Annals of Operations Research 115, 193–205, 2002

A General Approach to the Physician Rostering Problems

LOUIS-MARTIN ROUSSEAU, MICHEL GENDREAU * and GILLES PESANT
{louism,michelg,pesant}@crt.umontreal.ca
Centre for Research on Transportation, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal (Québec), Canada H3C 3J7

Abstract. This paper presents a hybridization of a Constraint Programming (CP) model and search techniques with Local Search (LS) and some ideas borrowed from Genetic Algorithms (GA). The context is the physician rostering problem, whose instances can vary greatly and for which almost no general tool has been developed. It is hoped that the combination of the three techniques will lead to an algorithm that has sufficient flexibility to solve most instances with a small amount of customization. To achieve this goal we also introduce Generic constraints: these constraints are used to model several types of ergonomic constraints that are found amongst physician rostering problems.

1. Introduction

The Personnel Rostering Problem is a very common problem present in numerous contexts. It can be described as follows: given a set of shifts for each day of the planning period, one must assign an individual or a group to cover each shift. This problem can be divided in two distinct classes, cyclic rosters and non-cyclic rosters. When building cyclic rosters one considers members of the personnel as interchangeable: the personnel then rotates on the schedules until everyone has been assigned to each schedule. Laporte [7], for example, gives details and insights on how this can be done. This planning strategy is very equitable since all the physicians will have worked exactly the same number of each shift when the rotation is completed. However it provides almost no flexibility since under a cyclic roster it is very difficult for members of the personnel to take outside engagements or to state personal preferences. This restriction was problematic in the instances we have encountered, that is why we have chosen non-cyclic rosters, where a personalized schedule is provided for each member of the personnel according to his availability and preferences.

Generating a roster for a group of physicians is often a complex task which requires taking into account a large number of (often conflicting) rules, such as the number of consecutive shifts one can work, the availability of physicians, special rules for nights and weekends and so on. In Canada most rosters are still done by hand, usually consuming a day or two of work of a highly specialized physician. The great variety

* Web: www.crt.umontreal.ca.

amongst instances is one of the major obstacles in the development of a general tool and if a lot of techniques have been developed for the rostering of nurses (Hung gives a survey in [4]) few exist for the case of physician rostering. However a good description of the problem is given by Lapierre and Carter in [5] along with a survey of six different contexts and a mathematical model to describe the different constraints they have encountered. Lapierre et al. [6] describe the Tabu Search method used by these authors to solve some of the instances presented. Although it is very efficient, this method produces cyclic roster with some minor modifications to accommodate some physicians and thus is more difficult to implement in a truly acyclic context. A pure mathematical approach is given by Beaulieu et al. in [1] where it is showed that the method developed by Berrada et al. [2] for nurse scheduling can successfully be applied to physicians. The solution time and the amount of manual customization needed to obtain good solution is prohibitive. On the constraint programming side, Trilling in [13] proposes to solve the Physician Rostering Problem as a Constraint Satisfaction Problem and to use specialized variable and value selection strategies to obtain good results in a very short time. These selection strategies are however custom tailored to the problem definition and the algorithm does not perform any search to improve the first found solution. Finally, Cheng et al. in [3] propose to use redundant modelling to significantly improve the efficiency of a CP rostering algorithm; even though their work addresses a Nurse Rostering Problem, their particular problem is sufficiently close to the physician rostering problem so that their results could be applied to a physician unit.

In order to be able to solve easily the rostering problem of several physician units, we need to devise a very general method. This method must be both *model flexible* and *algorithm flexible*. *Model flexibility* is crucial in modelling different problems as the constraints set varies greatly from one unit to another. As for *algorithm flexibility*, it insures that the method will behave efficiently on most problems, even though they have different objective functions and solution space. This paper thus presents a method combining Constraint Programming (CP), Local Search (LS) and Genetic Algorithms (GA). While CP is providing model flexibility, the conjunction of LS and GA yields a solution method that shows good results on different applications.

The proof of concept was undertaken using Ilog's OPL Studio Pro, which is a high level modelling language combined with a constraint programming solver. The set oriented programming and database access features of OPL allow rapid development of optimization software, but its high level nature also prevents the specification of customized filtering algorithms. The objective of this first phase was to show that a flexible modelling framework combined with robust algorithms can provide rapid solutions to real life applications.

The paper is organized as follows: next section presents a model for the Physician Rostering Problem along with two *generic constraints*. Section 3 presents the hybrid solution method and section 4 evaluates its performance in two real life applications. Finally section 5 compares the flexibility and performance of the proposed method with those of two existing methods, one being a pure Mathematical Programming (MP) approach and the other a pure CP one.

2. Model

2.1. CP model

The following simple Constraint Programming model was used to address the Physician Rostering Problem.

$$\begin{aligned} &\text{minimize } f(W) \\ &\text{subject to } W_{ds} \in A_{ds}, \quad A_{ds} \subseteq P, \quad \forall d \in D, s \in S, \\ &\qquad \textit{Distribution} \text{ constraints}, \\ &\qquad \textit{Pattern} \text{ constraints}. \end{aligned}$$

Here D is the set of days, S is the set of shifts each day and P represents the set of physicians. The W are assignment variables where W_{ds} shows the physician who will be working on shift s of day d. The physicians who can legally work this shift are elements of the set A_{ds}, which can be used to forbid or to preassign a physician p to a certain shift (d, s), by eliminating p from A_{ds} or setting it to a singleton ($A_{ds} = \{p\}$). The *Distribution* and *Pattern* constraints are generic constraints that are used to model all the specifics of each problem; we cover these constraints in the next section. As for the definition of $f(W)$, the objective function, it is problem dependent and needs to be specified for each physician unit: we will however give examples in section 4.

2.2. Using generic constraints

Each instance of the Physician Rostering Problem is subject to a wide variety of constraints depending on the context and traditions of each hospital. Each has a set of rules developed over the years and along the preferences of the physicians who have worked there. Some have special rules concerning night shifts and weekend shifts, as others have regulations for special kinds of shifts that are present at all time (like trauma shifts for instance). In some contexts seniority is of great importance and in others it is completely irrelevant. As we mentioned earlier, the only common aspect of all the instances we have encountered is the desire to assign physicians to shifts. To design a model that applies to most problems we need generic constraints that capture the essence of particular requirements and that can be instantiated to a particular constraint.

The motivation behind the specification of Generic Constraints was thus the development of a *model flexible* approach to the Physician Rostering Problem. We use the term model flexible to describe a method that can apply to different contexts (here physician units) with no, or very little, change to the implementation of the model.

2.2.1. Distribution constraint

The first generic constraint we propose is a distribution constraint. This constraint stipulates that a certain set of physicians P' can only work a number of shifts in set S' relative ($r \in \{=, <, \leqslant, >, \geqslant\}$) to a number n during the days in set D'. Two options are also available: $(O1)$ the worked shifts are required to be consecutive and $(O2)$ if no shift is

worked during the selected period, then the constraint may be ignored. Its signature is *Distribution*(P', S', D', r, n, $O1$, $O2$) where $O1$ and $O2$ take their value in [*true*, *false*]. Here are some applications of this constraint.

- In some hospitals physicians specify whether they would like to work their weekend shifts in one block or separated: this could be easily modeled by a distribution constraint specifying that physicians who like their weekend in one block (*Pb*) should work a number of weekend shifts (S_{we}) equal (r) to 2 (n) over each weekend (D_{we}). We also use the options ($O1 = \textit{false}$, $O2 = \textit{true}$) to insure that these physicians are not assigned to every weekend. Hence *Distribution*(*Pb*, S_{we}, D_{we}, $=$, 2, *false*, *true*) describes this constraint. A similar constraint is defined for physicians who like their weekend shifts separated (*Ps*) by setting n to 1.
- It is also very useful to limit the number (n_p) of night shifts a physician p is allowed to work per week. Here the relational operator (r) would be $\leqslant$, the set of days (D_{w}) would be weekdays and the set of shifts (S_{n}) would be night shifts. The options ($O1 = \textit{true}$, $O2 = \textit{false}$) are often used to insure that worked night shifts are consecutive and that the created schedules are ergonomic. The constraint *Distribution*($\{p\}$, S_{n}, D_{w}, $\leqslant$, n_p, *true*, *false*) thus represents this constraint.

2.2.2. Pattern constraint

The second generic constraint is what we call a pattern constraint, which is used to model patterns of shifts that physicians want to forbid or impose. Such constraints come in great variety and can be as simple or as intricate as these examples.

- It is a general rule that there must be at least a 16 hour break between shifts, so someone working on a night shift must be off the next day and evening, and one assigned to an evening shift must be off the next day.
- Some hospitals impose that no physicians should work two consecutive full weekends.
- Others require that a physician working a day shift followed by a night shift should either work night shifts or be off for the next two days.

We can generalize these constraints by seeing them as patterns. Each constraint is modeled with two patterns, one of which we will call the detection pattern (DP) and the other the forbidden pattern (FP). Each pattern is represented by a set of starting days ($\mathcal{D}$) and two increment vectors J for the DP and K the FP. For each element of the DP or FP a set of possible shifts is stored. Table 1 illustrates a pattern of length 6 where $d \in \mathcal{D}$.

The DP–FP sequence means that if a physician works a shift in DP_1 on day $d + J_1$, a shift in DP_2 on day $d + J_2$ and a shift in DP_3 on day $d + J_3$ then he must not work on any shift in FP_1 on day $d + K_1$, on any shift in FP_2 on day $d + K_2$ nor on any shift in FP_3 on day $d + K_3$. Its signature is thus *Pattern*(DP, FP, $\mathcal{D}$, J, K). The constraints previously given as examples would be modeled by the following, where S_{d}, S_{e} and S_{n} represent the set of shifts that are respectively worked during the day, evening and night and D_{s} is the set of all Saturdays.

Table 1
Structure of the Pattern constraint.

	Pattern constraint					
	Detection pattern			Forbidden pattern		
Days	$d + J_1$	$d + J_2$	$d + J_3$	$d + K_1$	$d + K_2$	$d + K_3$
Set of shifts	DP_1	DP_2	DP_3	FP_1	FP_2	FP_3

- The 16 hour break becomes *Pattern*([S_n], [$S_d \cup S_e$], D, [0], [1]) and *Pattern*([S_e], [S_d]).
- The two consecutive weekends constraint is modeled with *Pattern*([S, S], [S, S], D_s, [0, 1], [7, 8])
- The day followed by night constraint with *Pattern*([S_d, S_n], [$S_d \cup S_e$, $S_d \cup S_e$, $S_d \cup S_e$], D, [0, 1], [2, 3, 4]).

2.3. Implementation

The detailed implementation of the distribution constraint is not reported here since we have used a very straightforward OPL statement of the constraint. However, if one was to implement this constraint using a more complete CP solver, a detailed implementation of this constraint could be found in the following papers. Firstly the Distribution Constraint with both options turned turn off is equivalent to the global sequencing constraint introduced by Puget and Régin in [11]. A detailed filtering algorithm for the first option (consecutive shift requirement) can be found in the following paper of Pesant [8]. As for the second option its implementation is quite trivial.

The pattern constraint being more novel, we will detail more of its implementation. The construction of a pattern constraint using a DP–FP sequence can be done by first looking at the structure of the constraint. Since the constraint is violated when only one FP element is detected, we generate a constraint for each of those elements. This constraint should eliminate from the domain of variable W the values that contradict the FP once the DP is satisfied. Letting DP_i and FP_i be the set of shifts contained in the ith element of DP and FP, the '=' operator be a boolean function that returns 1 if the equality stands and 0 otherwise, '#' be the cardinality operator and L() be a function that returns the length of a vector, we implement the pattern constraint *Pattern*(DP, FP, $\mathcal{D}$, J, K) in OPL using this formulation:

$$\forall p \in P,\ \forall d \in \mathcal{D},\ \forall k = 1, \ldots, L(K),\ \forall s \in FP_k,$$
$$\left(L(J) = \left(\sum_{j=1}^{L(J)} \sum_{s' \in DP_j} (W_{d+J_j,s'} = p) \right) \right) \Rightarrow W_{d+K_k,s} \neq p.$$

This implementation is very straightforward since OPL Studio's grammar is very close to the mathematical programming language. However, using a different and more flexible programming language, we could improve the efficiency of this constraint. Here are some ideas.

In fact, using this representation is equivalent to generating a set of what we will call Complete Forbidden Patterns (CFP). A CFP is a sequence of length $Ł(DP) + 1$, the $Ł(DP)$ first elements are chosen in the sets of the DP while the last one is chosen in any of the sets of the FP. If all possible combinations are enumerated, we obtain a set of sequences that should never occur. $[S_n, S_d]$, $[S_d, S_e]$ and $[S_d, S_n, S_d]$ are examples of CFPs. Knowing which part is the DP or the FP is irrelevant because the sequence is forbidden as a whole. We then have a representation that is more general. A simple filtering algorithm for the CFPs is to verify that when all but one of the elements of a CFP are found, the last one is removed from the domain of the unbound variable. We could furthermore improve the efficiency of this by reducing the number of constraints generated or if early satisfaction is detected, when for instance two shifts are assigned to two different physicians, the constraint could be deactivated. Alternatively, Pesant in [9] gives a complete filtering algorithm for this constraint based on maintaining support counters.

3. Solution strategy

3.1. General idea

The Physician Rostering Problem is highly combinatorial and very difficult to solve. It is sometimes easy to obtain a feasible solution according to hard constraints but it is much more difficult to achieve one that meets all the physician's requests (requests which are most of the time impossible to meet collectively). Constructing a good solution with regard to requests generally involves developing sophisticated variable and value selection heuristics for each application since you need to grasp the essence of the hardness of a particular problem to devise efficient selection heuristics.

The motivation behind the solution method we propose is the development of an *algorithm flexible* approach. Just like model flexibility, algorithm flexibility describes the ability to perform well on most instances without the need of intricate adaptation or parameterization.

Typical rostering problems are quite easy when treated as satisfaction problems but much harder to solve to optimality. It is thus possible to generate rapidly a set a rosters (using a CP satisfaction model for instance) and then try to improve some of those solutions. By looking closely at the generated rosters we could see that some physician had very good schedules in some solution and very poor one in some others. The natural process of improvement was thus to try and combine the good individual schedules from different solutions into a single complete roster. This process is similar to the Genetic Programming framework where we would define the individual physicians schedules as chromosomes and call the combination procedure a crossover operator.

What we propose to do is to construct solutions with only little incentive towards the objective, but to do so very rapidly using a simple CP depth first search. Then, in the manner of a genetic algorithm, we generate a population of these solutions, which we rank according to a performance criteria (number of physicians who are satisfied,

number of shifts whose requirements are met or any given objective function). This population is then used as a base to an iterative process that attempts to constantly improve the quality of the best solutions. This optimization process is based on a partial reconstruction of existing solutions as well as the combination of elements coming from different solutions.

3.2. Details of the implementation

To solve our Constraint Programming model we use a Branch and Bound Search, which we try to keep as simple as possible. Selecting the variables in chronological order, since most of the constraints concern shifts that are close to each other, seems to be more efficient than the traditional *firstfail* strategy. As for value selection we use a simple *best-first* selection, where the physician whose assignment gives the best increase to the objective value is selected. As we mentioned earlier, much more intricate selection strategies have been devised ([13] is a good example), but these are context specific and more difficult to reuse.

We use the Constraint Programming model to solve the physician rostering problem with respect to the hard constraints. We generate a set (or population) of solutions by starting the search on different days. Although this step usually takes only a few seconds we do impose a cutoff time to avoid wasting time on bad instances.

To improve the pool of solutions, we try to combine attributes of the best solutions using a crossover operator. This combination generate a partial solution to the problem which is then completed by invoking the CP Solver on the incomplete part, a technique very similar to shuffling or Large Neighborhood Search [12]. This evolutionary process will tend to generate better solutions at each iteration; we can terminate it when we are satisfied with the best solution or after the maximum amount of computational time has expired.

As a basic crossover operator we use the following algorithm. We select a number $P1$ of physicians from a first parent and a number $P2$ from a second parent. We must make sure that all the selected physicians are different otherwise the resulting solution could be infeasible. We then proceed to copy the individual schedule of each of the selected physicians to the child roster and in case of conflict we simply keep the assignment of the most satisfied physician. This procedure leaves us with a partially filled roster, which we proceed to complete using our CP model and search procedure. An example of this crossover is given in figure 1.

However this basic crossover shows very poor convergence on the tested instances, so we felt the need to devise an enhanced crossover operator. The improvement lies in a selective relaxation of some variables. The idea is to identify problematic instantiation, whether they violate soft constraints or have a negative contribution to the objective function. Once those variables are identified it is only necessary to exclude them from the copying phase of the basic crossover operator. Since they are left free and that the remaining problem to solve is still simpler than the original one (most of the variables

Crossover Operator Example

P1 and **P3** are senior physicians and they have nice schedules
P2 is senior and he doesn't work enough,
P6 is part time and he works too much.

Parent #1

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Day	**P1**	**P1**	**P1**	P4	P4	**P3**	**P3**
Evening	**P3**	**P3**	**P3**	P2	P2	P4	P4
Night	P6	P6	P6	P6	P5	P5	P5

P2 and **P6** have a nice schedule but **P1** and **P3** are not satisfied.

Parent #2

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Day	P1	P3	P4	P4	P3	P4	P4
Evening	P5	P1	P3	**P2**	**P2**	**P2**	**P2**
Night	**P6**	**P6**	P5	P5	P1	P3	P3

We thus combine those rosters to generate an incomplete one.

Child

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Day	P1	P1	P1			P3	P3
Evening	P3	P3	P3	P2	P2	P2	P2
Night	P6	P6					

And we use our original Constraint Programming Model to complete the solution.

Child

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Day	P1	P1	P1	P4	P4	P3	P3
Evening	P3	P3	P3	P2	P2	P2	P2
Night	P6	P6	P5	P5	P5	P4	P4

Figure 1. Description of basic crossover operator.

being fixed a priori) the solver generally improves the instantiation to these problematic variables with the first feasible solution it finds.

This crossover operator could also be viewed as constraint programming based local search [10]. That is a local search operator that explores the neighborhood it defines by solving an optimization problem. Here the neighborhood is defined by the variables left uninstantiated, because they were not covered by any of the physicians selected for the crossover.

4. Experimentation

We have applied this solution strategy on two real-world rostering problems, the physicians of the Santa Cabrini Hospital emergency room in Montreal (case study 1) and those of Côte-des-Neiges Geriatric Hospital (case study 2). We have to note that since this project was undertaken using Ilog's OPL Studio Pro a part of the code (script) is interpreted and code optimizations were not possible. However this work was a proof of concept and a C++ code could be generated and optimized, providing a significant decrease in solution time.

4.1. Case study 1

The problem involves 23 physicians, a planning horizon of one month and six shifts per day: three during daytime, two in the evening, and one at night. Letting $\mathcal{W}$ be the set of weekends (ex: {{day 6, day 7},{day 13, day 14}, ... }) and $\mathcal{W}'$ be the sets of all weekend days (ex: {day 6, day 7, day 13, day 14, ... }), the problem is subject to the following distribution constraints:

- some physicians P_{block} want to work weekends in block
  $\forall w \in \mathcal{W}$, *Distribution*$(P_{\text{block}}, S, w, =, 2, \textit{false}, \textit{true})$,
- some physicians P_{sep} want to their weekends split
  $\forall w \in \mathcal{W}$, *Distribution*$(P_{\text{sep}}, S, w, =, 1, \textit{false}, \textit{true})$,
- some physicians P_{oneWE} do not want to work more then one weekend
  Distribution$(P_{\text{oneWE}}, S, \mathcal{W}', \leqslant, 2, \textit{true}, \textit{false})$,
- some physicians P_{noNight} do not want to work at night
  Distribution$(P_{\text{noNight}}, S_{\text{n}}, D, =, 0, \textit{false}, \textit{false})$,

and the following pattern constraints:

- 16 hour break after a night shift *Pattern*$([S_{\text{n}}], [S_{\text{d}} \cup S_{\text{e}}], D, [0], [1])$,
- 16 hour break after a Evenging shift *Pattern*$([S_{\text{e}}], [S_{\text{d}}], D, [0], [1])$.

One peculiarity of this problem is that the physicians express a minimum and maximum number of shifts they wish to work, and they are considered satisfied if they do so. The objective value is defined as follows: we count 100 units for each unsatisfied physician and 1 unit for each assignment over or under the expressed limit.

The first parameter to experiment with is the number of physicians we need to copy from both parents. Let $P1$ and $P2$ be the number of physicians copied from the first and second parent. Since there are 23 physicians, we consider the following possible values of $(P1, P2)$:

- As a baseline for comparison $(0, 0)^*$ is a single run of the CP model in optimization (rather than satisfaction) mode, with the same amount of time as the other test runs.
- $(0, 0)$ which represents a CP search with multiple restarts.
- $(10, 0)$, $(16, 0)$, $(23, 0)$ represents shuffling or CP based local search on respectively 40%, 70% and 100% of the original problem. This test run evaluates the performance of pure Local Search.
- $(5, 5)$, $(8, 8)$, $(12, 11)$ which are different possibilities for the Genetic Algorithm, with the same amount of information copied from the parents.

Since generating a feasible solution is quite easy (about 5 seconds of CPU time) and convergence is quite fast, we stop the GA after only 30 iterations or 15 minutes of CPU time, but the parameter setting of $(12, 11)$ gives an optimal solution after less then 5 minutes of computation. We use a population of 10 individuals and consider a crossover of the best 5 individuals. Of course these are very small values compared

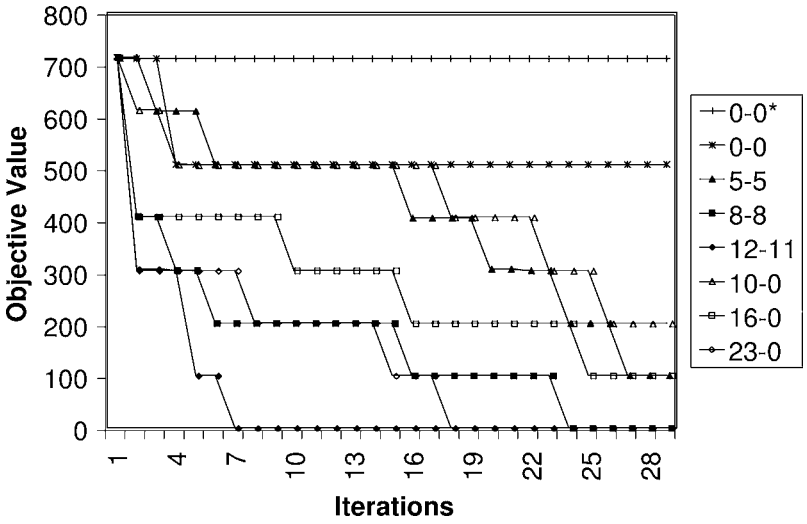

Figure 2. Santa Cabrini's results, best solution after each iteration.

to what they would be in a traditional GA, but here the use of CP based Local Search compensate for a large population.

By looking at figure 2 we make the following observations: firstly, using only the traditional branch and bound search does not provide good results. This is not surprising since we have not developed sophisticated selection heuristics but rather use a simple depth first search approach. Then we can see that the (5–5) and (10–0) give approximately the same performance: it seems that if a child solution does not inherit sufficient genes (information) from its parents, then the search space to be explored by the CP algorithm is too large for it to find an improving solution. The main factor of performance seems to be the amount of information passed along the generations. However we can see that using a genetic approach (12–11) instead of a simple shuffling or LNS strategy (23–0) pays off, as the search process shows better convergence.

4.2. Case study 2

This problem concerns 15 physicians over a planning horizon of 14 weeks. A weekday is divided into 3 shifts (*day*, *evening*, *night*). As for weekends and holidays, they are composed of a single *on call* 24 hour shift. Letting WE_{set} be the set of weekends (ex: {{day 6, day 7},{day 13, day 14}, ...}) and WE_{day} be the sets of all weekend days (ex: {day 6, day 7, day 13, day 14, ...}), the problem is subject to the following distribution constraints:

- some physicians P_{block} want to work weekends in block
  $\forall w \in WE_{\text{set}}, \mathit{Distribution}(P_{\text{block}}, S, w, =, 2, \mathit{false}, \mathit{true})$,
- some physicians P_{sep} want their weekends split
  $\forall w \in WE_{\text{set}}, \mathit{Distribution}(P_{\text{sep}}, S, w, =, 1, \mathit{false}, \mathit{true})$,

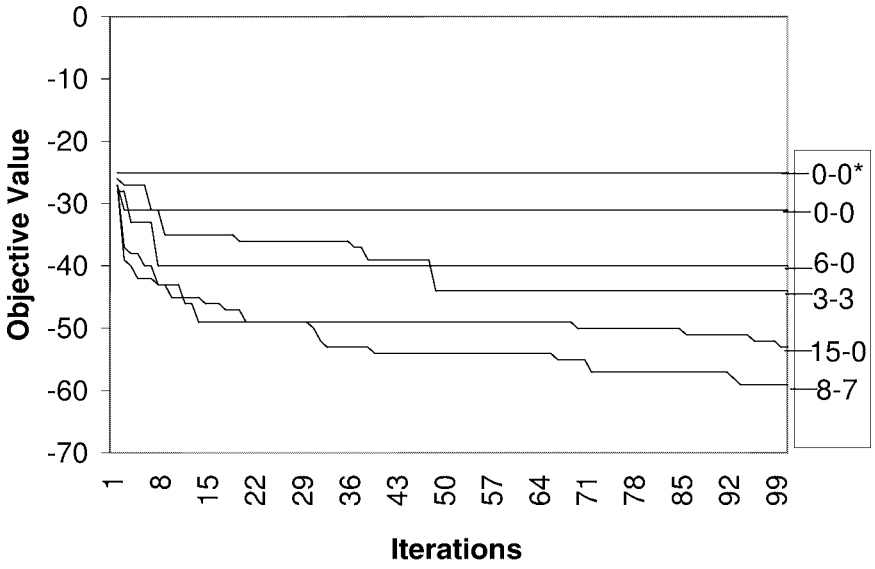

Figure 3. Côte-des-Neiges results, best solution after each iteration.

- all physicians must work a fixed number f_{ps} of each type of shift s
  $\forall p \in P, \forall s \in S, \mathit{Distribution}(\{p\}, \{s\}, D, =, f_{ps}, \mathit{false}, \mathit{false})$,

and the following pattern constaints:

- no physician should work to consecutive nights $\mathit{Pattern}([S_{\mathrm{n}}], [S_{\mathrm{n}}], D, [0][1])$,
- 8 break after a night shift or an *on call* shift $\mathit{Pattern}([\{S_{\mathrm{n}} \cup S_{\mathrm{c}}\}], [S_{\mathrm{d}}], D, [0], [1]])$.

The objective is to generate a roster that, while satisfying all the constraints, maximizes the number of preferences expressed by the physicians and evenly distributes the workload of each individual throughout the planning horizon. The objective function is the number of assignments violating a given minimal distance between consecutive shifts for a physician minus the number of expressed preferences that are satisfied.

This problem is much more complex than the preceding one, due namely to the number of constraints and the length of the planning horizon. Running the algorithm on this problem thus takes much more time (an hour an a half). We perform the same tests on this problem with all parameters taking the same values except for $(P1, P2)$ taking the values $(0, 0)^*$, $(0, 0)$, $(6, 0)$, $(3, 3)$, $(15, 0)$ and $(8, 7)$ and the number of iterations set to 100.

Figure 3 confirms the observations made on the smaller Santa–Cabrini problem, mainly that the major factor for performance is the amount of information copied from the parents to the children. Whether all that information comes from the same parent, as in shuffling, or from both parents like in the Genetic algorithm framework is of secondary importance. However combining information from both parents seems to produce greater diversification which, in turn, improves the convergence speed.

5. Method comparison

Solution methods have already been developed to solve both the case studies presented. In this section we briefly present those methods and give insights on their strengths and weaknesses. Both case studies are hard problems to solve and neither have yet been solved to optimality. However, by putting the necessary efforts, one can find good solutions to these problems. The problem is that the efforts being deployed are rarely reusable which translates in most approaches solving only the problem for which it has been developed.

5.1. Mathematical programming

A method based on the work of Beaulieu et al. [1] has been adapted (with considerable effort) to solve the problem of the Santa Cabrini Hospital, our first case study. This method is based on a mathematical program describing the problem which is solved using the standard branch and bound approach of a commercial linear solver. Although this method is not able to prove optimality on the given problem (due to time constraint) it is still able to identify a good solution quite quickly. The results obtained are similar to those obtained by our method both in quality and cpu time, but what about flexibility?

The mathematical model needs to describe in detail every constraint using a unique linear equation and a set of internal variables. Which means that if one would like to add a new pattern constraint, for instance, one would have to add a new linear equation to the model. This clearly makes the task of managing the constraints set impossible for any physician unit scheduler and prevents this method from being qualified as *model flexible*. However, since the model is solved by a very standard mathematical solver, the approach is *algorithm flexible*.

5.2. Constraint programming

Trilling has proposed in [13] a method solving the Côte-des-Neiges Geriatric Hospital problem (our second case study) which relies solely on Constraint programming. This technological choice makes this method close to being *model flexible*, since the modification needed to adapt the method to a new instance is very small.

Her approach regards the rostering problem as a satisfaction problem rather than an optimization one which means the first identified solution must be of good quality. To insure this property the author has developed sophisticated labelling strategies and value selection algorithms. The method is very efficient since it can identify a reasonably good solution in a very short time. However this extensive work is tailored around the problem it addressed (namely the distance measure between worked shifts) and cannot really be reused in another context. This approach thus does not qualify as being *algorithm flexible*.

6. Conclusion

The design of an approach equally flexible at the modelling and algorithmic levels leads to a general method that can be applied to different instances with a reduced amount of work needed for customization.

The proposed generic constraints are very useful in modelling complex constraints and allow the constraint models to be stored in a simple database or in plain text format, since all we need to store are the different sets of parameters. We are now working on constructing a simple constraint modelling tool that would generate the different generic constraints: such a tool could then be used directly by physicians or nurses to model or modify their rostering application.

Since it makes no doubt that the differences in the physician rostering policies amongst different groups of physicians are a major cause of rosters still being made by hand, it is hoped that the method presented here is a step on the way to building a general tool.

References

[1] H. Beaulieu, J. Ferland, B. Gendron and P. Michelon, A mathematical programming approach for scheduling physicians in the emergency room, Publication 1159, Département D'Informatique et de Recherche Operationnelle, Université de Montréal, Montréal (1999).

[2] I. Berrada, J. Ferland and P. Michelon, A multi-objective approach to nurse scheduling with both hard and soft constraints, Socio-Economic Planning Sciences 30 (1996) 183–193.

[3] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee and J.C.K. Wu, Increasing constraint propagation by redundant modeling: an experience report, Constraints 4 (1999) 167–192.

[4] R. Hung, Hospital nurse scheduling, Journal of Nursing Administration 25 (1995) 21–23.

[5] S. Lapierre and M. Carter, Scheduling physicians in the emergency room, in: *INFORMS*, Atlanta, (November 1996).

[6] S. Lapierre, P. Soriano, I. Buzon, S. Labbé and M. Gendreau, *Cyclic Schedules for Emergency Room Physicians*, Working Paper for the Centre for Research on Transportation (Montreal, 2000).

[7] G. Laporte, The art and science of designing rotating schedules, Journal of the Operational Research Society 50 (1999) 1011–1017.

[8] G. Pesant, A filtering algorithm for the stretch constraint, in: *Principles and Practice of Constraint Programming – CP01: The Proceedings of the Seventh Internationnal Conference*, Lecture Notes in Computer Science, Vol. 2239 (Springer, Berlin, 2001) pp. 183–195.

[9] G. Pesant, The shift change constraint, Technical report, Center for Research on Transportation (Montreal, 2001).

[10] G. Pesant and M. Gendreau, A constraint programming framework for local search methods, Journal of Heuristics 5 (1999) 255–279.

[11] J.-C. Régin and J.-F. Puget, A filtering algorithm for global sequencing constraint, in: *Principles and Practice of Constraint Programming – CP97: The Proceedings of the Third Internationnal Conference*, Lecture Notes in Computer Science, Vol. 1330 (Springer, Berlin, 1997) pp. 32–46.

[12] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: *Principles and Practice of Constraint Programming – CP98*, Lecture Notes in Computer Science, Vol. 150, Pisa, Italy (Springer, Berlin, 1998).

[13] G. Trilling, Génération automatique d'horaires de médecins de garde pour l'Hôpital Côte-des-Neiges de Montreal, Publication CRT-98-05, Centre de recherche sur les transports, Université de Montréal, Montréal (1998).