Hybrid of Hill Climbing and SAT Solving for Air Traffic Controller Shift Scheduling

Mirko Stojadinović
Faculty of Mathematics, University of Belgrade (Serbia)
mirkos@matf.bg.ac.rs

Contribution to the state of the art
DOI: **10.7251/JIT1502081S** UDC: **656.7.025**

Abstract: Modern computers solve many problems by using exact methods, heuristic methods and very often by using their combination. Air Traffic Controller Shift Scheduling Problem has been successfully solved by using SAT technology (reduction to logical formulas) and several models of the problem exist. We present a technique for solving this problem that is a combination of SAT solving and meta-heuristic method hill climbing, and consists of three phases. First, SAT solver is used to generate feasible solution. Then, the hill climbing is used to improve this solution, in terms of number of satisfied wishes of controllers. Finally, SAT solving is used to further improve the found solution by fixing some parts of the solution. Three phases are repeated until optimal solution is found. Usage of exact method (SAT solving) guarantees that the found solution is optimal; usage of meta-heuristic (hill climbing) increases the efficiency in finding good solutions. By using these essentially different ways of solving, we aim to use the best from both worlds. Results indicate that this hybrid technique outperforms previously most efficient developed techniques.

Keywords: controller shift schedule, reduction to SAT, hill climbing.

Introduction

The importance of automatic completion of many tasks by using contemporary computers instead of performing tasks manually has been stressed many times (e.g. [9]). Personnel scheduling problems have been extensively studied in the last few decades (e.g., nurse scheduling problem [3], course timetabling [4]). When solving these problems, a schedule needs to be generated satisfying specified constraints (e.g. a worker cannot work more than certain number of days in a row), taking into account some input parameters (e.g. total number of working days, number of workers).

Air Traffic Controller (ATCo) Shift Scheduling Problem (ATCoSSP) [11] consists of finding assignments of controllers to shifts and in each shift assignments to different positions, so that all the specified constraints are satisfied. Besides satisfying constraints, schedule should also take into account wishes of the controllers and the aim is to satisfy as many wishes as possible.

Constraint satisfaction problems (CSP) and *constraint optimization problems (COP)* [8] include large number of problems relevant for real world applications (e.g., scheduling, timetabling, sequencing, routing, rostering, planning). In this paper, we consider only finite linear versions of these problems. Problems are represented by domains of variables and constraints specifying the relations between these variables. CSPs are solved by finding assignments of values from the corresponding domains to variables, such that all constraints are satisfied. COPs are optimization version of CSPs and in case of solving COPs the goal is to find solution with optimal (minimal/maximal) value of some expression.

There are many different approaches for solving CSPs and COPs (e.g., constraint programming, mathematical programming, systematic search algorithms, forward checking, answer set programming). *Propositional satisfiability problem (SAT)* [2] examines if the variables of a given Boolean formula can be consistently replaced by the values *true* or *false* in such a way that the formula evaluates to true. This problem holds a central position in the field of computational complexity. One approach to solving CSP instances is by *reduction to SAT*. It consists of translating input formula to SAT (called *encoding to SAT*), running some of the free and efficient SAT solvers, and translating the solution, if it exists, to the solution of the original problem. Solving COP instances using reduction to SAT can be done by solving several CSP instances using reduction to SAT – one by one, CSP instances having fixed, different values of optimization expression are reduced to SAT.

Hill climbing [10] is a mathematical optimization technique which belongs to the family of local search techniques. It is an iterative algorithm that starts with some solution to a problem and tries to find better solution by changing some of its parts. If the considered change would produce a better solution, then the change is made to the solution. The changes are repeated until no further improvements can be made. The change to be made on the solution can be selected in different ways. *Stochastic hill climbing* selects change to be made at random. The problem with hill climbing method is that it can finish in local optimum and not in the global one. There are many ways of dealing with this problem (e.g. restarting with different solution if the local optimum is found).

Related work. A work introduced by Stojadinović [11] describes ATCoSSP in detail and introduces three encodings of the problem: in two of them ATCoSSP is specified as a CSP and in third as a SAT. Several methods for solving the problem were developed and a variety of different solvers were used for solving. The variant achieving the best results is the one where in each iteration problem specification is reduced to SAT, SAT instance is solved and then local optimum is found by fixing parts of the solution and replacing shift constraints with position constraints.

Contributions. In this paper we present a new hybrid technique for solving ATCoSSP . Each iteration tries to find better solution and consists of several phases. First, the reduction to SAT is used to find a solution. The solution is then improved by using stochastic hill climbing that exchanges shift assignments and then by using encoding to SAT, such that some parts of the solution are fixed. Our hybrid aims at finding good solutions quickly, so we focus on finding solutions by using shorter timeout compared to one used in the paper to which we compare. Results show that the proposed hybrid is more efficient in finding quick and good solutions.

Overview of the paper. In the next section we present the problem being solved. Then, we describe two existing solving techniques for this problem and our modification of the second one. Then, we present experimental evaluation, draw conclusions and present ideas for further work.

Problem Description

ATCoSSP is solved by 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., [1], [7], [5]). Some of the requirements are necessary and they are expressed by imposing so called *hard constraints* (these are essential for shift schedule correctness). *Soft constraints* represent staff wishes (or preferences).

Hard constraints

The period consists of a number of days and each day consists of time slots. A controller takes exactly one of three possible types of shifts on each day. In case of *working shifts*, a controller works in an ATCo facility on a given day from the first until the last time slot of that shift and rests in the remaining time slots of that day. Working shifts are of a different length and depending on the first time slot they are separated in morning, day, afternoon and night shifts. The time slots of working shifts are known in advance and therefore fixed prior to making the schedule. Between days in which controller takes working shifts, he has one or more rest days and we say the controller takes a *rest shift* on these days (they are equivalent to weekends for the majority of professions as teachers, lawyers, etc.). A number of paid vacation days is on the disposal to each controller and we say the controller takes a *vacation shift* on these days. Officials approve or disapprove vacations in advance. To get a full wage, controller must work certain number of working hours in the considered period, but should not exceed some maximum threshold value. In case of working shift, the number of *working hours* for controller on one day is equal to the duration of that shift. For each vacation shift, a number of predetermined working hours is counted for the controller. No working hours are counted for rest shifts.

It is not allowed that controller takes more than a specified number of consecutive working/rest shifts (usually 2 or 3). At each working hour, one controller needs to be in charge - this must be one of the controllers that have the license for this duty. Each controller must take at least a minimum number of rest shifts per month. Regulations usually specify a minimum number of rest time slots between working shifts.

When solving ATCoSSP, one also needs to assign controllers to positions within their working shifts. Several types of positions exist in ATCo facilities (e.g., tower, terminal, en route) and depending on a facility size the number of these positions is present. In any time slot of a working shift a controller can either be on position or can have a break. In one time slot a controller can be assigned to maximum one position. A controller can be on two different positions in two consecutive time slots. Some fixed number of consecutive time slots on position is permitted (e.g. 3 hours). For each day in each time slot of working hours of a facility intensity of air traffic is estimated and based on this estimation, it is decided how many controllers are needed for each position. A controller needs certain skills and passed exams in order to obtain a license to work on some position. The licenses of controllers and the number of needed controllers for each time slot are known in advance.

Soft constraints

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. [1].

Problem encoding

Three encodings of the problem were already described in detail [11], so we do not describe them in this paper. We use only the first encoding (it performed approximately the same as the second, but better than the third). The problem is represented by a COP instance consisting of linear arithmetic constraints and global constraints (these constraints describe relations between a non-fixed number of variables) and COP solvers can be used for solving. One way of solving COP instances is by reduction to SAT. Soft constraints are important for the technique introduced in this paper, so we describe them in the following paragraph.

Wishes of the controllers have different importance and each wish is associated with a weight, specifying its importance. Weights for each controller are scaled in order to make sums of weights for each controller equal to some value. The scaling is used to make fairer schedule (e.g. controllers having small number of wishes will have associated greater weights than the ones having greater number of wishes). Number m_c is the number of wishes of controller c, where each wish is associated with the Boolean variables $x_{c,i}$ and each of the corresponding weights is scaled to value $w_{c,i}$. *Controller's penalty* is defined as $c_{penalty} = \sum_{i=1}^{m_c} w_{c\,i} \cdot x_{c\,i}$. The goal is to find the minimum non-negative value of variable *cost*, such that for each controller c, the constraint $c_{penalty} \leq cost$ is satisfied (the maximum of all controllers' penalties is to be minimized).

Solving techniques

We used three optimization techniques; the first two already introduced in literature [11] and the new one. For all of them, instances for different values of variable *cost* (with bounds $cost_l$ and $cost_r$) are generated and for each value these instances are solved by new runs of the associated solver. For each new value of *cost*, new solving is performed on the instance that differs from the previous instance only in this value. In all techniques we use (asymmetric) binary search algorithm, in some cases combined with additional methods.

Asymmetric binary search. The pseudo code of this algorithm is presented in Figure 1. The algorithm gets as arguments the specification of the problem (e.g. number of workers, number of days, etc.) and the maximum value of variable *cost*. The bounds of the optimum are set and the binary search is started. Instance (in case of this paper the SAT instance) is generated such that the maximum controller's penalty is bounded by the value *cost*. After solving the instance, there are two cases. In the first case, a solution is found and the maximum controller's penalty is some value *best_opt* (*best_opt* = $\max_{c=1}^{n_c} c_{penalty}$ is calculated for the found values of $x_{c,i}$). Solution refining is tried (thus possibly further improving the value of *best_opt*), and then the value of the right bound is updated: $cost_r = best\_opt - 1$. In case there is no solution (instance is unsatisfiable), the value of left bound is updated: $cost_l = cost + 1$. In next iteration of loop, instance with the new value of cost is considered: $cost = cost_l + 4/5 \cdot (cost_r - cost_l)$. Number 4/5 indicates that the satisfiable instances are favored as they are usually easier (1/2 would be used in case of symmetric binary search). This number is used as good results were obtained by using it in the original paper [11]. The search is ended and an optimum is found when $cost_l$ becomes greater than $cost_r$.

```
bin_search (problem_spec, cost_max)
    cost_l = 0;
    cost_r = cost_max;
    while (cost_l <= cost_r)
        cost = cost_l + 4/5 * (cost_r - cost_l);
        instance = generate (problem_spec, cost);
        if (run_solver (instance, &best_opt, &solution) == SAT)
            refine_solution (instance, &best_opt, &solution);
            cost_r = best_opt - 1;
            best_solution = solution;
        else
            cost_l = best_opt + 1;
return best_solution;
```

Figure 1. Binary search for solving ATCoSSP

The techniques we present here differ in implementation of the function *refine_solution* and the rest of the search is the same in case of all three techniques.

Basic binary search

This technique does not do anything in the phase of refining solution. After finding one solution, it just continues with generating new instance and solving it.

Position avoiding

For each position, a number of controllers is needed in each time slot. In any found solution to the problem, controllers are assigned to shifts and within these shifts they are assigned to positions. A sufficient number of controllers to each position in each time slot is assigned. This technique aims to get smaller instances that are easier to solve by replacing position requirements with working shift requirements.

There are two phases in solution refinement: the first is used to make the solution schedule less empty and the second is used to find the better solution. We will describe these two phases by following the example given in Table 1.

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

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	4 (rest)	4 (rest)	1 (04-12)	2 (08-16)
Dave	4 (rest)	3 (12-20)	5 (vac.)	5 (vac.)
Ethan	1 (04-12)	4 (rest)	4 (rest)	1 (04-12)

Emptying the solution. In each iteration of this phase, two days d_1 and d_2 are found so that they fulfill three requirements. The first is that 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 $d_1 = 1$ and $d_2 = 3$ in the example). The second requirement 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 requirement 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. Different pairs of days are tried and when days are found satisfying three requirements, then position and shift assignments for d_1 are copied to assignments for d_2. If n controllers are assigned working shifts on day d_1 and m controllers are assigned working shifts on day d_2 ($n < m$), then n from m controllers on day d_2 are assigned working shifts and position assignments from day d_1 and the remaining $m - n$ controllers have rest day on d_2. In the example, let $d_1 = 1$ and $d_2 = 3$ be the days that satisfy three requirements. More controllers are assigned to work on day 3 than it is needed. One option of change may be: Bob is assigned shift 1 on day 3, Charlie is assigned rest shift on day 3 and other shift assignments on day 3 are left unchanged. This may open space for Charlie to work on some other day and maybe some of his previously unfulfilled wishes will be possible to satisfy. Note that some constraints may not be satisfied at these moments (e.g. maximum number of rest days in a row). We repeat this until no two days can be found fulfilling three requirements.

Reassigning shifts to controllers. Now, the changed schedule is used as a basis for generating a new one. Constraints specifying position assignments are not considered and only the constraints specifying the number of controllers for each day and each shift are considered. If enough controllers are assigned to each shift on each day, then the positions will be filled as each shift on each day is mapped to certain position assignment in previously found solution. The original encoding is called *complete* and the encoding avoiding position constraints is called *reduced encoding*. A separate asymmetric (inner) binary search is used on the problem now specified only using shift constraints. Optimum on the reduced encoding is not necessary the optimum of the initial problem. However, this optimum can reduce the upper bound of the outer binary search. When refining of the solution is complete, the outer binary search can continue finding better solution on complete encoding.

Position avoiding with hill climbing

Refining solution in case of this technique is done in two phases.

The first phase is stochastic hill climbing algorithm that uses restarting. The algorithm consists of iterative exchanging shifts between any two controllers. Two shifts can be exchanged if neither of them is vacation shift and if the exchange does not violate any of the hard constraints. For the purpose of checking if after exchange all constraints will be still satisfied, special testing software was developed and used. Exchange is actually performed if it does not increase the higher of the penalties of the two controllers involved in the exchange. This way, some number of exchanges (p) is tried. After specified number of trials is completed, the number of exchanges (q) is performed, even in case the higher of the penalties is increased - this enables the escape from a local minimum and continuation of the search from some solution potentially leading to global minimum.

The second phase is the technique 'Position avoiding technique', presented in previous section. It is used to potentially further improve the solution found prior to using the first phase, but starting with the upper bound that was potentially improved (decreased) by the first phase.

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. The timeout per instance was 60 minutes (1 hour), including both encoding and solving time.

Techniques used. In our experiments, we used three techniques that were described in previous section: Basic binary search (denoted *bs*), Position avoiding (*bsNoPos*) and Position avoiding with hill climbing (*hybrid HC*). When using hybrid hill climbing, after each 9000 iterations of exchanges aimed not to increase the value of greater of the considered sums of penalties, we conducted 50 exchanges possibly increasing greater of the sum of penalties of controllers considered for the exchange (as already stated, the aim is to escape from local minimum).

Instances. We used the existing set of 13 hard instances previously introduced [11] (they are called interesting instances and represent the subset not easily solved in the cited paper). This set consists of instances used for generating shift schedule for a small airport in Vršac, Serbia.

Experimental results

Table 2 summarizes the results of the experiments. The new technique (*hybrid HC*) is better than both previously developed techniques in case of solving 10 out of 13 instances. Average objective value also confirms that the new technique is more efficient than the other techniques. Note that the timeout used is smaller than in the cited paper (60 minutes comparing to the original 600 minutes). As already stated, the focus here is on faster search for the solutions.

Table 2. Results of three techniques (timeout 60 minutes): each cell contains objective value and the time needed to achieve this value is given in parenthesis; columns corresponding to different techniques and rows represent different instances

Instance index\	bsBasic	bsNoPos	hybrid HC
2	60 (3)	61 (1)	60 (1)
3	71 (2)	55 (3)	50 (2)
4	32 (30)	34 (7)	30 (5)
5	20 (54)	6 (49)	12 (26)
6	20 (26)	20 (28)	20 (15)
10	28 (12)	22 (5)	22 (4)
14	0 (58)	16 (37)	6 (41)
15	102 (28)	102 (27)	28 (28)
18	24 (15)	20 (13)	16 (30)
19	72 (50)	8 (28)	8 (17)
20	160 (0)	105 (58)	14 (60)
21	160 (0)	160 (0)	160 (0)
22	114 (35)	71 (55)	31 (37)
average	66.4	52.3	35.2

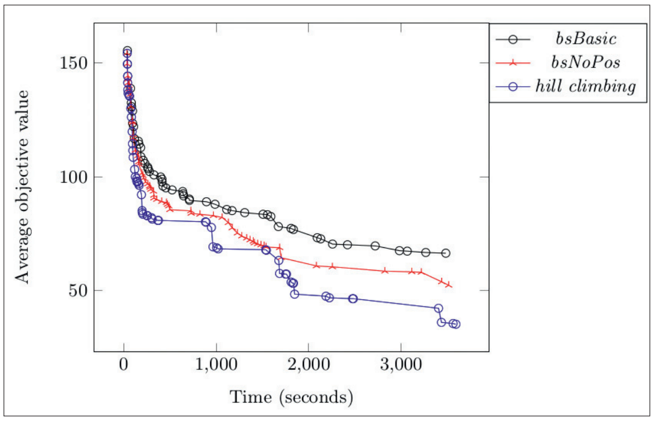

Figure 2. Average objective value achieved in time - each mark on the curves represents one decrease in value.

Figure 2 shows the decrease of the optimum achieved during time. Solutions are found quickly in the first couple of minutes in case of all three techniques (in this period, their solving process is mostly the same). After some time, the new technique manages to improve found solutions more quickly than the other techniques (its graph is closer to the horizontal axis). This means that this technique finds solutions fulfilling more preferences of the employees in shorter time. This trend continues until the timeout is reached.

Conclusions and Further Work

We have introduced technique for solving Air traffic controller shift scheduling problem that is the combination of the SAT solving and hill climbing. These two solving approaches are interleaved and the technique manages to find more quality solutions in less time, comparing to the best technique previously reported in literature.

In the future, we plan to implement more metaheuristic techniques for solving the problem. Comparison and investigating the influence of parameters of the techniques on the quality of the solution is another interesting direction.

References

[1] Arnvig, M., Beermann, B., Koper, B., Maziul, M., Mellett, U., Niesing, C., Vogt, J. (2006). Managing shiftwork in european atm. Literature Review. European Organisation for the safety of air navigation.

[2] Biere, A., Heule, M., van Maaren, H., Walsh, T. (2009). Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press.

[3] Burke, E.K., De Causmaecker, P., Berghe, G.V., Landeghem, H.V. (2004). The state of the art of nurse rostering. Journal of Scheduling, 7(6):441–499.

[4] Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. Journal of Scheduling, 9(5):403–432.

[5] Committee for a Review of the En Route Air Traffic Control Complexity and Workload Model (2010) Air traffic controller staffing in the en route domain: A review of the federal aviation administration's task load model.

[6] Cook, S.A. The complexity of theorem-proving procedures (1971). In STOC, pp 151–158.

[7] EUROCONTROL. (2006). Shiftwork practices study - atm and related industries. DAP/SAF-2006/56 Brussels : EUROCONTROL.

[8] Krzysztof, R.A. (2003). Principles of constraint programming. Cambridge University Press.

[9] Lev, K., Dovgobrod, M., Moiseevich, G. (2015). Control systems for automated vessel piloting through local stationary obstacle. Journal of Information Technology and Application (JITA), 5(1):61–64.

[10] Rossi, F., Beek, P.V., Walsh, T. (2006). Handbook of Constraint Programming. Elsevier.

[11] Stojadinović, M. (2014). Air traffic controller shift scheduling by reduction to csp, SAT and sat-related problems. In Principles and Practice of Constraint Programming- 20th International Conference, CP 2014, Lyon, France, pp 886–902.

Submitted: July 29, 2015.
Accepted: December 3, 2015.