding of the at-most-one constraint. In *Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation*, 2010.
9. Marco Chiarandini, Mauro Birattari, Krzysztof Socha, and Olivia Rossi-Doria. An effective hybrid algorithm for university course timetabling. *J. Scheduling*, 9(5):403–432, 2006.
10. Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *STOC*, pages 151–158. ACM, 1971.
11. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
12. Bruno Dutertre and Leonardo De Moura. The yices smt solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, 2:2, 2006.
13. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
14. Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.
15. Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *JSAT*, 2(1-4):1–26, 2006.
16. EUROCONTROL. Shiftwork practices study - atm and related industries. *DAP/SAF-2006/56 Brussels : EUROCONTROL*, 2006.
17. Committee for a Review of the En Route Air Traffic Control Complexity and Workload Model. Air traffic controller staffing in the en route domain: A review of the federal aviation administration's task load model. 2010.
18. Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp* : A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

19. Emmanuel Hebrard. Mistral, a constraint satisfaction library. *Proceedings of the 3rd International CSP Solver Competition*, pages 31–39.
20. Will Klieber and Gihwon Kwon. Efficient cnf encoding for selecting 1 from n objects. In *Proc. International Workshop on Constraints in Formal Verification*, 2007.
21. Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012.
22. Joao Marques-Silva. The msuncore maxsat solver. *SAT 2009 competitive events booklet: preliminary version*, page 151, 2009.
23. Sylvain Merchez, Christophe Lecoutre, and Frédéric Boussemart. Abscon: A prototype to solve csps with abstraction. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 730–744. Springer, 2001.
24. Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.
25. Olivier Roussel and Christophe Lecoutre. Xml representation of constraint networks: Format xcsp 2.1. *CoRR*, abs/0902.2362, 2009.
26. Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode. *Software download and online material at the website: http://www. gecode. org*, 2006.
27. Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.
28. Mirko Stojadinović and Filip Marić. mesat: Multiple encodings of csp to sat. Constraints. 2014, doi: 10.1007/s10601-014-9165-7.
29. Naoyuki Tamura and Mutsunori Banbara. Sugar: A csp to sat translator based on order encoding. In *Proceedings of the third constraint solver competition*, pages 65–69, 2008.
30. Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. Sugar++: a sat-based max-csp/cop solver. *Proc. the Third International CSP Solver Competition*, pages 144–151, 2008.
31. EATCHIP Human Resources Team. Ats manpower planning in practice: Introduction to a qualitative and quantitative staffing methodology. *HUM.ET1.ST02.2000-REP-01 Brussels : EUROCONTROL*, 1998.