Constraint-Based Rostering

Gilles Pesant

Abstract This short paper presents how rostering problems can be modeled and solved using constraint programming. The emphasis is on the choice of constraints which efficiently exploit the substructures of rostering problems, on the way to occasionally bend the rules in order to handle overconstrained instances, and on generic search heuristics built from the constraints in the model.

Keywords constraint programming · rostering

1 Introduction

Rostering problems appear both in the public and the private sectors. Despite a few decades of scientific investigation, providing a solution method that is robust across problems or even instances of the same problem remains a challenge. Typically in rostering the notion of optimality is not easily defined — in practice, human judgement is often required to make a choice from a set of candidate solutions meeting predefined criteria. Many therefore consider rostering as a satisfaction problem. Constraint programming (CP) is an method of choice to solve satisfaction problems and this paper intends to describe its *modus operandi*. Let us first review the rostering variants we shall consider.

1.1 Cyclical scheduling

In many industries and public services work is carried out on a continuous basis, twenty-four hours a day, seven days a week. Typical examples arise in police and fire departments, in automobile assembly plants, in steel mills, etc. In such contexts work schedules often comprise sequences of work shifts of several types separated by rest periods. When the personnel is interchangeable, cyclical schedules, a repeating pattern

Département de génie informatique et génie logiciel
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montreal, Canada H3C 3A7
E-mail: pesant@crt.umontreal.ca

Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	–	–	–	*D*	*D*	*D*	*D*
2	–	–	*E*	*E*	*E*	–	–
3	*D*	*D*	*D*	–	–	*E*	*E*
4	*E*	*E*	–	–	*N*	*N*	*N*
5	*N*	*N*	*N*	*N*	–	–	–

Fig. 1 A five-week rotating schedule

n°	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	-	-	-	-	-	-	-	-	D	D	-	-	A	B	H	H	-	B	-	B	A	-	D	-	B	-	-	-
2	-	E	E	-	-	-	-	-	-	-	-	-	-	-	-	C	H	H	H	-	-	-	E	-	D	-	-	-
3	-	F	-	B	-	-	-	-	C	-	B	-	B	A	-	F	-	E	-	-	-	-	F	-	A	-	-	H
4	-	C	-	A	E	E	E	-	-	-	F	F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	D	C	C	H	-	H	-	E	E	E	-	-	-	F	D	C	C	-	H	H	-	C	C	C	H	-	-
6	-	A	-	D	-	H	-	-	F	-	D	-	-	-	B	-	-	D	-	-	-	-	B	-	-	A	-	-
7	-	B	-	H	C	D	C	-	B	-	-	A	-	-	-	B	-	F	-	A	B	-	A	H	-	B	-	-
8	-	-	-	-	-	-	-	-	A	-	C	-	H	H	D	-	D	-	E	E	E	-	-	-	F	F	-	-
9	F	-	D	-	-	A	B	E	-	H	-	B	-	-	A	-	F	-	F	-	-	H	H	E	-	-	H	-
10	H	H	-	F	B	-	-	H	H	-	A	-	-	-	-	-	-	-	-	-	-	D	-	-	E	-	B	A
11	E	-	H	-	F	-	-	D	-	C	-	D	C	D	C	-	B	-	-	-	-	C	-	D	-	-	A	B
12	D	-	F	-	D	C	D	C	-	F	-	H	-	-	-	-	-	-	-	-	-	B	-	-	H	E	E	E
13	C	-	B	-	-	B	A	F	-	B	-	-	-	-	E	E	E	-	-	-	-	F	-	F	-	D	C	D
14	B	-	-	E	-	-	-	B	-	-	H	C	D	C	-	A	A	-	B	-	-	E	-	B	-	-	-	-
15	G	-	G	-	G	-	-	G	-	G	-	G	-	-	G	-	G	-	G	-	-	G	-	G	-	G	-	-
16	A	-	A	-	A	-	-	A	-	A	-	E	E	E	-	-	-	A	A	-	-	A	-	A	-	C	D	C
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	D	C	D	-	-	-	-	-	-	-
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	D	C	-	-	-	-	-	-	-

Fig. 2 A personalized four-week schedule for doctors

of sequences of work and rest days alternating over a few weeks, are particularly well adapted. In effect, everyone has an identical schedule but that is out of phase with the others, thus ensuring fairness among the staff.

A simple example will help understand the rules commonly applied to the construction of rotating schedules. Figure 1 depicts a five-week cycle containing three types of work shift (D = day, E = evening, N = night) and rest periods denoted by "–" . This schedule operates with five teams of workers. When it is first implemented, each team is assigned a different week. At the end of each week, each team moves down to the following row of the grid and the last team moves up to the first row. After five weeks, each team has gone through the same cycle. In this example, the required manpower is the same on each shift every day of the week, but this need not always be the case.

1.2 Personnel Scheduling

When members of the personnel have individual restrictions or preferences that must be taken into consideration, such as unavailabilities due to other activities or particular skills, cyclical schedules become inappropriate. Personalized schedules for each member of personnel are then preferred. The planning period generally ranges from a few weeks to a few months at a time. That latter family is typical for doctors and nurses. Figure 2 illustrates a typical schedule, each row corresponding to an individual roster.

1.3 Shift Scheduling

Whereas the previous two problems scheduled shifts over individual days, shift scheduling operates on a different scale. It builds the latter shifts by planning the individual activities during a work day, typically breaking it up into 15-minute intervals. This makes the planning period (in terms of the number of basic units to plan) slightly

longer than the previous two. Work regulations will require rest periods and lunch breaks depending on the duration and number of activities planned. This problem often arises in the service industry, for example in retail and banking.

In the rest of the paper, Section 2 gives a short introduction to constraint programming and its modeling primitives, Section 3 reviews the common requirements of rostering problems and shows how they are modeled in CP, Section 4 presents how we can allow a controlled violation of constraints, and Section 5 describes a promising generic search heuristic. Final comments are given in Section 6.

2 Constraint Programming

This section gives a very brief introduction to constraint programming. Constraint programming solves combinatorial problems by actively using the constraints of the problem to implicitly eliminate infeasible regions of the solution space. The algorithm at the heart of this approach implements complex logical reasoning over the set of constraints.

To every variable of a CP model is associated a finite domain: each value in that domain represents a possible value for the variable. Constraints on the variables forbid certain combinations of values. Picturing the model as a network whose vertices are the variables and whose (hyper)edges are the constraints provides insight into the basic algorithm used in CP. A vertex is labeled with the set of values in the domain of the corresponding variable and an edge is incident to those vertices representing the variables appearing in the associated constraint. Looking locally at a particular edge (constraint), the algorithm attempts to modify the label (reduce the domain) of the incident vertices (variables) by removing values which cannot be part of any solution because they would violate that individual constraint; this *local consistency* step can be performed efficiently. If every violating variable-value pair is identified and removed, we achieve *domain consistency* which is the best we can do locally; sometimes achieving that level of consistency is computationally too costly and we will only remove (numerical) values at both ends of a domain, achieving *bound consistency*.

The modification of a vertex's label triggers the inspection of all incident edges, which in turn may modify other labels. This recursive process stops when either all label modifications have been dealt with or the empty label is obtained, in which case no solution exists. The overall behavior is called *constraint propagation*.

Since constraint propagation may stop with indeterminate variables (i.e. whose domain still contains several values), the solution process requires search and its potentially exponential cost. It usually takes the form of a tree search in which branching corresponds to fixing a variable to a value in its domain, thus triggering more constraint propagation. We call *variable-selection heuristic* and *value-selection heuristic* the way one decides which variable to branch on and which value to try first, respectively. For combinatorial optimization problems, the tree search evolves into a branch-and-bound search in which one branches in the same way and lower bounds at tree nodes are obtained by various means.

2.1 Core of the Rostering Model

Starting this time with personnel scheduling, let J denote the index set for successive days of the planning period, I for staff members, and let Q denote the set of possible shifts, including off and vacation.

For each staff member $i \in I$ and each day $j \in J$, we define an *assignment variable* $\mathtt{X}_{ij} \in Q$ that indicates which shift is assigned to i on day j. For short, we use $\mathbf{X}_{i\star}$ (respectively $\mathbf{X}_{\star j}$) to represent successive assignment variables $\langle \mathtt{X}_{i1}, \mathtt{X}_{i2}, \ldots, \mathtt{X}_{i\ |J|} \rangle$ associated to staff member i (respectively $\langle \mathtt{X}_{1j}, \mathtt{X}_{2j}, \ldots, \mathtt{X}_{|I|j} \rangle$ associated to day j).

In cyclical scheduling, the same basic assignment model applies except there will be no index I of staff members since they share the same roster. In shift scheduling, index J ranges over short time intervals and set Q is made up of activities and breaks.

2.2 CP Modeling Primitives

In constraint programming just as in linear programming, one must define a formal mathematical model of the problem to solve. Unlike linear programming, constraint programming has a rich heterogeneous set of constraints in its modeling language with which one must familiarize himself. We present some of the most useful ones for rostering.

Linear constraints. Consider a vector $\mathbf{X} = \langle \mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n \rangle$ of integer finite-domain variables, a vector $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ of coefficients and integers ℓ and u. For linear constraint

$$\ell \leq \mathbf{cX} \leq u,$$

we can achieve bound consistency in time linear in n or the stronger domain consistency in time linear in n, in the domain sizes, and in the magnitude of ℓ, u, the c_i's, and the domain values, by computing a recursion through dynamic programming [1].

Functional constraints. These are useful to state a functional relationship between a pair of variables. Consider a vector $\mathbf{V} = \langle v_1, v_2, \ldots, v_m \rangle$ of values and finite domain variables $\mathtt{X}$ and $\mathtt{Y}$. Constraint

$$\mathtt{ELEMENT}(\mathtt{X}, \mathbf{V}, \mathtt{Y})$$

makes $\mathtt{Y}$ take the value from $\mathbf{V}$ which is indexed by $\mathtt{X}$. An equivalent but more direct syntax is

$$\mathtt{Y} = \mathbf{V}[\mathtt{X}].$$

Any modification to the domain of $\mathtt{X}$ is reflected on the domain of $\mathtt{Y}$ and conversely any change on $\mathtt{Y}$ can affect $\mathtt{X}$. Domain consistency is cheaply enforced [2].

Extensional constraints. Given a vector $\mathbf{X} = \langle \mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n \rangle$ of finite domain variables, constraint

$$\mathtt{EXTENSION}(\mathbf{X},\ \mathcal{T})$$

defines in extension the set $\mathcal{T}$ of admissible n-tuples for $\mathbf{X}$. The filtering algorithm that maintains domain consistency is exponential in n though for some special cases it can be low polynomial [3].

Value occurrence constraints. Consider vector $\mathbf{V} = \langle v_1, v_2, \ldots, v_m \rangle$ of values, vector $\mathbf{X} = \langle \mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n \rangle$ of finite domain variables taking their values in $\mathbf{V}$, and vector $\mathbf{C} = \langle \mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_m \rangle$ of nonnegative integer variables. Constraint

$$\mathtt{GCC}(\mathbf{C},\ \mathbf{V},\ \mathbf{X})$$

guarantees that each $\mathtt{C}_j$ equals the number of variables in $\mathbf{X}$ whose value is v_j. Its filtering algorithm, based on network flow theory, achieves domain consistency and runs in polynomial time [4].

Additionally, consider a positive integer w and two vectors of m nonnegative integers $\underline{\lambda}$ and $\overline{\lambda}$. Constraint

$$\mathtt{SLIDING\_GCC}(\mathbf{X}, \mathbf{V}, \underline{\lambda}, \overline{\lambda}, w)$$

guarantees that in each subsequence of $\mathbf{X}$ of length w, value v_k $(1 \leq k \leq m)$ appears between $\underline{\lambda}_k$ and $\overline{\lambda}_k$ times. This constraint is conceptually equivalent to `GCC` constraints expressed on each position of a sliding window but treating them all at once improves the filtering capability [5][6].

Value distribution constraints. Given a set of finite-domain variables $\mathbf{X} = \{\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n\}$ and bounded-domain continuous variables μ and σ, constraint

$$\mathtt{SPREAD}(\mathbf{X}, \mu, \sigma)$$

states that the collection of values taken by the variables of $\mathbf{X}$ exhibits an arithmetic mean μ and a standard deviation σ. Bound consistency is achieved in low polynomial time [7] [8].

Formal language membership constraints. A deterministic finite automaton (DFA) may be described by a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string one symbol at a time, applying the transition function δ at each step to update the current state. The string is accepted if and only if the last state reached belongs to the set of final states F. The languages recognized by DFA's are called regular languages.

Let $\mathbf{X} = \{\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n\}$ denote a vector of finite-domain variables with respective domains D_1, D_2, ..., $D_n \subseteq \Sigma$. Under a *regular language membership constraint*

$$\mathtt{REGULAR}(\mathbf{X}, \mathcal{A}),$$

any sequence of values taken by the variables of $\mathbf{X}$ must belong to the regular language recognized by $\mathcal{A}$. Domain consistency is achieved in time linear in the size of a layered graph built by unfolding the automaton over the variables [9]. A constraint based on the larger class of context-free languages was subsequently proposed [10][11].

3 Common Requirements

While the precise requirements for an adequate solution vary between problems and even among instances of a same problem, we attempt here a broad classification of the typical requirements, based on the relevant literature and on our own experience [12] [13] [14] [15]. More importantly, we will show how each class is modeled in constraint programming.

3.1 Coverage

Perhaps the most basic requirement is that a sufficient number and variety of shifts must be staffed throughout the planning period in order to guarantee minimum coverage of a demand curve. Sometimes there may be a target and/or a maximum allowed coverage. In the case of shift scheduling, the demand is expressed at the finer level of activities which are performed during a shift. The demand may also be expressed indirectly in terms of periods of the day instead of shifts. Such periods form a partition of the day into disjoint intervals of time. A staff member is present during a particular period if and only if his shift covers the considered period (shifts are usually designed so that they entirely cover one or several periods).

Consider the following example featuring such an indirection. There are five shifts: q_1 from midnight to 8am, q_2 from 8am to 4pm, q_3 from 8am to 8pm, q_4 from 4pm to midnight, and q_5 (off). The day is partitioned according to four time periods p_1 =[midnight, 8am], p_2 =[8am, 4pm], p_3 =[4pm, 8pm] and p_4 =[8pm, midnight]. Note that p_1 is covered by shift q_1, p_2 by q_2 and q_3, p_3 by q_3, and q_4, and p_4 by q_4. Consider vectors $\underline{q} = (3, 5, 6, 3)$ and $\overline{q} = (6, 8, 9, 6)$ indicating the minimum and maximum workforce allowed for each period ($\underline{q}_\ell \leq$ workforce at $p_\ell \leq \overline{q}_\ell$) on a given day j. In order to express the constraint, we first introduce a vector $\mathbf{M} = \langle \mathtt{M}_1, \ldots, \mathtt{M}_5 \rangle$ of auxiliary variables to represent the number of occurrences of each shift during day j. Then, we state

$$\mathtt{GCC}(\mathbf{M},\ Q,\ \mathbf{X}_{\star j}),$$

$$\mathtt{P}_1 = \mathtt{M}_1,\ \ \mathtt{P}_2 = \mathtt{M}_2 + \mathtt{M}_3,\ \ \mathtt{P}_3 = \mathtt{M}_3 + \mathtt{M}_4,\ \ \mathtt{P}_4 = \mathtt{M}_4,$$

$$3 \leq \mathtt{P}_1 \leq 6,\ \ 5 \leq \mathtt{P}_2 \leq 8,\ \ 6 \leq \mathtt{P}_3 \leq 9,\ \ 3 \leq \mathtt{P}_4 \leq 6.$$

The first line links the occurrence variables to the main decision variables through the global cardinality constraint described in Section 2.2. The second line builds simple linear expressions of the occurrence variables to represent the number of occurrences $\mathtt{P}_k$ for each period p_k. Finally the third line expresses the coverage requirements for periods.

3.2 Availability

A given staff member, according to his qualifications, full/part time status, vacation, and outside responsibilities, is not available at all times. Some activities are not possible at certain times of the day. There may be *preassignments*, *forbidden assignments*, and even *candidate vacation days* from which a certain number must be selected. Most availability constraints are easily modeled as unary constraints ($\mathtt{X}_{ij} = q$ or $\mathtt{X}_{ij} \neq q$).

3.3 Workload

The number of hours worked by a staff member (or some other measure of his workload) in the course of a week, two weeks, a month, or the whole planning period are regulated by the work contract, sometimes with very little flexibility such as for cyclical scheduling or in the case of nurses.

A workload constraint is defined by a 5-tuple $(i, j_{\text{beg}}, j_{\text{end}}, \underline{h}, \overline{h})$ and imposes that the number of hours worked by staff member i over the time period (set of successive days) $[j_{\text{beg}}, j_{\text{beg}+1} \dots j_{\text{end}}]$ lies between $\underline{h}$ and $\overline{h}$.

Consider the following example where there are seven shifts in Q: off (that lasts 0 hours); D4 and E4 (4 hours); D6 (6 hours); D8, N and E8 (8 hours). We consider the workload requirement $(i, 15, 21, 30, 35)$ that requires staff member i to work between 30 and 35 hours over the third week of the planning period (from day 15 to day 21). Let $h(\mathtt{X})$ represents the duration of the shift assigned to $\mathtt{X}$. A straightforward way to express the restriction is to state

$$\mathtt{Y}_j = h(\mathtt{X}_{ij}) \quad (15 \leq j \leq 21)$$

$$30 \leq \mathtt{Y}_{15} + \mathtt{Y}_{16} + \dots + \mathtt{Y}_{21} \leq 35,$$

using both linear and functional constraints.

In this case, simple bound consistency would normally be applied, unless the number of variables and the permitted range of total hours worked are small enough to make the dynamic programming approach feasible. There is an alternate way to express the constraint that will achieve domain consistency. Its cost is exponential in the number of distinct shift durations but this is always small in rostering problems. Let $\mathcal{Q} = \{Q_1, \dots, Q_p\}$ be the partition of Q into classes of shifts having the same duration and $h_1, \dots, h_p$ be the durations of shifts in $Q_1, \dots, Q_p$. In our example, there are four classes and the possible durations of the shifts are 0, 4, 6, and 8. Let $\mathcal{T}$ represent the set of tuples $(m_1 \dots m_p)$ such that $\underline{h} \leq \sum_{1 \leq k \leq p} h_k m_k \leq \overline{h}$ and $\sum_{1 \leq k \leq p} m_k = j_{\text{end}} - j_{\text{beg}} + 1$. For the example, $\mathcal{T} = \{(m_1, m_2, m_3, m_4) : 30 \leq 0 * m_1 + 4 * m_2 + 6 * m_3 + 8 * m_4 \leq 35$ and $m_1 + m_2 + m_3 + m_4 = 7\} = \{(3, 0, 0, 4), (3, 0, 1, 3), \dots\}$. We introduce auxiliary variables $\mathbf{Y} = \langle \mathtt{Y}_1, \dots, \mathtt{Y}_{j_{\text{end}} - j_{\text{beg}} + 1} \rangle$ with $\mathtt{Y}_k = t_{\mathcal{Q}}(\mathtt{X}_{ij_{\text{beg}}+k-1})$ $(1 \leq k \leq j_{\text{end}} - j_{\text{beg}} + 1)$, table $t_{\mathcal{Q}}$ giving the index of the duration class of the shift assigned to the decision variable, and multiplicity variables $\mathbf{M} = \langle \mathtt{M}_1, \dots, \mathtt{M}_p \rangle$, with $\mathtt{M}_k$ $(1 \leq k \leq p)$ counting the number of variables in $\{X_{ij_{\text{beg}}}, \dots, X_{ij_{\text{end}}}\}$ whose assigned value belongs to class Q_k. To express the constraint, we state

$$\mathtt{GCC}(\mathbf{M},\ \langle 1, \dots, p \rangle,\ \mathbf{Y}), \quad \mathtt{EXTENSION}(\mathbf{M},\ \mathcal{T}).$$

In cyclical scheduling, workloads are often expressed not on calendar weeks but on a sliding window of a given number of days. For example on any nine consecutive days, five or six must be worked:

$$\mathtt{SLIDING\_GCC}(\mathbf{X}, \langle \mathsf{off}, \mathsf{day}, \mathsf{evening}, \mathsf{night} \rangle, \langle 3, 1, 1, 1 \rangle, \langle 4, 9, 9, 9 \rangle, 9).$$

3.4 Distribution

In rostering, many requirements aim for a fair distribution of shifts among staff members and for balanced individual schedules. We distinguish *balance for a certain type of shift or for some other feature among the staff*, either evenly or according to some criterion such as seniority, *distribution of weekends off* across the planning period for individual staff members, and *relative proportion of certain types of shifts* in individual schedules. For the most part, this is not an issue for cyclical schedules but it can make quite a difference in the quality of personnel schedules.

Consider the following example. A weekend on which one day is worked and the other not is called a *broken weekend*, a generally undesirable feature. Let $\mathtt{B}_i$ represent the number of broken weekends in staff member i's schedule (these can be linked to the decision variables in a similar way as before). We often do not know in advance how many such weekends will occur in a schedule but we nevertheless wish the number of broken weekends to be about evenly distributed among all staff members except the first two, who have more seniority and should therefore have about half that of the others:

$$\mathtt{Y}_1 = 2\mathtt{B}_1,\ \mathtt{Y}_2 = 2\mathtt{B}_2,\ \mathtt{Y}_i = \mathtt{B}_i\ (3 \leq i \leq |I|), \quad \mathtt{SPREAD}(\mathbf{Y}, \mu, \sigma), \quad \sigma < 1$$

3.5 Ergonomic considerations

This is the largest and most heterogeneous class. Various requirements ensure a certain level of quality for the schedules produced and may be specified either globally for the staff or only for certain individuals.

Typical requirements are: *patterns of shifts over certain days* such as alternating between two types of shifts on weekends, *length of stretches of shifts of identical type* to avoid working too few or too many days in a row on a certain shift, *patterns of stretches* such as forward rotation (going from day shifts to evening shifts to night shifts to day shifts again), and *patterns of stretches of a given length* that ask for at least so many consecutive shifts of a certain type right after shifts of another type or for (lunch) breaks after so many hours worked in shift scheduling. We also include *preferences and aversions*.

One way to handle the latter is to consider them as the previously discussed availabilities, though this may be too strict and lead to overconstrained instances. We discuss another way in Section 4. As for the rest, they are all restrictions on the combinations of values taken by a sequence of variables and each can be modeled as a regular language. In shift scheduling, because the sequencing rules for activities and breaks are very constraining, all such rules are usually handled together in a single regular language.

For example in cyclical scheduling, most authors assume that a change of shift type can only occur after a rest period. If in addition only forward rotations are allowed, the only possible patterns in a three-shift system (a, b and c) are then aoa, bob, coc, aob, boc and coa ("o" indicates an off shift). The corresponding regular language can be described by the automaton $\mathcal{A}$ illustrated at Figure 3, giving the following constraint for each staff member i:

$$\mathtt{REGULAR}(\mathbf{X}_{i\star}, \mathcal{A}).$$

In some cyclical scheduling instances, *vertical constraints* on stretches of shifts are also imposed, e.g., no more than three Fridays in a row may be worked in the evening. This can be handled in a similar way as before, constraining a column of decision variables instead of a row.

4 Soft Constraints

Constraint programming's main strength is the reduction of the search space through powerful inference at level of individual constraints. For this to work, the underlying

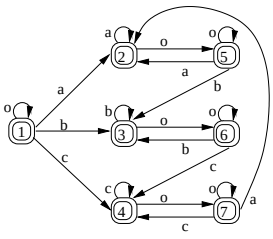

Fig. 3 An automaton corresponding to a forward rotation requirement.

assumption is that constraints are hard, i.e. cannot be violated. In rostering, requirements regarding coverage, availability, and even workload are usually hard. However some distribution and ergonomic rules may be soft. Staff members may also have conflicting preferences, leading to an overconstrained instance. We outline how CP can soften constraints while still performing inference on them.

The idea is to augment each soft constraint with a "cost" variable $\mathtt{Z}$ which represents the amount of violation of the constraint under the corresponding assignment of its variables [16]. The newly defined problem is not overconstrained anymore. For example, a binary constraint $\gamma(\mathtt{X}, \mathtt{Y})$ will be augmented to $\gamma(\mathtt{X}, \mathtt{Y}, \mathtt{Z})$. If $(\mathtt{X} = 1, \mathtt{Y} = 3)$ is a solution of γ, the corresponding tuple will be $(\mathtt{X} = 1, \mathtt{Y} = 3, \mathtt{Z} = 0)$; if $(\mathtt{X} = 1, \mathtt{Y} = 2)$ isn't, the corresponding tuple could be $(\mathtt{X} = 1, \mathtt{Y} = 2, \mathtt{Z} = 1)$ or some other positive value for $\mathtt{Z}$, depending on the violation measure.

If we ask to minimize the (weighted) sum of violation costs or simply bound the violation cost of each soft constraint, we can solve the problem with a traditional constraint programming solver. The important feature of this approach is that inference can still be applied, this time based on cost. For example, several of the most common constraints admit an inference algorithm that can be formulated as a network flow algorithm. In that case, *violation arcs* are added to the network, each with a cost corresponding to some violation measure of the constraint. The modified inference algorithm, instead of looking for a feasible flow, will compute minimum cost flows [17].

5 Constraint-Centered Search Heuristics

One of the main criticisms of constraint programming is that it is often necessary to design customized search heuristics in order to solve one's problem efficiently and robustly. Some generic search heuristics are available but they may not be robust enough for the given problem and suffer from occasionally long runtimes. The design of generic robust search heuristics has recently caught the attention of the research community — we present one such approach [18].

Constraints have played a central role in constraint programming because they capture key substructures of a problem and efficiently exploit them to boost inference. We can do the same thing for search with *constraint-centered heuristics* which guide the exploration of the search space toward areas that are likely to contain a high number of solutions. These heuristics are based on solution counting information at the level of individual constraints and revolve around the following two concepts. Given a constraint $\gamma(\mathtt{X}_1, \ldots, \mathtt{X}_k)$ and respective finite domains D_i $1 \leq i \leq k$, let $\#\gamma(\mathtt{X}_1, \ldots, \mathtt{X}_k)$ denote the number of solutions of constraint γ, called its *solution count*. Given a variable

$\mathtt{X}_i$ in the scope of γ, and a value $d \in D_i$, we will call

$$\sigma(\mathtt{X}_i, d, \gamma) = \frac{\#\gamma(\mathtt{X}_1, \ldots, \mathtt{X}_{i-1}, d, \mathtt{X}_{i+1}, \ldots, \mathtt{X}_k)}{\#\gamma(\mathtt{X}_1, \ldots, \mathtt{X}_k)}$$

the *solution density* of pair $(\mathtt{X}_i, d)$ in γ. It measures how often a certain assignment is part of a solution.

From this information we can derive simple branching heuristics, such as selecting the variable-value pair of highest solution density among all constraints or first choosing the constraint with the lowest solution count and then selecting the highest solution density pair in the scope of that constraint. One interesting feature of such search heuristics is that they are automatically built from the model since they are a product of its component constraints — adjustments to a model in order to reflect a problem's evolution over time will be accompanied by adjustments to the search heuristic.

6 Conclusion

Constraint programming is well adapted to solve rostering problems because their complex requirements are matched by the expressiveness of the approach and because the focus is on satisfaction rather than optimization. The study of rostering by CP researchers also led to the identification of new combinatorial substructures which enriched the modeling language with new families of constraints.

References

1. Trick, M.: A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. Annals of Operations Research **118**, 73–84 (2003)
2. Hentenryck, P.V., Carillon, J.P.: Generality versus specificity: An experience with ai and or techniques. In: AAAI, pp. 660–664 (1988)
3. Lecoutre, C., Szymanek, R.: Generalized arc consistency for positive table constraints. In: Benhamou [19], pp. 284–298
4. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: AAAI/IAAI, Vol. 1, pp. 209–215 (1996)
5. Régin, J.C., Puget, J.F.: A filtering algorithm for global sequencing constraints. In: G. Smolka (ed.) CP, *Lecture Notes in Computer Science*, vol. 1330, pp. 32–46. Springer (1997)
6. van Hoeve, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the sequence constraint. In: Benhamou [19], pp. 620–634
7. Pesant, G., Régin, J.C.: Spread: A balancing constraint based on statistics. In: P. van Beek (ed.) CP, *Lecture Notes in Computer Science*, vol. 3709, pp. 460–474. Springer (2005)
8. Schaus, P., Deville, Y., Dupont, P., Régin, J.C.: The deviation constraint. In: P.V. Hentenryck, L.A. Wolsey (eds.) CPAIOR, *Lecture Notes in Computer Science*, vol. 4510, pp. 260–274. Springer (2007)
9. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: M. Wallace (ed.) CP, *Lecture Notes in Computer Science*, vol. 3258, pp. 482–495. Springer (2004)
10. Sellmann, M.: The theory of grammar constraints. In: Benhamou [19], pp. 530–544
11. Quimper, C.G., Walsh, T.: Global grammar constraints. In: Benhamou [19], pp. 751–755
12. Laporte, G., Pesant, G.: A General Multi-Shift Scheduling System. J. Oper. Res. Soc. **55**, 1208–1217 (2004)
13. Bourdais, S., Galinier, P., Pesant, G.: Hibiscus: A constraint programming application to staff scheduling in health care. In: F. Rossi (ed.) CP, *Lecture Notes in Computer Science*, vol. 2833, pp. 153–167. Springer (2003)

14. Gendreau, M., Ferland, J.A., Gendron, B., Hail, N., Jaumard, B., Lapierre, S.D., Pesant, G., Soriano, P.: Physician scheduling in emergency rooms. In: E.K. Burke, H. Rudová (eds.) PATAT, *Lecture Notes in Computer Science*, vol. 3867, pp. 53–66. Springer (2006)
15. Demassey, S., Pesant, G., Rousseau, L.M.: Constraint programming based column generation for employee timetabling. In: R. Barták, M. Milano (eds.) CPAIOR, *Lecture Notes in Computer Science*, vol. 3524, pp. 140–154. Springer (2005)
16. Régin, J.C., Petit, T., Bessière, C., Puget, J.F.: An Original Constraint Based Approach for Solving over Constrained Problems. In: R. Dechter (ed.) Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP 2000), *LNCS*, vol. 1894, pp. 543–548. Springer (2000)
17. van Hoeve, W.J., Pesant, G., Rousseau, L.M.: On global warming: Flow-based soft global constraints. J. Heuristics **12**(4-5), 347–373 (2006)
18. Zanarini, A., Pesant, G.: Solution counting algorithms for constraint-centered search heuristics. In: C. Bessiere (ed.) CP, *Lecture Notes in Computer Science*, vol. 4741, pp. 743–757. Springer (2007)
19. Benhamou, F. (ed.): Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4204. Springer (2006)