A Constraint Programming Application for Rotating Workforce Scheduling

Markus Triska and Nysret Musliu

Database and Artificial Intelligence Group
Vienna University of Technology
{triska,musliu}@dbai.tuwien.ac.at

Abstract. We describe *CP-Rota*, a new constraint programming application for rotating workforce scheduling that is currently being developed at our institute to solve real-life problems from industry. It is intended to complement *FCS*, a previously developed application. The advantages of *CP-Rota* over *FCS* are a significantly smaller and more maintainable code base, portability across a range of different language implementations and a more declarative approach that makes extensions easier and mistakes less likely. Our benchmarks show that *CP-Rota* is already competitive with *FCS* and even outperforms it on several hard real-life instances from the literature.

Keywords: Staff Scheduling, Cyclic Schedule, Manpower Scheduling

1 Introduction

Computerized workforce scheduling has interested researchers for over 30 years. To solve rotating workforce scheduling problems, different approaches have been used in the literature, including exhaustive enumeration ([5], [2]), constraint (logic) programming, genetic algorithms ([8]) and local search methods.

In the present paper, we describe *CP-Rota*, a new constraint application for rotating workforce scheduling that is currently being developed at our institute to solve real-life problems from industry. It is intended to complement *FCS*, a previously developed application that is currently commercially used in many companies in Europe. *CP-Rota* builds upon, contributes to and improves previous constraint programming approaches to rotating workforce scheduling in the following ways:

– *CP-Rota* is written in portable Prolog and will eventually be released under a permissive licence to benefit both researchers and practitioners. Much of its code is already available on request at the time of publication.
– *CP-Rota* implements new allocation strategies (available as options for users to choose) that we discovered and discuss in this paper, which yield significantly improved performance on some real-life instances.
– Our benchmarks on real-life instances show the tremendous potential of constraint programming in rotating workforce scheduling, also and especially due to using different language implementations where they excel.

2 Related work

Many different approaches for solving rotating workforce instances are documented in the literature. Balakrishnan and Wong [1] solved a problem of rotating workforce scheduling by modeling it as a network flow problem. Laporte [6] considered developing the rotating workforce schedules by hand and showed how the constraints can be relaxed to get acceptable schedules. Musliu et al. [9] proposed and implemented a method for the generation of rotating workforce schedules, which is based on pruning the search space by involving the decision maker during the generation of partial solutions. The algorithms have been included in a commercial product called First Class Scheduler (FCS) [4], which is used by many companies in Europe. In [10], Musliu applied a min-conflicts heuristic in combination with tabu search. Although this yields good performance on many instances, the resulting search method is incomplete and its results are therefore not directly comparable with FCS. This paper also introduced 20 real-life problems collected from different areas in industry and the literature. [1]

The use of constraint logic programming for rotating workforce scheduling was first shown by Chan in [3]. Recently, Laporte and Pesant [7] have also proposed a constraint programming algorithm for solving rotating workforce scheduling problems, implemented in ILOG and requiring custom extensions.

3 The rotating workforce scheduling problem

With *CP-Rota* and in the present paper, we focus on a specific variant of a general workforce-scheduling problem, which we formally define in this section. The following definition is from [9] and proved to be able to satisfactorily handle a broad range of real-life scheduling instances in commercial settings. A rotating workforce scheduling *instance* as discussed in the present paper consists of:

- n: Number of employees.
- A: Set of m shifts (activities) : $a_1, a_2, \ldots, a_m$.
- w: Length of the schedule. A typical value is $w = 7$, to assign one shift type for each day of the week to each employee. The total length of a planning period is $n \times w$ due to the schedule's cyclicity as discussed below.
- R: Temporal requirements matrix, an $m \times w$-matrix where each element $R_{i,j}$ shows the required number of employees that need to be assigned shift type i during day j. The number o_j of day-off "shifts" for a specific day j is implicit in the requirements and can be computed as $o_j = n - \sum_{i=1}^{n} R_{i,j}$.
- Sequences of shifts not permitted to be assigned to employees. For example, one such sequence might be ND (Night Day): after working in the night shift, it is not allowed to work the next day in the day shift. A typical rotating workforce instance forbids several shift sequences, often due to legal reasons and safety concerns.

[1] Examples available from *http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/*

- MIN_s and MAX_s: Each element of these vectors shows, respectively, the required minimal and permitted maximal length of periods of consecutive shifts of the same type.
- MIN_w and MAX_w: Minimal and maximal length of blocks of consecutive work shifts. This constraint limits the number of consecutive days on which the employees can work without having a day off.

The task in rotating workforce scheduling is to construct a *cyclic schedule*, which we represent as an $n \times w$ matrix $S_{n,w} \in A \cup \{\text{day-off}\}$. Each element $S_{i,j}$ denotes the shift that employee i is assigned during day j, or whether the employee has time off. In a cyclic schedule, the schedule for one employee consists of a sequence of all rows of the matrix S.

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

In the present paper, we consider the generation of a single schedule that satisfies all the hard constraints given in the problem definition. Fulfilling all these constraints is usually sufficient in practice. The same constraints that we use in this paper are used in the commercial software *FCS* for generating rotating workforce schedules. This system has been used since 2000 in practice for many companies in Europe and the scheduling variant we discuss in this paper proved to be sufficient for a broad range of uses. However, *FCS* has several shortcomings that led us to consider constraint programming as an alternative approach. We discuss these shortcomings and their remedies with *CP-Rota* in the next section.

4 Shortcomings of FCS and their remedies in CP-Rota

Although *FCS* has been in commercial use since 2000 in several companies and proved to be an acceptable solution for many applications in practice, it has several disadvantages that hinder its further development:

- The code base of *FCS* has gotten quite large and hard to maintain. This makes user modifications difficult and error-prone. New scheduling ideas cannot be rapidly prototyped but require substantial development effort.
- *FCS* was implemented in *Visual Basic* and thus depends on essentially a single supported language implementation, which is in addition also not freely available. This complicates the sharing of code with other researchers and practitioners for joint development and turns every mistake in the language implementation into a potentially show-stopping problem.
- From [10], it is known that local search approaches – although incomplete – can significantly outperform *FCS* on some instances. Clearly, it is highly desirable to improve the running times of *FCS* to more closely match such competing approaches while retaining the completeness of the search.

When we started to work on the successor of *FCS* for the above reasons, we initially looked into constraint programming in the hope to significantly reduce the size of its code base. The promise of constraint programming was to just state the necessary requirements with high-level constraints and to then use built-in enumeration methods to search for solutions.

We implemented the successor using the portable constraint programming model described in Section 5 and named it *CP-Rota*. Eventually (see Section 7), *CP-Rota* even outperformed its predecessor on many instances, making constraint technology a potential remedy for all of the above original shortcomings.

5 A constraint formulation for rotating workforce scheduling

Our initial development environment was SWI-Prolog, which we chose due to its convenient libraries, tools and workflow, and also because it is freely available. We then ported the model to GNU Prolog due to its much better performance, and because it is also freely available. changes. When experimenting with custom allocation strategies (Section 6), GNU Prolog's lack of garbage collection hindered testing with larger instances, and we therefore ported the model also to B-Prolog, which is also a very efficient Prolog implementation and available free of charge for personal use. In all these systems, we model the rotating workforce problem as follows:

- The schedule is represented as a list of lists, and each element is a finite domain variable that denotes the shift type scheduled for this position.
- The temporal requirements are enforced via `global_cardinality/2` constraints on the columns of the schedule. In GNU Prolog, `fd_exactly/3` constraints are used instead.
- The minimal/maximal-length constraints on consecutive shifts of the same type are enforced via `automaton/3`.
- Reified constraints are used to map shifts of all types to either "work" or "day-off", and a second `automaton/3` constraint is used on these reified variables to limit the number of consecutive work and day-off shifts.
- Reified constraints are also used to express forbidden patterns. For example, if "0 4 3" is forbidden, the constraint is:

  $$X_k \;\#{=}\; 0 \;\#/\backslash\; X_{k+1} \;\#{=}\; 4 \;\#{==}{>}\; X_{k+2} \;\#\backslash{=}\; 3$$

  for all variables X_k, also taking into account the schedule's cyclicity. In B-Prolog, `notin/2` (negated extensional) constraints are used for better performance.

It only took a few days to implement this basic model (700 LOC, including a 50 LOC parser for instance specification files and 50 LOC for visualisations) and to get it to run on all of the above Prolog implementations. Only built-in constraints are used in all systems. In contrast, the development of *FCS* took several months.

6 Labeling and allocation strategies

The default strategy in *CP-Rota* is to first label the (reified) work/"day-off" Boolean variables. Then, all original variables of the schedule are labeled with the "first-fail" option. We call this strategy S_1. When S_1 did not yield a solution within 1000 seconds, we used Strategy S_2, which is to label all schedule variables from left to right, trying their values from lowest to highest. If this does not yield a solution within 1000 seconds, S_3 is used: Reified constraints are used to compute, for each column, the number of still missing shifts of each type. Processing the columns in order, we then choose the shift type that misses the *least* number of elements in that column, and assign it to a feasible variable with *smallest* domain. When S_3 also fails to find a solution within 1000 seconds, S_4 is used: It is similar to S_3, except the columns are not processed from left to right, but in descending order of their number of still missing shifts of any type.

7 Comparison with the commercial system FCS

Table 1 compares the performance of *CP-Rota* with that of *FCS* on 20 real-life instances from [10]. To the best of our knowledge, *FCS* is a state-of-the-art commercial system for generating rotating workforce schedules. We tested all instances on a Pentium 4, 1.8 GHZ, 512 MB RAM, using the latest versions of *FCS*, GNU Prolog (1.3.1) and SWI-Prolog (5.9.10). Except where stated otherwise, timing results are from GNU Prolog.

The table shows that *CP-Rota* nicely complements its predecessor so that 3 more instances than previously can now be solved. On 7 instances, *CP-Rota* outperforms *FCS* already with its default strategy (S_1), the converse holds for 6 instances.

8 Future work

Future improvements to *CP-Rota* include the addition of a more convenient user interface, real-time interaction with decision makers and the implementation of additional allocation strategies.

References

1. Nagraj Balakrishnan and Richard T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20:25–42, 1990.
2. B. Butler. Computerized manpower scheduling. Master's thesis, University of Alberta, Canada, 1978.
3. Weil G. Chan, P. Cyclical staff scheduling using constraint logic programming. *In: Burke, E, Erben W. (Eds.): PATAT 2000, Lecture Notes in Computer Science*, 2079:159–175, 2001.
4. Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Rota: A research project on algorithms for workforce scheduling and shift design optimisation. *Artificial Intelligence Communications*, 14(2):83–92, 2001.

Table 1. Comparison between *FCS* and *CP-Rota*

Ex.	n	*FCS* (time in sec)	*CP-Rota* (sec)	Strategy
1	9	0.9	**0.02**	S_1
2	9	0.4	**0.02**	S_1
3	17	1.9	**0.24**	S_1
4	13	1.7	**0.03**	S_1
5	11	3.5	**0.98**	S_1
6	7	2	**0.02**	S_1
7	29	16.1	**0.07**	S_2
8	16	**124**	964	S_1
9	47	>1000s	**19**	SWI, S_4
10	27	**9.5**	>1000s	–
11	30	**367**	>1000s	–
12	20	>1000s	>1000	–
13	24	>1000s	**114**	S_1
14	13	**0.54**	940	S_1
15	64	>1000s	>1000s	–
16	29	**2.44**	216	S_1
17	33	>1000s	**18**	SWI, S_3
18	53	**2.57**	>1000s	–
19	120	>1000s	>1000s	–
20	163	>1000s	>1000s	–

5. N. Heller, J. McEwen, and W. Stenzel. Computerized scheduling of police manpower. *St. Louis Police Department, St. Louis, MO*, 1973.
6. G. Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 1999.
7. G. Laporte and G. Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55/11:1208–1217, 2004.
8. Michael Mörz and Nysret Musliu. Genetic algorithm for rotating workforce scheduling. In *Proceedings of second IEEE International Conference on Computational Cybernetics (pages 121-126), Vienna, Austria*, 2004.
9. Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.
10. Nysret Musliu. Heuristic Methods for Automatic Rotating Workforce Scheduling. *International Journal of Computational Intelligence Research*, Volume 2, Issue 4, pp. 309-326, 2006.
11. Gilles Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. *CP 2004*, pp. 482-495, 2004.