J Sched
DOI 10.1007/s10951-008-0082-8

A hybrid metaheuristic case-based reasoning system for nurse rostering

Gareth Beddoe · Sanja Petrovic · Jingpeng Li

Received: 4 July 2006 / Accepted: 7 July 2008

Abstract In this paper we present a novel Case-Based Reasoning (CBR) system called CABAROST (CAsed-BAsed ROSTering) which was developed for personnel scheduling problems. CBR is used to capture and store examples of personnel manager behaviour which are then used to solve future problems. Previous examples of constraint violations in schedules and the repairs that were used to solve the violations are stored as cases. The sequence in which violations are repaired can have a great impact on schedule quality. A novel memetic algorithm is proposed which evolves good quality sequences of repairs generated by CABAROST. The algorithm was tested on instances of the real-world nurse rostering problem at the Queens Medical Centre NHS Trust in Nottingham.

Keywords Case-based reasoning · Personnel scheduling · Nurse rostering · Memetic algorithm · Learning from failure

1 Introduction

Case-Based Reasoning (CBR) is an artificial intelligence methodology which aims to solve new problems by using information about the solutions to previous similar problems (Kolodner 1993). It operates under the premise that similar problems require similar solutions. A history of previous problem solving *episodes*, or *cases*, is stored in a database (called a *case-base*). The CBR process is often described using the '4 REs', or R4 model: REtreive, REuse, REvise, and REtain (Watson 1999). During the *retrieval* process the case(s) in the case-base containing the most similar problem(s) to the current problem are identified. The solutions in the retrieved cases are then *reused* to solve the new problem. The suggested solutions from the retrieved cases may need to be *revised* so that they are relevant in the context of the current problem. Any useful information or experience that can be gained from the current problem solving instance is *retained* for future reasoning as a new case in the case-base.

A case is a unit encapsulating knowledge relevant to a particular experience (Watson and Marir 1994). Typically, cases contain a description or representation of the problem, the solution that was used to address the problem, the outcome (i.e. success, failure or a description of the outcome state), and sometimes the context in which the case was generated (or the context in which it can be used in the future).

This paper aims to give an insight into application of CBR to constraint optimisation problems. In particular, it considers personnel scheduling problems which can be treated as constraint optimisation problems (Meyer auf'm Hofe 2000). The personnel scheduling problem is defined as the problem of placing resources into slots in a pattern, subject to given constraints, where the pattern denotes a set of legal shifts defined in terms of work to be done (Wren 1996). However, the constraints imposed on the problem cannot often be satisfied completely, and the aim is to find the solution which violates as few of them as possible. Personnel scheduling problems are found in many different types of organisations and industries, including health care institutions, manufacturing, call centres, schools, emergency ser-

G. Beddoe (✉) · S. Petrovic · J. Li
Automated Scheduling, Optimisation, and Planning Research Group, School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK
e-mail: gareth.beddoe@medicsight.com

S. Petrovic
e-mail: sxp@cs.nott.ac.uk

J. Li
e-mail: jpl@cs.nott.ac.uk

vices, energy producers, and transportation. The problem of allocation working shifts to employees is a very difficult and time consuming task. Personnel managers often spend a large percentage of their time constructing schedules and do not always succeed in producing solutions which satisfy both operational requirements and staff preferences. Poorly constructed schedules can have a number of detrimental effects on the performance of an organisation. Schedules which are perceived by employees to be inflexible or unfair can significantly affect staff morale and lead to increased absenteeism and staff turnover. This research was focused on nurse rostering. This is a particularly constrained and complex problem, which is often very difficult to solve manually. The overwhelming complexity of this problem lies in different staff qualifications, the large number of legal, management, and staff requirements which can be conflicting, general working patterns and patterns specific to some member of staff, and a wide variety of constraints. For example, management requirements for the cover and skill mix needed for a particular task often conflict with the maximum working hours allowed (by law and contract) as well as with individual staff preferences (e.g. a request for a particular shift). In general, it is difficult, if not impossible, to predict how an attempt to satisfy one requirement will affect the others. Therefore, there is increased interest in tools for automated personnel scheduling that can cope with complex problems.

Personnel rostering problems have been a subject of interest within both artificial intelligence and operational research communities for a number of decades (Burke et al. 2004). Traditional operational research optimisation methods were employed to solve such problems (Azaiez and Sharif 2005; Miller et al. 1976). Meisels and Lusternik performed a set of experiments on constraint networks, which were used to represent personnel rostering problems (Meisels and Lusternik 1998). Smith and Bennett used constraint satisfaction techniques to construct rosters for anaesthetists in a hospital (Smith and Bennett 1992). Recent research has been influenced by the recognition that most real-life rostering problems are NP-hard (Bailey et al. 1997) leading researchers to explore a number of different meta-heuristic approaches. In general, in these approaches an objective function is defined which measures solution quality by counting the number of constraint violations in rosters, usually weighted according to the perceived relative importance of each constraint type. The definition of the objective function depends heavily on the researcher's a priori understanding of the problem. Simulated Annealing and Genetic Algorithms were used to solve a nurse rostering problem involving nurses of different types (Bailey et al. 1997). Tabu search in which neighbourhoods were strategically chosen depending on the current characteristics of the search proved also to be successful (Dowsland 1998). A hybrid tabu search was developed by Burke et al. for highly constrained real-life rostering problems and is in use in a number of hospitals (Burke et al. 1999). Another tabu search approach is described in Bester et al. (2007). A method combining heuristic ordering and variable neighbourhood search is described in Burke et al. (2008). Memetic algorithms for nurse rostering were also investigated which consisted of a hybridisation of a genetic algorithm and the local optimisation based on steepest descent and a tabu search which improved members of the population between generations (Burke et al. 2001). A hyper-heuristic approach that attempts to increase the level of generality of meta-heuristic approaches to nurse rostering by iterative selection of low-level heuristics according to their on-going performance was described in Burke et al. (2003). Li and Aickelin described a Bayesian optimisation approach to nurse rostering which aims to learn good scheduling rules that determine the working pattern for each nurse (Li and Aickelin 2003; Aickelin and Li 2007). Chiaramonte and Chiaramonte described a method which employed competitive agent-based negotiation (Chiaramonte and Chiaramonte 2008). In general, meta-heuristic approaches employ a modular evaluation function which measures roster quality based on a large variety of optional constraint types including the satisfaction of coverage constraints and individual nurse preference.

Recent years have seen an acceleration of interest in developing CBR systems for a variety of combinatorial optimisation problems including scheduling. We will focus on CBR applications in scheduling. They can be classified into three groups with respect to the type of knowledge contained in the cases (Petrovic et al. 2005):

(a) A case is used to construct the whole or a partial solution to a new scheduling problem. Therefore, a case contains the description of the scheduling problem and the full or partial solution to it. This approach was used in a number of production scheduling systems (Cunningham and Smyth 1997; Legato and Monaco 2004; MacCarthy and Jou 1995). It was also successfully used in course university timetabling where each problem was represented by an attribute graph which enables the representation of courses to be scheduled and the relationships between them (Burke et al. 2000). In order to enable the CBR system to handle timetabling problems of large size, a multiple retrieval system was developed which constructs a new timetable by iteratively using partial solutions of the previous timetabling problems.
(b) A case suggests an algorithm for the new scheduling problem that worked well for a similar problem in the past. For example, such an approach was used in Schmidt (1998) where a case stored machine scheduling problems, while the cases in the case base were organised in a transformation graph which showed relations between cases. A vertex in the graph represented

a group of scheduling problems for which an appropriate algorithm was suggested. In a CBR system for examination university timetabling, instead of suggesting an algorithm for the whole problem, a heuristic which worked well in the previous similar situations was selected at each step during the construction of a timetable (Burke et al. 2006). CBR was also used to construct initial solutions for examination timetabling (Petrovic et al. 2005).

(c) A case describes a context in which a scheduling operator is used to repair/adapt a schedule in order to improve its quality (in terms of constraint satisfaction). Miyashita and Sycara (1995) used the knowledge capturing capabilities of CBR to elicit human experts' preference for particular measures of roster quality when making repairs to sub-optimal schedules. The research presented in this paper is focused on the operator reuse in personnel rostering.

Despite the existence of various CBR approaches to a wide range of scheduling problems limited research has been conducted on the investigation of CBR approaches to personnel scheduling problems. Moreover, a majority of the algorithms developed so far for personnel scheduling problems tend to form the solution from scratch, starting from an empty roster, rather than understanding a new problem in terms of old experience. To the best of our knowledge Scott and Simpson were the only researchers who combined CBR and constraint logic programming in solving simple nurse rostering (Scott and Simpson 1998). A case-base of commonly used weekly shift patterns was collected from manual rostering solutions. These shift patterns were then allocated to nurses according to the number and type of nurses in the problem being solved. They were used as a "draft solution" which was then improved with local search.

In this paper we demonstrate the possibilities of case-based reasoning in personnel scheduling. A system named CABAROST (CAse-BAsed ROSTering) was developed for nurse rostering in modern hospitals. The nurse rostering serves as an excellent framework for investigation of CBR in complex optimisation problems with a variety of constraints and personnel preferences (e.g. individual preferences for working shift types in a particular order). The main idea is to capture the knowledge/experience of human experts used in solving constraint optimisation problems by storing a history of constraint violations and their corresponding repairs. A number of research issues that arise in using CBR in constraint optimisation problems are addressed including: (a) representation of cases for constraint optimisation; (b) generalisation of case indices which will make the cases applicable to new problem instances; (c) adaptation of the repair of the retrieved case to meet the context of the new problem; (d) training of the case-base; and (e) learning from failure.

We adopted an approach to constraint optimisation problems where a large number of constraint violations have to be repaired in an iterative fashion, thus generating sequences of repairs. Obviously, the order in which constraint violations are repaired can have a large effect on the final solution quality (Sadeh et al. 1995). We suggest an evolutionary approach, which searches for good quality sequence of generated repairs. A hybridised system has been developed in which a genetic algorithm was used to evolve the repair sequences while the CBR approach was employed to repair the initial constraint violation and newly created violations at each generation.

The research work described in this paper was guided by the real-world nurse rostering problem at the Queen's Medical Centre University Hospital Trust (QMC) Nottingham, United Kingdom. All cases that are stored in the case-base are acquired during the course of actual scheduling/rescheduling actions of the head nurse of one of the wards (the Ophthalmological Ward).

The paper is organised as follows. In Sect. 2 we introduce a formalised description of the rostering problem. Section 3 gives details of the CABAROST system for solving constraint violations in nurse rostering. In order to utilise the previous experience in rostering the training cases are acquired in the course of actual scheduling (rescheduling) actions performed by the personnel manager. The training cases also reflect the preferences of the personnel managers, which are usually difficult to model explicitly. The memetic algorithm is described in Sect. 4 incorporating case-based repair generation and genetic algorithm for searching for good repair sequences. In Sect. 5 the performance of the developed variants of the algorithm are compared using a number of problem instances from the QMC. A discussion of the benefits of the methods proposed is given in Sect. 6, followed by conclusions.

2 Problem specification

In this section we will give a general formal specification of the personnel scheduling problem. This specification will then be further detailed to describe the nurse rostering problem. The given specification differs from the standard constraint programming specification which is based on the set of decision variables and the associated variable domains. The emphasis in our problem specification is on violations of constraints and their repairs.

A roster is defined as a set of assignments of employees to shifts on each day over a specific period (usually 4 weeks). Rosters should conform to a set of constraints, which reflect legal, management, and staff requirements. Initially, a set of shift preferences is collected from each staff member, indicating which shift they would prefer to work on each day.

This set of shift preferences forms the initial roster, which will be improved by 'repairing' the violations of the constraints which it inevitably contains.

The general personnel scheduling problem is represented as a 5-tuple:

$$\mathcal{PS} = \langle \mathcal{N}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \varepsilon \rangle, \tag{1}$$

where $\mathcal{N}$ is the set of employees, $\mathcal{P}$ is the set of shift preferences expressed by each employee, $\mathcal{R}$ is the set of shifts assigned to each employee, $\mathcal{C}$ is the set of constraints imposed on the shift assignments in $\mathcal{R}$, and ε is the set of *constraint evaluators* which are used to identify violations of the constraints in the roster $\mathcal{R}$.

The set of employees,

$$\mathcal{N} = \{empl_n : n = 1, \ldots, N\}, \tag{2}$$

represents the N employees to be scheduled, where $empl_n$ represents information about the nth employee. Each $empl_n$ is assigned a *type* $emplType_t$, $t = 1, \ldots, T$, which is problem domain specific, and represents employee characteristics, such as qualifications, skill level, and experience. For the nurse rostering problem these will be defined in Sect. 2.1.

The set of shift preferences,

$$\mathcal{P} = \{P_n : n = 1, \ldots, N\}, \tag{3}$$

represent the preferences for particular shifts of each employee $empl_n$. The preferences of each employee $empl_n$ are represented by a vector P_n, where

$$P_n = \big\{\langle p_{n,d}, h_{n,d} \rangle : d = 1, \ldots, D\big\}, \quad n = 1, \ldots, N, \tag{4}$$

where $p_{n,d}$ represents the shift that the nth employee would prefer to work on day d, $h_{n,d}$ is a Boolean variable which indicates whether the corresponding preference $p_{n,d}$ is a hard or a soft request, respectively, and D is the number of days over which the rostering is performed.

For example, if employee n requests an "Early" shift on day d then $p_{n,d}$ would be have the value "Early". If this request represented an essential requirement (e.g. it forms part of the employee's employment contract or has been requested to meet a particular personal requirement) then the corresponding $h_{n,d}$ value would be set to *true*, indicating that it is a hard constraint. Alternatively, if the request is desirable but not essential then the $h_{n,d}$ value would be set to *false*, indicating that the shift request is soft constraint.

The set of shift assignments,

$$\mathcal{R} = \{R_n : n = 1, \ldots, N\}, \tag{5}$$

represents the shifts assigned to each employee $empl_n$ in the roster. The shift assignments of each employee $empl_n$ are represented by a vector R_n, where

$$R_n = \big\{\langle s_{n,d} \rangle : d = 1, \ldots, D\big\}, \quad n = 1, \ldots, N, \tag{6}$$

where $s_{n,d}$ is the shift assignment of employee $empl_n$ on day d, and D is the number of days over which the rostering is performed. Initially, the shifts for each employee are the same as the preferences indicated in $\mathcal{P}$. These shift assignments in $\mathcal{R}$ will be changed in order to satisfy the constraints in $\mathcal{C}$ as much as possible.

The set of constraints,

$$\mathcal{C} = \{Constraint_k : k = 1, \ldots, K\}, \tag{7}$$

represents the K constraints imposed on the shift assignments. Each constraint is a 3-tuple:

$$Constraint_k = \langle cType_k, EmplType_k, cParamSet_k \rangle, \quad k = 1, \ldots, K, \tag{8}$$

where $cType_k$ is the type of constraint, $EmplType_k$ describes the types employees whose shift assignments will be affected by the constraint, and $cParamSet_k$ is a set of parameters specific to the constraint type. The definitions of $cType_k$, $EmplType_k$, and $cParamSet_k$, depend on the problem domain and will be described for the nurse rostering problem in Sect. 2.1.

The set of constraint evaluators,

$$\varepsilon = \{e_k : k = 1, \ldots, K\}, \tag{9}$$

represents the constraint evaluators assigned to each constraint in $\mathcal{C}$ which evaluate roster $\mathcal{R}$. The constraint evaluators take into consideration the data about employees, their preferences and assigned shifts, and produces a set of violations of each constraint. Each constraint evaluator e_k is defined as a function:

$$e_k(Constraint_k, \mathcal{N}, \mathcal{P}, \mathcal{R}) = \{violation_{k_m} : m = 1, \ldots, M_k\}. \tag{10}$$

Each violation has the form:

$$violation_{k_m} = \langle cType_k, vParamSet_{k_m} \rangle, \quad k_m = 1, \ldots, M_k, \tag{11}$$

where $vParamSet_{k_m}$ is a set of parameters describing the violation of the constraint of type $cType_k$.

Finally, the concept of a *repair* is introduced as an operator which changes the shift assignments in the roster $\mathcal{R}$. Each repair describes the shift assignments which are to be changed, the employees involved, and the new constraint violations which are generated as a result of its application to roster $\mathcal{R}$. A repair is defined as:

$$Repair = \langle rType, rParamSet, ViolationList \rangle, \tag{12}$$

where *rType* is the type of repair, *rParamSet* is a set of parameters specific to the type of repair, and *ViolationList* =

$\{violation_j : j = 1, \ldots, J\}$ is a list of the J new violations which are *caused* by applying the repair to the roster.

The task is to find a sequence of repairs so that the roster $\mathcal{R}$ satisfies the constraints in $\mathcal{C}$ as much as possible.

2.1 Nurse rostering problem specification

Based on the description of the general personnel scheduling problem we present a formal description of the nurse rostering problem considered in our research. It will be followed by some illustrative examples. For the nurse rostering problem elements of the set $\mathcal{N}$ will be denoted by $nurse_n$ (defined as $empl_n$ in the previous section).

Each nurse is assigned a type which is presented as a 6-tuple:

$$NurseType_t = \langle qual_t, gend_t, intl_t, train_t, grade_t, hours_t \rangle, \quad t = 1, \ldots, T, \tag{13}$$

where the elements of $NurseType_t$ are given in Table 1, and T is the number of different types of nurses (the $NurseType_t$ replace the $emplType_n$ variables in the previous section).

In the QMC ward nurses belong to one of four possible qualification categories. These are, in descending order of seniority: registered (RN), enrolled (EN), auxiliary (AN), and student (SN). Registered nurses are the most qualified and have had extensive training in both the practical and managerial aspects of nursing, whereas enrolled nurses have received mainly practical training. Auxiliary nurses are unqualified nurses who can perform basic duties and student nurses are training to be either registered or enrolled. These four nurse qualifications are classified hierarchically by using three additional qualifications. Registered and enrolled nurses are classified as *qualified* (QN) while qualified and auxiliary nurses are both *employed* (PN). Finally, XN denotes nurses of any qualification level.

The gender and nationality of a nurse are also factors that affect rostering decisions. Registered and enrolled nurses can receive additional training specific to the ward that they work in. In the ophthalmology ward these nurses receive *eye-training*, denoted ET (a nurse without eye-training is called *non-eye-trained* denoted NT). The grade of each nurse is determined by a National Health Service (NHS) standard which is based on the qualification level of the nurse and the amount of hospital experience that they have.

In our nurse rostering problem there are five possible values for shifts in the roster:

UNASSIGNED: (**U**) No shift is currently assigned and the nurse has no request.
OFF: (**O**) No shift is currently assigned and the nurse has requested that they do not work.
EARLY: (**E**) The early shift (07:00–14:45).
LATE: (**L**) The late shift (13:30–21:15).
NIGHT: (**N**) The night shift (21:00–07:15).

The nurse rostering problem that was investigated is characterised by a variety of different constraint types. In total 12 constraint types were identified:

Cover constraints define the minimum number of nurses of a particular type that must be assigned to a particular shift. For example, the early shift requires 4 qualified nurses. A number of cover constraints imposed on a single shift together define the skill mix that must be present on the ward for that shift.

HardRequest constraints are violated if a nurse's preferred shift is different from their assigned shift, and the preferred shift is a hard request.

MaxDaysOn constraints limit the number of days that nurses may work in a row. For the QMC ward this is generally 6 for all nurses.

MaxHours constraints set the maximum number of hours a nurse may work over a period. This can differ considerably between nurses and depends on their individual working contracts. For example, full time nurses at the QMC ward may not work more than 75 hours in a fortnight.

MinDaysOn constraints define the minimum number of days that nurses may work successively.

Table 1 The fields of *nurseType*

Field	Domain	Description
qual	{RN, EN, AN, SN, QN, PN, XN}	The qualification level of the nurse
gend	{M, F}	The gender of the nurse—(M)ale or (F)emale
intl	{I, H}	The nationality of the nurse—(I)nternational or (H)ome
train	{ET, NT}	The specialty training of the nurse—eye-trained (ET) or non-eye-trained (NT)
grade	{A, B, C, D, E, F, G, H, I}	The *grade* of the nurse according to the NHS grading system
hours	[8, 37.5]	The number of hours the nurse should work each week (by contract)

MinHours constraints set the minimum number of hours a nurse may work over a period. This constraint is violated if nurses time is under-utilised by the roster.

SingleNight constraints are violated if nurses are assigned individual night shifts. Nurses at the QMC ward prefer to work night shifts in blocks of two or more.

SoftRequest constraints are violated if a nurse's preferred shift is different from their assigned shift and the preferred shift is a soft request.

Succession constraints define illegal shift combinations for nurses. It is not desirable to work shifts of one type on one day followed by shifts of another type on the day after. For example, a NIGHT shift followed by an EARLY shift is one such combination.

WeekendBalance constraints set the number of weekends that nurses may work over a period. They state the maximum number of weekends a nurse may work in any given number of weekends. For example, nurses in the QMC ward may not work more than 3 weekends out of every 4, unless it is stipulated in their contracts.

WeekendsInARow constraints set the maximum number of weekends that nurses may work in a row. This is usually three in the QMC ward.

WeekendSplit constraints are violated if a nurse works only one of the days in a weekend.

Following the definition of constraints given in (8), a parameter set is associated with each constraint type. For illustration purposes the parameter sets are given here for 3 constraint types:

Cover: $cParamSet = \langle shift, min \rangle$ where $shift \in \{\mathbf{E}, \mathbf{L}, \mathbf{N}\}$ and $min \in \mathbb{Z}^+$. It is interpreted that there must be at least *min* nurses working the indicated *shift*.

HardRequest: $cParamSet = \langle\rangle$ which means that no additional parameters are used to describe these constraints;

MaxDaysOn: $cParamSet = \langle max \rangle$ where $max \in \mathbb{Z}^+$. It denotes that nurses may work no more than *max* days in a row.

The parameters for violations (see (11)) of these three constraints are:

Cover: $vParamSet = \langle NurseType, day, shift \rangle$ where *NurseType* is taken from the constraint which generated the violation, $day \in [1, D]$ and $shift \in \{\mathbf{E}, \mathbf{L}, \mathbf{N}\}$. The interpretation is that there are insufficient nurses of type *NurseType* assigned to *shift* on *day*.

HardRequest: $vParamSet = \langle day, nurse \rangle$ where $day \in [1, D]$ and $nurse \in N$. These parameters denote that the hard request of *nurse* on *day* has been violated.

MaxDaysOn: $vParamSet = \langle startDay, numDays, nurse \rangle$ where $startday \in [1, D]$, $numDays \in [2, D - startDay]$, and $nurse \in N$. Here *nurse* is working too many shifts in a row starting on *startDay* for *numDays* days.

As an example, a constraint which restricts the maximum number of consecutive days on which any nurse may work to six is defined as follows:

$$\langle MaxDaysOn, \langle \mathrm{XN}, *, *, *, *, * \rangle, \langle 6 \rangle \rangle, \tag{14}$$

where the wildcard symbol * in the $NurseType_t$ structure indicates that the element may take any value.

A violation of this constraint involving $nurse_8$ starting on day 0 for six days would be written:

$$\langle MaxDaysOn, \langle 0, 6, nurse_8 \rangle \rangle. \tag{15}$$

Violations of the constraints are repaired by four different operations: reassign, swap, switch, and ignore. These repairs are commonly used by personnel managers. They are defined below along with the parameter sets used to describe them.

Reassign: This repair assigns *shift* to *nurse* on *day*. $rParamSet = \langle nurse, day, shift \rangle$ where $nurse \in \mathcal{N}$, $day \in [1, D]$, and $shift \in \{\mathbf{U}, \mathbf{E}, \mathbf{L}, \mathbf{N}\}$.

Swap: This repair interchanges the shift assignments of $nurse1$ and $nurse2$ on *day*. $rParamSet = \langle nurse1, nurse2, day \rangle$, where $nurse1, nurse2 \in N$ and $day \in [1, D]$.

Switch: This repair interchanges the shift assignments of *nurse* on $day1$ and $day2$. $rParamSet = \langle nurse, day1, day2 \rangle$, where $nurse \in N$ and $day1, day2 \in [1, D]$.

Ignore: This repair does not change anything in the roster. $rParamSet = \langle\rangle$.

For example, according to (12) a repair which assigns the EARLY shift to $nurse_7$ on day 3 would be written:

$$\langle Reassign, \langle nurse_7, 3, \mathbf{E} \rangle, ViolationList \rangle. \tag{16}$$

The set of violations of constraints in an instance of a rostering problem and the set of possible repairs that can be made to the roster are described as the spaces of violations and repairs for a given instance of the rostering problem, respectively. Given $\mathcal{PS} = \langle \mathcal{N}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \varepsilon \rangle$, $P_v^{\mathcal{PS}}$ denotes the instance space of violations and $P_r^{\mathcal{PS}}$ denotes the instance space of repairs.

3 The case-based rostering method

The main idea of the CAse-BAsed ROSTering (CABAROST) system is to capture examples of individual constraint violations and the repairs that were used by personnel managers to solve them. Violations and repairs are stored as cases in a case-base. Each violation and its corresponding repair represents a *case* which can be used to solve new violations.

Cases are organised in a case-base. When a new violation is encountered the case-base is searched for cases containing

similar violations. If such cases can be found then the repairs stored in these cases are *adapted* so that they can be used to repair the new violation. Finally, each time a violation is repaired the experience can be stored as a new case in the case-base, thus increasing the ability of the case-base to solve future problems. Figure 1 gives a flow chart of the system.

To each instance of a rostering problem we associate a space of violations, that is defined as the union of all the sets of violations generated by applying the constraints in $\mathcal{C}$ to $\mathcal{R}$, and the space of repairs that contains all the repairs that can be performed.

The violations and repairs contained in the cases must be independent of any concrete problem instance. This is essential if the experience stored about solving violations in previous problem instances is to be used to solve violations in new instances. This can be illustrated by considering a MaxDaysOn violation involving a particular nurse, $nurse_{10}$, in N. In order to solve this violation cases containing similar violations are retrieved from the case-base. The applicability of past experience is increased if the search is not limited to previous violations involving only $nurse_{10}$. Violations of the same type involving other nurses must also be considered. Hence, the experience stored about repairing the violations in one roster can be used to solve violations (of the same type) in any other roster, regardless of any staffing differences.

This concept is achieved through the use of *generalisation functions* which convert violations and repairs from concrete instances of a problem into generalised to be stored in the case-base. This process of generalisation ensures that only information relevant to general problem solving is stored in the case-base—e.g. nurse type rather than specific nurses, repair types rather than specific actions.

3.1 Case structure

The case-base, *CB*, is a database of previous violations of constraints and their corresponding repairs. A case is therefore defined as an ordered pair:

$$case_\gamma = \langle v_\gamma, r_\gamma \rangle, \quad \gamma = 1, \ldots, \Gamma, \tag{17}$$

where v_γ and r_γ represent a generalised violation and a generalised repair, respectively, and Γ is the total number of cases. A summary of the data stored in each case is presented in Fig. 2. They will be explained in the remainder of this section. The violation part contains structural information and features. Structural information describes the parameters of violations and repairs. It is used in the first phase of the retrieval process for initial filtering of the case-base. Feature values are statistical measurements of the roster at

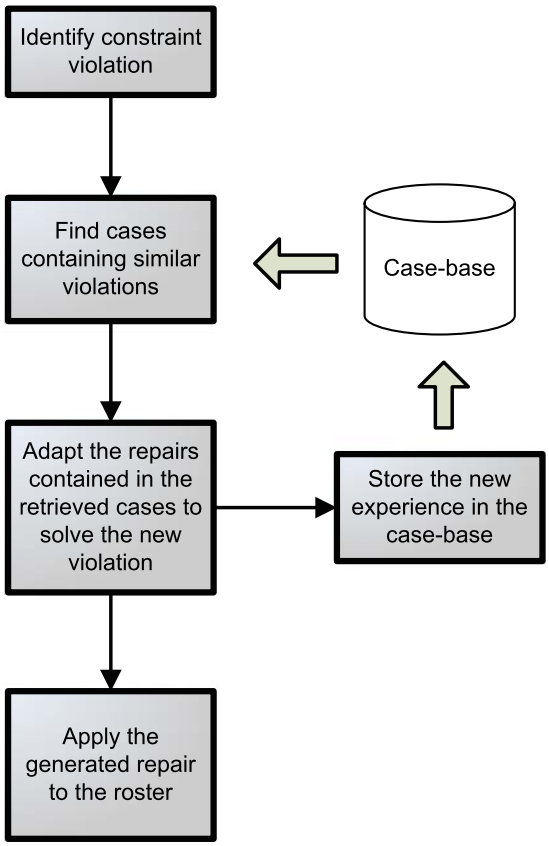

Fig. 1 Flow chart of the system

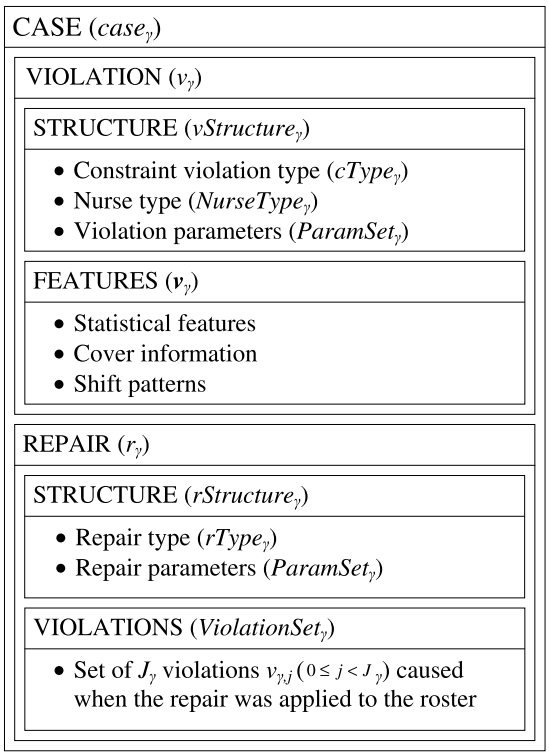

Fig. 2 Case structure

the time a case was created. These values are used to define the *similarity* of violations and repairs.

A violation is defined in a formal way as a duple,

$$v_\gamma = \langle vStructure_\gamma, \bar{\mathbf{v}}_\gamma \rangle, \quad \gamma = 1, \ldots, \Gamma, \tag{18}$$

where $vStructure_\gamma$ represents *structural* information extracted from the roster about the violation being stored and $\bar{\mathbf{v}}_\gamma$ is a vector of violation *feature* values.

The element $vStructure_\gamma$ is defined as follows

$$vStructure_\gamma = \langle cType_\gamma, NurseType_\gamma, vParamSet_\gamma \rangle, \quad \gamma = 1, \ldots, \Gamma, \tag{19}$$

where $cType_\gamma$ is the type of constraint which was violated, $NurseType_\gamma$ is the information about the nurse type that was involved, and $vParamSet_\gamma$ is a set of parameters which depends on the constraint type.

For example, the parameter sets stored for each of the examples of constraint violations that were described in Sect. 2.1 are:

Cover: $vParamSet = \{shift\}$ where $shift \in \{\mathbf{E}, \mathbf{L}, \mathbf{N}\}$.
HardRequest: $vParamSet = \{assigned, requested\}$, where $assigned \in \{\mathbf{U}, \mathbf{E}, \mathbf{L}, \mathbf{N}\}$ and $requested \in \{\mathbf{O}, \mathbf{E}, \mathbf{L}, \mathbf{N}\}$.
MaxDaysOn: $vParamSet = \emptyset$.

The *violation* has to be generalised in the sense that it should identify information about the violation which is not specific to the current roster. For example, only the *NurseType* information is stored about any nurses that are involved in the violation, rather than all of the information about these nurses. Concrete violations are generalised using a function $\theta_v^{\mathcal{PS}}$, defined for the violation problem instance space $P_v^{\mathcal{PS}}$, which converts a concrete violation into its generalised form. When a concrete violation is generalised the *NurseType* and other parameter information is extracted from it to be stored in the case-base.

For example, in the nurse rostering problem when the actual HardRequest violation

$$\langle HardRequest, \{day, nurse_n\} \rangle \tag{20}$$

is generalised then the structural information stored in the case is:

$$\langle HardRequest, NurseType_n, \{s_{n,day}, p_{n,day}\} \rangle \tag{21}$$

denoting a hard request violation of a nurse of type $NurseType_n$ who requested shift $p_{n,day}$ but was allocated shift $s_{n,day}$.

The vector of F feature values, $\bar{\mathbf{v}}_\gamma = \{v_{\gamma,f} : f = 1, \ldots, F\}$, stores a number of measurements of the roster at the time which the violation took place. The features vary considerably depending on the constraint violation type.

The features fall into one of three categories:

Statistical information is stored about various aspects of the roster. This includes measurements of the current levels of shift assignment (as percentages of the total number of assignable hours available), nurse preference satisfaction, the magnitude of the violation, and the total number of constraint violations. These measurements can be taken over the whole roster or certain subsets of nurses and days, depending on the violation being stored.

Cover information is the measurement of the number of nurses working on a specified shift. This is subdivided by nurse type and the shifts measured depend on the type of violation.

Shift Patterns are recorded on and around days during which violations have occurred. This is particularly useful for violations which involve a single nurse over a relatively short period—for example, Succession and MinDaysOn violations. The knowledge of shift patterns allows rostering decisions to be made which take into account the assignments that nurses have on the days either side of the day on which the violation occurs.

These features have been selected automatically from a large candidate set using a genetic algorithm. A description of this algorithm, and a detailed discussion of the different features used, can be found in Beddoe and Petrovic (2006). Examples of the features are given in Table 2.

The second half of each case stores the repair used to solve the violation. It is also stored in a generalised form:

$$r_\gamma = \langle rStructure_\gamma, ViolationSet_\gamma \rangle, \quad \gamma = 1, \ldots, \Gamma, \tag{22}$$

where $rStructure_\gamma$ describes structural information about the repair, and $ViolationSet_\gamma$ is the set of new violations which were *caused* as a result of performing the repair.

The element $rStructure_\gamma$ contains the structural information about a repair, i.e. information about the types of nurses and shifts that were involved in the repair. It is defined as:

$$rStructure_\gamma = \langle rType_\gamma, rParamSet_\gamma \rangle, \tag{23}$$

where *rType* is the type of repair and $rParamSet_\gamma$ describes the parameters which have been taken from the actual repairs being generalised.

In the nurse rostering example the parameters stored for each repair type are:

Reassign: $rParamSet = \langle NurseType, newShift, oldShift \rangle$, where *NurseType* is taken from the *nurse* field of the actual repair, *oldShift* is the shift that the nurse originally had on the day of the repair, and *newShift* is what they were reassigned.

Swap: $rParamSet = \langle NurseType1, NurseType2, shift1, shift2 \rangle$, where $NurseType1$ and $NurseType2$ are taken from the *nurse*1 and *nurse*2 fields of the actual repair, respectively, and *shift*1 and *shift*2 are the shifts originally assigned to each of the nurses on the day of the repair.

Table 2 Violation features

Feature	Description
Number of violations	The number of constraint violations in the current roster
Violation magnitude	The degree of magnitude of the violation (e.g. the shortfall in the number of required nurses of a particular type)
Percentage assigned hours	The ratio of the hours which are currently assign to the total hours that can be assigned
Preference satisfaction	The proportion of shift requests which are met by the current shift assignments
Shift pattern	The shift pattern of a nurse over a period relevant to the violation parameters
On-off pattern	Similar to the shift pattern except that this feature indicates whether the nurse was assigned any shift on each day, or if they were off
Qualification cover array	An array which records the number of nurses of each qualification type over a number of days relevant to the violation type (e.g. two weeks for a MaxHours violation which specifies the maximum number of hours that a nurse can work in a fortnight)
Specialty training cover array	An array which records the number of nurses of each specialty training over a number of days relevant to the violation type (e.g. two weeks for a MaxHours violation which specifies the maximum number of hours that a nurse can work in a fortnight)

Switch: $rParamSet = \langle NurseType, shift1, shift2 \rangle$, where *NurseType* is taken from the *nurse* field of the actual repair, *shift*1 was the nurses shift on *day*1 of the repair, and *shift*2 was the nurses shift on *day*2.

Ignore: $rParamSet = \langle\rangle$. No repair was performed.

The set of new violations that are caused by the repair is also stored in each case. The potential *damage* that will be done to a roster must be considered when choosing between various possible repair actions. The repair described in the case causes a list of violations $ViolationSet_\gamma$ which were generalised using $\theta_v^{\mathcal{PS}}$. The generalisation of violations allows repairs to be compared by considering the *damage* that they might do to the roster. Generalised violations will also be used in the calculation of the repair similarity.

The generalisation function for repairs, $\theta_r^{\mathcal{PS}}$, converts a concrete repair into its generalised form. This function is used during the adaptation process described later in the paper, and to store information about a repair in a new case.

3.2 Case retrieval

The case retrieval process takes a violation from the roster. This violation is called the *focus violation*. Violations are either chosen randomly when rosters are being repaired or by the user who can choose the violation interactively from a generated list. The goal of the case retrieval process is to rank relevant cases in the case-base according to their similarity to the focus violation.

The focus violation must first be generalised (using the $\theta_v^{\mathcal{PS}}$ function) so that it can be compared to the stored violations in the case-base. The retrieval algorithm, whose pseudo-code is given in Fig. 3, involves two distinct phases. In the first phase the cases in the case-base whose violations match the structural information of the focus violation are identified. This ensures that the retrieved cases contain only violations of the same type, involving the same types of nurses and shifts, as the focus violation.

The first phase results in a subset of cases CB_{TRIM} containing only matching violations. In the second phase these matching violations are then ranked according to the distance of their feature vector $\bar{\mathbf{v}}_\gamma$ from the feature vector of the focus violation, $\bar{\mathbf{v}}_\alpha$. Distance is calculated using a weighted nearest neighbour function:

$$d_v(\bar{\mathbf{v}}_\gamma, \bar{\mathbf{v}}_\alpha) = W_\gamma \sqrt{\sum_{f=1}^{F} w_f f_f(v_{\gamma,f}, v_{\alpha,f})^2}, \tag{24}$$

where $f_f(v_{\gamma,f}, v_{\alpha,f})$ is the distance function for the fth feature and depends on the feature type, F is the number of features, w_f, $f = 1, \ldots, F$, are feature weights W_γ is the case weight. The distance functions for each feature type are defined according to the nature of its values. For example, for real-valued features the straight line distance measure is used (i.e. $|b - a|$), while for more complex features specialised distance measures have been developed and are described in detail in Beddoe and Petrovic (2006).

The distance measure described by expression (24) returns a value indicating the *dissimilarity* of the two cases. Cases containing two identical violations will have zero distance between them. It should be noted that all of the distance functions used to measure the distance between individual feature values normalise the output to the inter-

Given $violation_\alpha$, case-base CB:
1: create $CB_{\text{TRIM}} \leftarrow \theta$
2: generate $\theta_n^{\mathcal{PS}}(violation_\alpha) \rightarrow \langle vStruct_\alpha, \bar{\mathbf{v}}_\alpha \rangle$
3: **for all** $case_\gamma = \langle v_\gamma, r_\gamma \rangle \in CB$ where $v_\gamma = \langle vStruct_\gamma, \bar{\mathbf{v}}_\gamma \rangle$ **do**
if $vStruct_\gamma = vStruct_\alpha$ **then** $CB_{\text{TRIM}} \leftarrow CB_{\text{TRIM}} \cup \{case_\gamma\}$
4: **if** $CB_{\text{TRIM}} = \theta$ **then return** *false*
5: generate an array $vDistance\big[|CB_{\text{TRIM}}|\big]$
6: **for all** $case_\gamma = \langle v_\gamma, r_\gamma \rangle \in CB_{\text{TRIM}}$ where $v_\gamma = \langle vStruct_\gamma, \bar{\mathbf{v}}_\gamma \rangle$ **do**
$vDistance[\gamma] \rightarrow d_v(\bar{\mathbf{v}}_\gamma, \bar{\mathbf{v}}_\alpha)$
7: **return** CB_{TRIM} sorted according to *vDistance*

Fig. 3 The retrieval algorithm

Table 3 A simple example of a nurse roster with nurse preferences

	0	1	2	3	4	5	6
$nurse_0$	E (E)	U (U)	U (U)	**U (O)**	L (L)	E (E)	E (E)
$nurse_1$	L (L)	L (L)	L (L)	**U** (U)	U (U)	L (L)	U (U)
$nurse_2$	U (U)	E (E)	E (E)	**L** (L)	E (E)	U (U)	**L** (E)

val [0,1]. For example, real-valued data is normalised by using the minimum and maximum values for the feature present in the case-base.

In the pseudo-code algorithm given in Fig. 3 the input data are the focus violation $violation_\alpha$ and the case-base CB. In line 1 an empty case-base is created which will store the output result and in line 2 the focus violation is generalised. The case-base is then searched for all cases containing violations with structural information which is similar to the focus violation in line 3. If no such cases can be found then the process returns with no results in line 4. In line 5 an array is created which is the same size as the filtered case-base CB_{TRIM} and in line 6 the distance between the features of the focus violation and each of the cases in the filtered case-base are calculated using (24). Finally, in line 7 the filtered case-base CB_{TRIM} is sorted according to the values in *vDistance* and returned.

To demonstrate the retrieval process we shall consider a simple example. Table 3 describes a roster for 3 nurses over a week long period. The early (E), late (L), and unassigned (U) shift types have been assigned to the nurses during this period. In each field of the table, the shift assigned to a nurse and their preferred shift (in brackets) has been indicated. In this example the only violation of a nurse's preference has occurred on day 6, where $nurse_2$ is assigned a late shift in place of the preferred early shift. On day 3 $nurse_0$ has indicated that an off shift (O) is desired, which in the current roster has been satisfied.

In this example, $nurse_0$ and $nurse_1$ are registered, female, non-international, eye-trained, E-grade nurses and $nurse_2$ is an enrolled, female, non-international, eye-trained, D-grade nurse, based on the descriptions of these characteristics given in Sect. 2.1 of this paper.

Let us suppose that a single "cover" type constraint is imposed on the roster, requiring that a minimum of 1 qualified nurse is assigned to every early shift. It can be seen that on day 3 there is no nurse assigned to the early shift and this represents a violation of the constraint. The violation that needs to be repaired is represented as:

$$violation_\alpha = \langle Cover, \langle \text{QN}, *, *, *, *, * \rangle, \langle 3 \rangle, \langle \text{E} \rangle \rangle \tag{25}$$

which describes a violation of type *cover* in which there are insufficient qualified nurses (QN) on day 3 assigned to the early (E) shift.

The identified violation is firstly generalised using $\theta_v^{\mathcal{PS}}$ so that it can be compared to violations in the case-base. This gives us the following violation structure:

$$vStructure_\alpha = \langle Cover, \langle \text{QN}, *, *, *, *, * \rangle, \langle \text{E} \rangle \rangle. \tag{26}$$

Notice that the information which was specific to the current roster (i.e. in this example the day on which the violation took place) has been removed in the generalised version of the violation. The case-base is then searched for cases containing violations that match $vStructure_\alpha$. Suppose, for this example, that three such cases are found.

The distances between the focus violation and the violations stored in the retrieved cases are calculated. For this example the following violation features are used in the distance measure: number of violations, preference satisfaction, percentage of assigned hours, and violation magnitude. The retrieval process returns the rank of the three retrieved cases according to the distance measure.

3.3 Repair adaptation

The retrieval algorithm returns a ranked list of relevant cases, each of which contains information about a possible repair. In order to generate a repair for the focus violation these generalised repairs must be adapted to the context of the current roster. The goal of the repair adaptation method is to generate a new repair for the focus violation based on the information about previous repairs from the cases returned by the retrieval process.

This algorithm initially generates a set of *candidate repairs* from each of the cases in the trimmed case-base. The candidate repairs must be adapted to work in the context of the focus violation. In particular, it is ensured that any nurses specified in the focus violation are also included in the repairs. For example, the repairs generated for a Cover violation will always involve the day on which the violation occurs. Likewise, for a MaxDaysOn violation the generated repairs will always include the nurse who was working too many days in a row.

The candidate repairs cause new violations in the roster which can be very different. The violations caused by

Fig. 4 The adaptation algorithm

Given CB_{TRIM}—the set of retrieved cases, *vDistance* the set of distances between the cases and the focus violation, a search parameter k, $violation_\alpha$:

1: $Candidates \leftarrow \theta$
2: $i \leftarrow 0$
3: **while** $|Candidates| = 0$ **do**
4: $case_i = \langle v_i, r_i \rangle \leftarrow CB_{\text{TRIM}}[i]$, where $r_i = \langle rStruct_i, vSet_i \rangle$
5: $Candidates \leftarrow Candidates \cup$ GenerateCandidates $(violation_\alpha, rStruct_i)$
6: $i \leftarrow i + 1$.
7: **if** $i = k$ **then stop**
8: **end while**
9: **if** $|Candidates| = 0$ **then return** false
10: generate an array $rDistance[|Candidates|]$
11: **for all** $\langle repair_\beta, case_i \rangle \in Candidates$ **do**
12: generalise $repair_\beta$ using $\theta_v^{\mathcal{PS}}$
13: calculate distance between sets of violations $vSet_\beta$ caused by $repair_\beta$ and the set of violations $vSet_i$
14: **end for**
15: **return** *Candidates* sorted according to *rDistance*

each candidate repair must be compared with the violations that were caused by the generalised repairs it was generated from. The rationale behind this is that when a personnel manager is constructing a roster they generally choose repairs which cause minimal violations of other constraints. However, this will not always lead to overall *optimal* solutions and so repairs deemed by the manager to be acceptable but which cause resolvable damage should also be considered. In order to do this the candidate repairs are themselves generalised.

A comparison between these two sets of newly created violations takes into account that the number of constraint violations caused by two different repairs is likely to be different. Furthermore, even if the number of violations is the same, their types are very likely to be different.

The distance between two sets of violations, $d_r(vSet_i, vSet_\beta)$ is calculated by summing the distances between each of the violations in the first set from the whole of the second set. For each violation in the first set the distance is calculated as being maximal (i.e. equal to 1) if there are no violations in the second set with the same structural information. Otherwise, the distance for the violation is calculated as the distance from the matching violation in the second set. The number of violations in the second set which do not match any violation in the first set is added to the overall distance. The distance is then normalised by dividing by the total number of *comparisons* that have been made.

The overall distance measure is the product of the violation distance, calculated in the retrieval stage and stored in the *vDistance* array, and the repair distance, which is calculated as the distance between the two sets of violations, $vSet_i$ and $vSet_\beta$. The value 1 is added to the repair distance in order to ensure that two identical repairs of two violations of non-zero distance do not have a zero distance between them. However, two repairs generated for two identical violations should *always* have zero distance between them. The definition of this measure ensures the final repair rankings are determined by the distance between focus and retrieved repairs as well as the distances between retrieved and generated repairs.

The adaptation algorithm is presented in pseudo-code in Fig. 4. The input data are the cases retrieved during the retrieval process, CB_{TRIM}, the corresponding distances from the focus violation, *vDistance*, a search parameter, k, and the focus violation, $violation_\alpha$. In line 1 of the algorithm a set of candidate repairs, *Candidates*, is initialised as an empty set. Lines 2 to 8 describe the search through the retrieved cases for candidate repairs. The *GenerateCandidates* function in line 5 encapsulates the process of adapting repairs from the retrieved cases into repairs of $violation_\alpha$. For brevity, this process is not described here formally. The process creates a candidate repair for each possible combination of nurses and shifts which match the repair information stored in the retrieved case. The search parameter k sets the number of retrieved cases which are used to generate candidate repairs, starting with the case containing the violation most similar to $violation_\alpha$.

The adaptation process fails in line 9 if no candidate repairs can be generated. If candidate repairs are generated then these are generalised in line 12 and the distance between these generalised repairs and the repair stored in the case they were adapted from is calculated in line 13. The candidate repairs are finally ranked according to this distance measure, completing the repair adaptation.

To demonstrate the adaptation process we continue the example provided in Sect. 3.2. The retrieval process returned

three ranked cases. We assume for this example that the search parameter k is 1. Therefore, the adaptation process will consider only the first case returned by the retrieval process, $case_0$.

Suppose that this case contained a repair with the following structure:

$$rStructure_0 = \langle Reassign, \langle \langle \mathrm{RN}, \mathrm{F}, \mathrm{H}, \mathrm{ET}, \mathrm{E} \rangle, \mathbf{E}, \mathbf{U} \rangle \rangle \quad (27)$$

which describes a reassign repair where a registered nurse (RN), who was female (F), UK trained (H), specialty trained for the ward (ET), and grade (E) was assigned an early (E) shift where previously they had been unassigned (U) on the day which the repair was applied.

The repair information stored in $case_0$ must now be adapted to create a repair of the current violation. The adaptation process generates a set of "reassign" repairs using nurses with the correct feature set and who are currently assigned UNASSIGNED on day 3. In our roster (Table 3) we have two nurses with such characteristics ($nurse_0$ and $nurse_1$), and therefore two repairs are generated:

$$repair_a = \langle Reassign \langle nurse_0, 3, \mathbf{E} \rangle, ViolationSet_a \rangle, \quad (28)$$

$$repair_b = \langle Reassign \langle nurse_1, 3, \mathbf{E} \rangle, ViolationSet_b \rangle. \quad (29)$$

However, these candidate repairs cause different new violations in the roster. In this example $repair_a$ will case a 'SoftRequest' violation:

$$\langle SoftRequest \langle 3, nurse_0 \rangle \rangle \quad (30)$$

indicating that the shift preference of $nurse_0$ on day 3 will be violated.

The $repair_b$ does not cause any new violations in the roster. In order to decide which repair will be applied to the roster to fix $violation_\alpha$ the candidate repairs are generalised and then compared to the repair information stored in the case which they were generated from (in this example $case_0$). Assuming $case_0$ contained a repair which caused no new violations the most similar repair will be $repair_b$. This will then be applied to the roster, which will resolve the cover violation.

3.4 Case-base training

Training the case-base is not a trivial task. It is important that the case-base contains a range of different experiences representing a large number of different possible rostering scenarios. In particular, it is vital that there are cases in the case-base which represent good examples of behaviour in extreme circumstances (e.g. no free nurses being available on the day of a violation). With this in mind, we attempt in this section to give guidelines for the construction of a case-base.

Case-bases should be trained on real-world data rather than on artificially created problems. It is difficult to artificially create the complexity found in real problems. The real-world problems should include a wide range of different constraint violations. The user (i.e. personnel manager) should select a representative cross-section of these before training commences.

For each violation that the user chooses to include in the training the user must consider some key points:

1. The user must provide a repair to the given violation, making sure that they take into account the context that the violation occurs in. In particular, any new violations which are caused by the repair should be thought about in terms of the user's overall goals for the roster.
2. If other repairs could also be used to address the violation then they could also be added into the case-base. This provides the case-base with a range of experience. These similar cases which provide different options will be retrieved from the case-base according to the value of the search parameter k and then the adapted repairs will be chosen based on their similarity with respect to the new violations that they cause.
3. The user should consider what they would have done if circumstances had been slightly different. For example if no nurses on the day of the violation had unassigned shifts then a repair involving a nurse with an off-request, or who is already rostered, may need to be used. By manually altering the roster the user can generate an alternative scenario for which a significantly different repair should be used. By including these examples of similar violations which require different repairs the case-base is trained to recognise the *border* regions in the decision space.
4. It is useful to also provide repairs to any new violations created by the original repair. This addresses the problem of solving a series of violations.

It is difficult to set in advance the number of cases that should be stored in the case-base. One advantage of the CABAROST system is that it is transparent to the user. After representatives of each violation type have been stored in the case-base the algorithm provides suggested solutions to the user during training, enabling them to either accept the suggested solution or amend it if necessary. In this way the user can develop an idea of when the case-base has been adequately trained by monitoring the number of suggestions it made which they accepted without change.

Training of the case-base can also be considered as an on-going process. When used without user supervision the CABAROST system can detect if the distance between the generated repair and the retrieved one is above a certain threshold. This threshold can be set so as to represent the point at which CABAROST cannot be confident in the decisions it is making. By setting this threshold to a small value

the user can indicate that they want to supervise the decision making closely and change any repairs that they disagree with. When the user is confident that the decisions being made are correct then the threshold can be set at a high value; so that they are required to intervene in extreme circumstances only.

4 Hybridisation of genetic algorithm with CABAROST

The CABAROST system generates repairs for violations that appear in rosters. However, the order in which repairs are applied to the roster has a great effect on the quality of the final solution. A memetic algorithm, which is a hybridisation of a genetic algorithm and CBR, was developed to evolve good quality orderings of the repairs produced by CABAROST.

Genetic algorithms are optimisation tools based on the concepts of natural evolution. Genetic algorithms were first investigated as optimisation tools by Holland et al. in the 1970s (Holland 1975). They have since been used with some success to solve a number of different scheduling problems, including personnel scheduling problems (Aickelin and Dowsland 2000; Bailey et al. 1997). There is a large body of literature dedicated to the theory and practice of genetic algorithms. Readers are referred to (Goldberg 1989; Michalewicz and Fogel 2000) for details of their history and applications.

In a genetic algorithm a *population* of chromosomes is manipulated through a number of *generations* according to the principles of natural selection. The solutions are represented as chromosomes which can be combined to produce offspring through *crossover* operations. Crossover operations use information from two or more parent chromosomes to generate new chromosomes. Members of the population can be altered between generations by applying local *mutations* to chromosomes. Members of a current population are selected for crossover and mutation according to their fitness—a measure of the quality of the solution that they represent. Members with higher fitness are more likely to be selected than those with lower fitness and are therefore more likely to pass good solution information to the next generation.

Memetic algorithms combine the concept of biological evolution used by genetic algorithms with the optimisation power of local search algorithms. There are a number of different ways to incorporate local search algorithms into the genetic algorithm. Usually the members of a population are locally optimised between each generation. Consequently, the genetic operators act only on the locally optimal individuals generated by the local search thus theoretically reducing the overall size of the search space (Burke et al. 1995).

The pseudo-code of the memetic algorithm is given in Fig. 5. In the memetic algorithm that was developed a chromosome represents a sequence of the constraint violations, i.e. a sequence of the corresponding repairs suggested by CABAROST. Each chromosome is of variable length. Each repair sequence s_a has the following form:

$$s_a = repair_{a,1}, repair_{a,2}, \ldots, repair_{a,B_a}, \quad a = 1, \ldots, A, \tag{31}$$

where B_a is the length of sequence s_a, and A is the size of the population.

Each sequence is generated from the preference roster, from which violations are chosen at random and then repaired using CABAROST. The length of the sequence is randomly chosen from an interval which is a parameter of the search. Initially, a population of short (i.e. 5 to 10 repairs) sequences is created.

The fitness of a repair sequence is measured by summing the magnitudes of all the constraint violations that occur in a roster after the sequence has been applied and taking the reciprocal value. For example, the magnitude of a cover violation is equal to the shortfall (i.e. the number of extra nurses required to satisfy the constraint). The magnitude of MaxHours and MinHours violations is the number of hours over or under the maximum or minimum specified in the

1. Generate initial population of repair sequences (members)
2. Calculate fitness of each member by applying repair sequences to the preference roster
3. For each generation:
   (a) Create new population:
      i. Copy member with highest fitness to new population
      ii. Apply selection and crossover until new population size is reached
   (b) Remove repetitions in each repair sequence
   (c) Apply mutation operation to members of new population
   (d) Apply each member (repair sequence) to the starting roster
      i. Repair a random selection of violations (using CABAROST) and add the generated repairs to the repair sequence
      ii. Calculate the fitness of the repair sequence
   (e) If stopping condition reached then exit, else repeat

Fig. 5 Pseudo-code of the memetic algorithm

constraints, respectively. The constraint violations are not weighted in any way. The cases in the case-base reflect the relative importance of constraints. The nature of the repairs used for the different types of violations, and indeed the differing repairs used for subsets of violations of the same type, implicitly represent relative constraint importance.

At each generation the population is manipulated firstly by the *genetic* operators of selection, crossover, and mutation, followed by the *memetic* operation which adds additional repairs to each of the sequences by repairing more violations from the roster using CABAROST. The genetic operators work towards finding good quality permutations of the existing repair sequences, whilst the memetic stage increases the size of the sequences thus allowing repairs of more of the violations within the roster.

One-point crossover (Beasley et al. 1993) with a repetition filter was used to combine two parents. This operation chooses a position in each of two chromosomes at random and then creates two children by swapping the parent's values to the right of the selected positions and removing all repetitions of repairs. Given two children $\mathbf{x} = x_0, x_1, \ldots, x_{X-1}$ and $\mathbf{y} = y_0, y_1, \ldots, y_{Y-1}$, and crossover points $c_X \in [0, X-1]$ and $c_Y \in [0, Y-1]$ the two children produced by a crossover of $\mathbf{x}$ and $\mathbf{y}$ are:

$$\mathbf{x}' = x_1, \ldots, x_{c_X}, y_{c_Y+1}, \ldots, y_Y \quad \text{s.t. } y_i \neq x_j,\ j = 1, \ldots, c_X, \tag{32}$$

$$\mathbf{y}' = y_1, \ldots, y_{c_Y}, x_{c_X+1}, \ldots, x_X \quad \text{s.t. } x_i \neq y_j,\ j = 1, \ldots, c_Y. \tag{33}$$

The removal of repetitions from the repairs sequences is required in the crossover operation. Repeated copies of Reassign repairs will not have any effect on the roster and are redundant. When swap and switch repairs are repeated they 'undo' themselves (for example, a swap repair applied twice will result in unchanged assignments for the nurses involved). Consequently, if they remain in the repair sequence after crossover they will reverse the repairs of previous violations, thus increasing the total number of violations in the roster and therefore reducing the fitness of the chromosome.

Parents are selected for the crossover operator using a roulette wheel selection process (Michalewicz and Fogel 2000). This determines the probability of the selection of the parent based on its fitness. An elitist strategy is used, guaranteeing that the fittest member of the old population is copied into the new population. This strategy ensures that at least one member of the new population is of the highest quality found thus far.

There are two mutation operators that are applied to members of the new population. The delete mutation chooses a random number of repairs to delete from each sequence. Although the initial population contains sequences which only repair a small number of the initial violations, the sequences of repairs grow in order to solve all of the violations in the roster. The delete mutation helps to both limit the size and improve the quality of the sequences over time by 'undoing' repairs which do not contribute (or may even be detrimental) to the overall chromosome quality. The swap mutation changes the order of the repairs in a sequence by choosing a *swap point* and then swapping all of the repairs that were before this point to the end of the sequence. Each of the mutation operators is applied to sequences with a 0.1 probability.

After the genetic operators have been applied to the chromosomes in a population the repair sequences encoded in the chromosomes are successively applied to the roster. After that a random selection of the remaining violations are repaired. Some of these may be new violations which have occurred as a result of the performed repairs.

4.1 Learning from failure

An important property of CBR systems is learning capability. Learning is driven not only by successfully repaired violations but also by repairs that are considered to be failures. A CBR system which learns from failure should improve over time by reducing the likelihood that cases with high failure rates are used to solve new problems.

In this problem, a failure is defined as the reappearance of a violation after it has already been repaired using a case from the case-base. This is assessed each time a repair is added to each repair sequence in the population. A record is kept of the violation that was solved by each repair in the repair sequence. When a violation is selected for repair the record is checked and if it has previously been solved then the case which was used to generate the repair is deemed to have failed.

The failure of a case is registered in the case-base by increasing the overall case weighting (W_γ) used in (24). This increase in weight acts as a penalty on the case. The effect of an increase in the weighting of a case is to increase its distance from any future violations; thus, reducing the chance that it will be used to solve future violations. This increase in weight also means that for the case to be selected in future, the focus violation would have to be even more similar to the stored violation than previously. In other words, as its weight increases the case is more and more likely to be used only for violations which are very similar. In the extreme, when a significantly large weight is applied to a case, it may be that only an almost 'identical' violation (in terms of the feature values) would lead to the case being retrieved, i.e. the case's applicability is narrowed to a small range of problems.

Given this definition of failure, it is likely that the majority of cases in the case-base will fail at some point due to

the highly constrained nature of rostering problems. Consequently, it is important that the increase in weight upon failure is small so that cases with a low rate of failure are still used to repair violations. The weighting is a gradual process which will penalise most the cases which consistently fail.

5 System evaluation

A set of experiments was designed to: (a) evaluate the performance of CABAROST in the repair of constraint violations; (b) evaluate the performance of CABAROST as a repair generation tool within the memetic algorithm; (c) investigate the effects of ongoing training of the case-base throughout its use; and (d) investigate the use of case weighting as a means by which to penalise failure.

The case-base was trained following the guidelines described in Sect. 3.4 using violations and repairs of one month of rostering data (MARCH2001) from the QMC hospital. In this problem instance 20 nurses were rostered over a four week period subject to the constraints given in Table 4. For example, the first constraint in the list is a cover constraint which requires that at least 4 qualified nurses are assigned the early shift every day.

The preference shifts requested by the nurses over this period are presented in the preference roster in Fig. 6. In the left-most column the nurse's name (anonymised) is followed by an indication of their qualifications and experience (using the abbreviations described in Sect. 2.1). Each column is headed by the date and day it represents, and these are divided into weekly sections (running Monday to Sunday). The "AL" shift type denotes annual leave. Nurses with AL indicated in their roster for a particular day have arranged this statutory time off prior to the rostering process, and this will not be altered. This preference roster contains 123 constraint violations.

The case-base contained 189 cases, with cases containing mix of violations listed in Table 5. The number of cases stored for each violation reflects the variety of different repairs that can be made. For example, HardRequest violations (when a nurse's preferred shift is not assigned) are always repaired by swapping the affected nurse's assigned shift with a nurse of any other type who happens to be currently assigned the requested shift. The number of cases stored containing repairs of this violation type is low because this repair behaviour takes place in all circumstances. Cover violations, however, require different repairs to be used in different circumstances, and therefore a large number of contrasting repairs are used in the case-base.

In the experiments described in this section the memetic algorithm is applied to the five new problem instances from the QMC (April 2001 to August 2001). A number of variations of the algorithm were investigated. Each variation of the algorithm was run 20 times for each problem instance. All of the roster quality results (i.e. fitness values) presented are averages (and standard deviations) of the best solution found during each of the 20 runs.

Table 4 Constraints for the QMC problem

Constraint type	Description
Cover	EARLY shifts require 4 Qualified Nurses
Cover	EARLY shifts require 1 Registered Nurse
Cover	EARLY shifts require 1 Eye-Trained Nurse
Cover	EARLY shifts require 1 Auxiliary Nurse
Cover	LATE shifts require 3 Qualified Nurses
Cover	LATE shifts require 1 Registered Nurse
Cover	LATE shifts require 1 Eye-Trained Nurse
Cover	LATE shifts require 1 Auxiliary Nurse
Cover	NIGHT shifts require 2 Qualified Nurses
Cover	NIGHT shifts require 1 Eye-Trained Nurse
Cover	NIGHT shifts require 1 Auxiliary Nurse
MaxDaysOn	Maximum number of consecutive shifts is 6
MinDaysOn	Minimum number of consecutive shifts is 2
SingleNight	nurses can not work only one night shift
Succession	An EARLY shift must not follow a NIGHT shift
TotalsMax	Maximum number of hours in a fortnight (14 days) is 75
TotalsMax	Maximum number of hours is as set in nurse contracts
TotalsMin	Minimum number of hours is 5 less than the contract maximum
WeekendBalance	No more than 3 weekends out of every 4
WeekendsInARow	No more than 3 weekends in a row
WeekendSplit	Either both days of a weekend or not at all

The first experiment was designed and carried out to determine the quality of the reasoning process in terms of agreement between automated decisions and those of the personnel manager. Three "verdicts" were defined: exact match when the generated repair was identical to the manager's repair, equivalent match when the generated repair involved nurses of the same type and the same shifts as those used in the manager's repair, and fail otherwise. The system started with an empty case base and stored new cases over time. For each constraint violation in the roster the system suggested three repairs. Figure 7 shows an average cumulative of exact and equivalent matched repairs against the case-base size for each of the three suggested repairs. It can be seen that the case-base learns how to produce more exact or equivalent repairs as its size increases. An increase in the amount of training given to the case-base leads to an increase in the quality of the repairs produced. It is particularly encouraging that the first suggested repairs, in general, score more exact and equivalent matches than the second and third. The increase in solution quality is made more

	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1
	M	T	W	Th	F	S	Su	M	T	W	Th	F	S	Su	M	T	W	Th	F	S	Su	M	T	W	Th	F	S	Su
Nurse 0 (RN,ET)	L	E	E	E	E	O	O	N	N	O	E	AL	O	O	L	E	O	O	L	L	L	E	E	E	O	O	E	L
Nurse 1 (EN, NT)			E	O	O	L	E				E	E	O	O	AL	AL	AL	AL	AL	AL	AL	AL	AL	L	E	N	N	O
Nurse 2 (RN, NT)	O	L	L	L	E	O	O	L	E	L	O	O	O	L	E	L	N	N	L	O	O	O						
Nurse 3 (EN, ET)						O	O		O				O	O														
Nurse 4 (RN, ET)	O	O		AL						E	AL	AL			L													
Nurse 5 (RN, ET)		E	L	E	E		N		L	L	AL	E	L		O	O	O	O	O	O	O	O	O	O	O	O	O	O
Nurse 6 (RN, ET)	L																							N	N			
Nurse 7 (RN, ET)	E	O	E	O	N	N	E	O	O	O	O	L	O	E	E	O	O	O	L	O	E	E	O	O	O	L	O	E
Nurse 8 (EN, ET)															AL	AL	AL	AL	AL							O		
Nurse 9 (RN, NT)	O	O	O	O	O	O	L	E	L	O	O	O	O	O	N	N	O	O	O	O	O	O	O	O	O	O	O	O
Nurse 10 (RN, ET)	O	O	O	E	L	L	O	L	O	L	AL	AL	AL	N	O	O	L	O	L	L	AL	AL	AL	AL	AL	AL	AL	AL	AL
Nurse 11 (RN, ET)	O	L	E	E	O	O	O	AL	AL	AL	O	O	O	O	AL	O	O	O	E	E	E	N	N	O	O	O	O	O
Nurse 12 (RN, ET)	E	E	N	N	O	O	E	E	L	E	L	E	O	O	E	E	E	E	E	O	O	L	E	E	L	L	O	O
Nurse 13 (RN, NT)	L	E	L	L	E	O	O	O	L	O	E	L	L	E	L	L	L	L	O	O	E	L	L	AL	AL	AL	O	O
Nurse 14 (RN, ET)	O	O	AL	E	L	O	L	E	E	O	E	E	O	O	E	O	E	E	O	O	L	E	O	E	E	E	O	O
Nurse 15 (RN, ET)	O	E	L	E	O	O	L	O	O	N	N	O	O	O	O	E	L	E	O	O	L	O	E	E	AL	O	O	O
Nurse 16 (RN, ET)	N	N	O	O	O	L	L	L	E	L	E	O	O	O	L	E	L	L	O	O	O	L	E	O	O	O	L	L
Nurse 17 (RN, ET)		AL	L			O	O		AL		AL				AL	AL	AL	AL	AL	AL	AL	L					O	O
Nurse 18 (RN, NT)												E	O	O	L	AL	AL	AL	AL	AL	AL	AL						

Fig. 6 Preference roster for March 2001

Table 5 Initial case-base contents

Violation type	Number
Cover	54
HardRequest	12
MaxDaysOn	7
MaxHours	13
MinDaysOn	20
MinHours	46
SingleNights	3
SoftRequest	14
Succession	4
WeekendBalance	5
WeekendsInARow	6
WeekendSplit	5

apparent in Fig. 8. This shows the percentage of exact and equivalent repairs at different stages in the run, with an increasing number of cases in the case-base. In general, in the later stages, when the case-base contains more experience, a large number of good suggestions are produced, demonstrating the learning capabilities of the system.

The second experiment compares the results achieved by using the memetic algorithm with the CABAROST generated repairs, and with repairs that have been created at random. Table 6 presents the values of the parameters used in the memetic algorithm. The random repair generation uses the basic parameter information for the focus violation and creates a repair of random type, with random nurses and shifts involved. By generating the repairs randomly the memetic algorithm is performing a search with no knowledge coming from the case-base. In this experiment the case-base trained using the MARCH2001 data was used with no additional cases added.

Table 7 shows the results for the experiment. The second column shows the starting quality of the preference roster of each month. In the third column the results (mean and standard deviation) of the memetic algorithm with random repair generation are given for each month. The final column shows the results of the memetic algorithm with CABAROST generated repairs. The results show that the knowledge contained in the case-base significantly improves the performance of the memetic algorithm. The quality of the solutions produced is better using CABAROST repairs for all five problem instances. The CABAROST repair generation also leads to more consistent solutions. This is

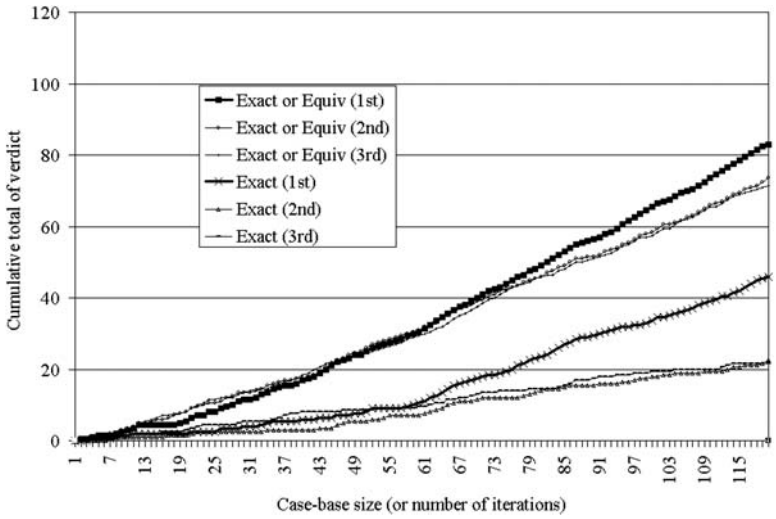

Fig. 7 Learning of the system over time

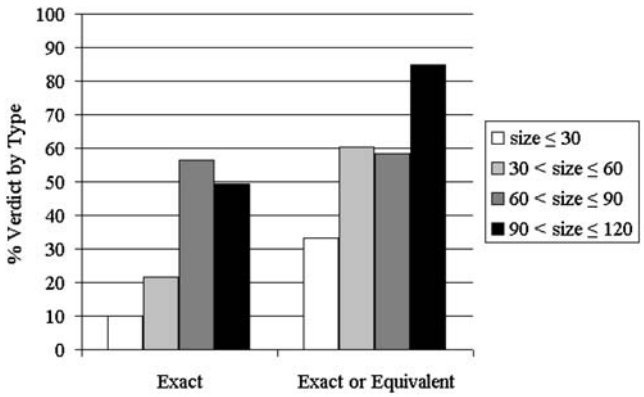

Fig. 8 Effects of case-base size on solution quality

Table 6 Parameters of the memetic algorithm

Parameter	Value
Crossover probability	0.8
Mutation probability	0.1
Number of repairs deleted by mutation	Between 1 and 5*
Population size	100
Initial chromosome length	Between 5 and 10*
Stopping condition	No improvement for 10 generations

*Chosen randomly

demonstrated by the lower standard deviations of the results of the twenty runs.

In the third experiment the effect of on-going training on the quality of solutions is assessed. On-going training of the case-base is performed by allowing the user to interactively participate in the search process. Whenever a repair is generated for a violation the repair distance is measured and this can be interpreted as a measure of confidence in the quality of the repair. If this repair distance is larger than a certain threshold value then the user is asked to confirm the repair. Including a configurable threshold allows the user to choose the level of control they have over the generation of repairs—to a greater extent (for a low threshold value), to a lesser extent (for a higher threshold values), or not at all if an 'infinite' threshold was used causing all repairs to be accepted (i.e. a case-base with no on-going training). When repairs exceed the threshold the user can accept the generated repair or improve by altering the repair parameters (e.g. by choosing a different nurse) it if it is not satisfactory. The new violation/repair (case) is added to the case-base, thus increasing the experience it contains.

Table 7 Roster quality—CABAROST vs. random repair generation

Month	Initial	Random	CABAROST
April 2001	545.00	197.30 (21.67)	161.34 (7.01)
May 2001	592.75	224.35 (38.07)	198.01 (10.49)
June 2001	698.25	243.84 (31.41)	229.73 (23.29)
July 2001	530.00	271.14 (41.69)	233.15 (17.36)
August 2001	675.25	234.51 (36.91)	189.04 (24.29)

The memetic algorithm is used to solve each of the problem instances in turn. Any new cases added to the case-base during the run of the memetic algorithm are kept and used in the case-base for the following month. The algorithm is used with repair confidence thresholds of 1.5 and 2.5, and also with a static case-base where all repairs retrieved from the case-base are accepted and no new cases are added.

Table 8 shows the average results over the 20 runs of the algorithms. The second column shows the mean and standard deviations of the results achieved by the memetic algorithm using CABAROST and the static case-base. The third and fourth columns show the results using on-going training

Table 8 Roster quality: ongoing training

Month	Static case-base	Threshold = 1.5	Threshold = 2.5
April 2001	161.34 (7.01)	**145.63** (9.64)	155.15 (11.28)
May 2001	198.01 (10.49)	**161.18** (12.64)	182.15 (13.16)
June 2001	229.73 (23.29)	**206.73** (10.17)	210.38 (10.85)
July 2001	233.15 (17.36)	**187.86** (17.02)	223.97 (16.30)
August 2001	189.04 (24.29)	**181.15** (8.38)	186.54 (11.23)

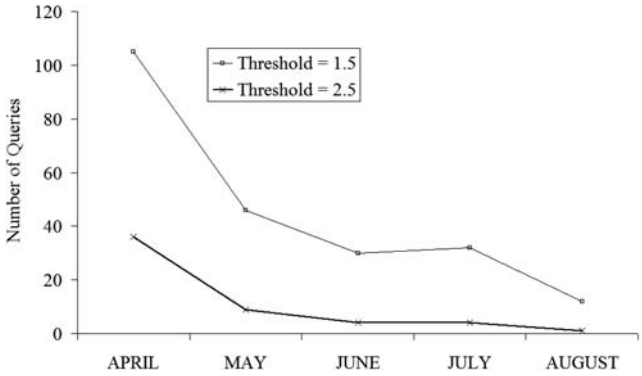

Fig. 9 On-going training: number of queries (1.5 and 2.5 thresholds)

with repair confidence thresholds set at 1.5 and 2.5, respectively.

It is evident that performing constant training of the case-base gives a considerable advantage. For most problems the performance of the algorithm improved significantly when it used the 1.5 acceptance threshold. The performance was better for the 1.5 acceptance threshold than it was for 2.5, because of the increased feedback from the user. The experience that is added to the case-base gained on one months problem also improves the performance of the memetic algorithm for new problems.

The number of times that the user is asked for feedback about repairs was also investigated because an algorithm which requires consistently high levels of user interaction may become tedious to use. Figure 9 shows the sharp decrease in the average number of times that the user is asked for input for each problem. For the APRIL2001 problem an average of 105.2 *queries* were presented to the user when the threshold was set at 1.5. This had reduced to just 12 for the AUGUST2001 problem, 5 months later. The figure also shows the difference in number of queries when different thresholds are used. By allowing the user to choose the threshold they can specify the amount of input they want to have depending on how much time is available for supervising the algorithm. This reduction in the number of queries provides further evidence that the case-base is learning from this supervised approach.

The fourth experiment is used to show that the penalising of cases which perform badly in the search by using case weights increases the overall performance of the memetic algorithm. The increase of the case weight when the case has failed is set at 0.1, 0.5, and 1.

Table 9 shows the results for the different case weighting levels. The second column shows the results for the memetic algorithm with CABAROST generated repairs and no case weighting. The third, fourth, and fifth columns show the results for increments of 0.1, 0.5, and 1.0, respectively.

The results show that the weighting of failed cases increases the overall performance of the memetic algorithm. The increment of 1 will not lead to good quality rosters because its effect on the case-base is too severe. The weight increment 0.1 performed best on average for all but one of the months data. An increment of 0.5 shows a small improvement on the un-weighted algorithm, except for the JULY2001 problem instance.

Overall, the results show that the knowledge that can be captured and used through case-based repair generation and the search capabilities of genetic algorithms can be successfully hybridised in a memetic algorithm. This algorithm is capable of generating good quality sequences of repairs based on the knowledge and experience held in the case-base.

6 Discussion

The development of successful personnel scheduling methods is highly dependent on the correct representation of domain knowledge. Many of the early mathematical models were not flexible because of the rigid structure that they imposed on the problem representation. Although there has been a lot of research effort in the development of artificial intelligence techniques for solving scheduling problems in general, and personnel scheduling in particular, the development of knowledge-based scheduling systems, which attempt to emulate the methods of the human schedulers, is recognised to be a difficult task (Kempf et al. 1991). It is an inexact and time-consuming process, and can also lead to the development of inflexible and incomplete domain models (Sqalli and Freuder 1998). In view of the complexity of real-world scheduling problems it is very difficult to define a list of "IF–THEN" rules that describe the reasoning of scheduling managers. The nature of the human scheduling process is based on specific experience rather than on a set of general guidelines or first principles.

Meta-heuristic methods that have been often used for personnel scheduling also suffer from inflexibility. The main criticism is that the researchers do little to generalise their methodologies beyond the particular problem instance they were attempting to solve. Their drawback is that they usually work well only in environments that are very similar to the instance of the problem for which they were designed.

Table 9 Roster quality: case weighting on failure

Month	No weighting	Increment = 0.1	Increment = 0.5	Increment = 1.0
April 2001	161.34 (7.01)	**154.95** (6.27)	159.74 (6.86)	179.01 (24.65)
May 2001	198.01 (10.49)	**185.73** (14.93)	195.63 (14.24)	216.48 (34.80)
June 2001	229.73 (23.29)	213.15 (5.45)	**199.86** (7.40)	234.86 (30.27)
July 2001	233.15 (17.36)	**227.99** (13.22)	234.14 (16.30)	251.17 (34.92)
August 2001	189.04 (24.29)	**176.75** (15.33)	188.53 (26.84)	216.35 (42.23)

However, the definitions of staff skill, shift types, planning periods, and of the myriad constraints and requirements vary considerably both between and within organisations. Methods suitable for one problem are not easily transferred to other problems. Each new instance of the problem requires significant changes to the model, such as changes to the objective function which depends heavily on the researcher's a prior understanding of the problem, changes to the neighbourhood structure in local search methods, etc. Frequently the unsatisfactory approach of 'changing the problem to suit the method' is employed which generates poor results.

The personnel scheduling problem can be formulated as a constraint optimisation problem and constraint programming methods have been developed with some success (Smith and Bennett 1992). However, the application of constraint programming requires the specification of the order of assignment of values to variables and also the procedure for constraint propagation. This depends heavily on the problem instance data and does not exploit the experience of the personnel managers in the schedule construction.

Eliciting problem *solving* knowledge from personnel managers is an open issue in personnel scheduling. Translation of operation information into mathematical models is not an easy task. The current successful methods for personnel scheduling rely on weighting constraints or patterns, and prioritisation of constraint types for capturing some of the knowledge about the features that the personnel managers would like to see in their schedules, without providing a means to acquire information about how this should be achieved. In addition, many of the decisions made by personnel managers are of a personal, subjective nature, and are therefore difficult to model explicitly. This is particular true when considering the treatment of staff requests and preferences, which may need to be treated on a case by case basis. This paper addressed this problem using CBR which allows users to specify how they would like problems to be solved.

The approach described in this paper have been designed to be applicable to a wide variety of scheduling and other constraint optimisation problems. In particular, it is applicable to most of the nurse rostering problems that can be found in the literature. In many other problem domains the relative importance of constraints is dependent on the opinion of individual personnel managers. CBR could be used to collect knowledge about how personnel managers repair constraint violations for such problems as scheduling, planning, and design. To facilitate the development of such methods the following key objects and processes must be defined:

- *Constraints/Violations*: *Structural* characteristics of both constraints and their violations have to be defined including information such as the type of constraint, the type of domain objects involved (e.g. jobs, resources, routes), and the requirements dictated by the constraint in terms of those objects.
- *Repair Operations*: The operations used to repair constraint violations must be defined. These may be elicited from domain experts by determining how manual changes to solutions are made. Alternatively, they may be determined through analysis of the mathematical characteristics of the problem. Again, the structural features of these repairs must be determined in terms of the domain objects.
- *Generalisation*: Key to the success of the case-based reasoning system developed is the translation of domain object characteristics into objects suitable for storing in the case-base. One principle must be adhered to when designing object generalisations, namely that no representation of a case should be applicable to only one instance of a problem. Objects must be abstracted out of their particular problem instance by, for example, removing resource identification labels.
- *Features*: After the structural characteristics of domain objects have been identified the set of features must be established for each type of constraint violation. Initially, a large set of features could be identified through analysis of the problem and interviews with personnel managers. A method has been developed for refining feature selection by automatically selecting and weighting features, increasing the classification accuracy of the case-base and reducing the storage requirements (Beddoe and Petrovic 2006).
- *Repair Adaptation Rules*: The cases retrieved from the case-base must be adapted to the context of the problem instance being solved. In particular, the generalised objects used to describe violations and repairs in the case-base must be combined with the information from the violation being solved in a sensible way. Adaptation rules must be designed so that they produce repairs that will have a measurable effect on the constraint violation in the problem instance, but they must not be too restrictive in their interpretation of the cases.

A case-based method which addresses these five issues could use the retrieval and adaptation algorithms described in this paper with very few alterations. The development of such methods should be the focus of future research into the application of case-based reasoning to personnel scheduling.

7 Conclusion

This paper has introduced the CABAROST system for personnel scheduling. The personnel scheduling problem is represented in terms of the constraints that should be satisfied. The approach is demonstrated on a real-world nurse rostering problem which is a highly constrained optimisation problem with a wide variety of constraints. CABAROST captures nurse rostering experience from personnel managers by storing examples of their repairs of constraint violations. The method developed is flexible because it allows different managers to train the case-base independently, thus providing a tool which is not bound by the understanding and bias of the algorithm developers.

The issues of representing rostering knowledge within the case-base have been addressed. Violations and the repairs used to solve them are stored in cases in a *generalised* form. This generalisation of domain objects ensures that cases are applicable to a wide range of possible future problems, even involving rosters for completely different sets of nurses.

CABAROST can be used to iteratively repair constraint violations in rosters. However, the order of the repairs significantly affects the final roster quality. This paper has introduced a memetic algorithm which searches for optimal sequences of repairs generated by the CBR system. This memetic algorithm differs from others in the literature through its use of CABAROST in place of the local search that would usually be applied to populations between generations. The results of experiments on the data from the QMC indicate that this hybridisation provides an excellent tool for solving nurse rostering problems. The quality and coverage of the case-base is maintained through a combination of ongoing training controlled by a case acceptance threshold and case weighting.

Future research work will be carried out to determine the quality of cases in the case-base and on more general maintenance issues including limiting case-base size and increasing the *coverage* of the case-base (Smyth and McKenna 1998). In particular, it should be possible to detect when the *style* of rostering decision making changes so that the case-base can be adapted in response. For example, personnel managers may decide to violate more of the nurses' shift preferences due to an increasing patient load, and the decisions made by CABAROST should reflect this.

Acknowledgements We acknowledge the support given by the Engineering and Physical Sciences Research Council (EPSRC) in the UK (grant number GR/N35205/01) and by the Queen's Medical Centre University Hospital Trust, Nottingham, for this research.

References

Aickelin, U., & Dowsland, K. A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, *3*(3), 139–153.

Aickelin, U., & Li, J. (2007). An estimation of distribution algorithm for nurse scheduling. *Annals of Operations Research*, *155*(1), 289–309.

Azaiez, M. N., & Sharif, S. S. A. (2005). A 0-1 goal programming model for nurse scheduling. *Computers & Operations Research*, *32*(3), 451–507.

Bailey, R. N., Garner, K. M., & Hobbs, M. F. (1997). Using simulated annealing and genetic algorithms to solve staff scheduling problems. *Asia-Pacific Journal of Operational Research*, *14*, 27–43.

Beasley, D., Bull, D., & Martin, R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, *15*(2), 58–69.

Beddoe, G., & Petrovic, S. (2006). Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *European Journal of Operational Research*, *175*(2), 649–671.

Bester, M. J., Nieuwoudt, I., & Van Vuuren, J. H. (2007). Finding good nurse duty schedules: a case study. *Journal of Scheduling*, *10*(6), 387–405.

Burke, E. K., Cowling, P. I., De Causmaecker, P., & Vanden Berghe, G. (2001). A memetic approach to the nurse rostering problem. *Applied Intelligence*, *15*(3), 199–214.

Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B. (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*. doi:10.1016/j.ejor.2007.04.030.

Burke, E. K., De Causmaecker, P., & Vanden Berghe, G. (1999). A hybrid tabu search algorithm for the nurse rostering problem. In B. McKay, X. Yao, C. Newton, J. Kim, & T. Furuhashi (Eds.), *Lecture notes in artificial intelligence: Vol. 1585. Simulated evolution and learning* (pp. 187–194). Berlin: Springer.

Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, *7*(6), 441–499.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, *9*(6), 451–470.

Burke, E. K., MacCarthy, B., Petrovic, S., & Qu, R. (2000). Structured cases in case-based reasoning—re-using and adapting cases for time-tabling problems. *Knowledge-Based Systems*, *13*, 159–165.

Burke, E. K., Newall, J. P., & Weare, R. F. (1995). A memetic algorithm for university exam timetabling. In *1st international conference on the practice and theory of automated timetabling (ICPTAT'95)*, Napier University, Edinburgh, UK, 30th Aug–1st Sept. 1995 (pp. 496–503).

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, *9*(2), 115–132.

Chiaramonte, M. V., & Chiaramonte, L. M. (2008). An agent-based nurse rostering system under minimal staffing conditions. *International Journal of Production Economics*. doi:10.1016/j.ijpe.2008.03.004.

Cunningham, P., & Smyth, B. (1997). Case-based reasoning in scheduling: reusing solution components. *International Journal of Production Research*, *35*(11), 2947–2961.

Dowsland, K. (1998). Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, *106*, 393–407.

Goldberg, D. (1989). *Genetic algorithms in search, optimisation, and machine learning*. Reading: Addison-Wesley.

Holland, J. (1975). *Adaption in natural and artificial systems*. Cambridge: MIT Press.

Kempf, K., LePape, C., Smith, C., & Fox, B. R. (1991). Issues in the design of AI-based schedulers: workshop report. *Artificial Intelligence Magazine*, *11*, 37–46.

Kolodner, J. L. (1993). *Case-based reasoning*. San Mateo: Morgan Kaufmann.

Legato, P., & Monaco, M. F. (2004). Human resources management at a marine container terminal. *European Journal of Operational Research*, *156*, 769–781.

Li, J., & Aickelin, U. (2003). A Bayesian optimization algorithm for the nurse scheduling problem. In *2003 congress on evolutionary computation* (pp. 2149–2156).

MacCarthy, B. L., & Jou, P. (1995). A case-based expert system for scheduling problems with sequence dependent set up times. In *Applications of artificial intelligence in engineering* (pp. 89–96).

Meisels, A., & Lusternik, N. (1998). Experiments on networks of employee timetabling problems. In E. Burke & M. Carter (Eds.), *Lecture notes in computer science: Vol. 1408. Practice and theory of automated timetabling: selected papers from PATAT 1997* (pp. 130–141). Berlin: Springer.

Meyer auf'm Hofe, H. (2000). Solving rostering tasks as constraint optimization. In *Lecture notes on computer science. Selected papers from the 3rd international conference on practice and theory of automated timetabling (PATAT)* (pp. 280–297). Berlin: Springer.

Michalewicz, Z., & Fogel, D. B. (2000). *How to solve it: modern heuristics*. Berlin: Springer.

Miller, H. E., Pierskalla, W. P., & Rath, G. J. (1976). Nurse scheduling using mathematical programming. *Operations Research*, *24*(5), 857–870.

Miyashita, K., & Sycara, K. (1995). CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, *76*, 377–426.

Petrovic, S., Yang, Y., & Dror, M. (2005). Case-based initialisation for examination timetabling. In G. Kendall, E. Burke, S. Petrovic, & M. Gendreau (Eds.), *Multidisciplinary scheduling: theory and applications* (pp. 289–308). Berlin: Springer.

Sadeh, N., Sycara, K., & Xiong, Y. (1995). Backtracking techniques for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, *76*, 455–480.

Schmidt, G. (1998). Case-based reasoning for production scheduling. *International Journal of Production Economics*, *56–57*, 537–546.

Scott, S., & Simpson, R. (1998). Case-bases incorporating scheduling constraint dimensions—experiences in nurse rostering. In *Lecture notes in artificial intelligence. Advances in case-based reasoning—EWCBR98* (pp. 392–401). Berlin: Springer.

Smith, B. M., & Bennett, S. (1992). Combining constraint satisfaction and local improvement algorithms to construct anaesthetists' rotas. In *Proceedings of the conference on artificial intelligence applications (CAIA92)* (pp. 106–112).

Smyth, B., & McKenna, E. (1998). Modelling the competence of case-bases. In *Proceedings of the European workshop on case based reasoning*, Dublin, Ireland (pp. 208–220). Berlin: Springer.

Sqalli, M. H., & Freuder, E. C. (1998). Integration of CSP and CBR to compensate for incompleteness and incorrectness of models. In *Proceedings of the AAAI-98 spring symposium on multimodal reasoning*, Stanford University.

Watson, I. (1999). Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems*, *12*, 303–308.

Watson, I., & Marir, F. (1994). Case-based reasoning: A review. *The Knowledge Engineering Review*, *9*(2), 327–354.

Wren, A. (1996). Scheduling, timetabling and rostering—a special relationship? In E. Burke & P. Ross (Eds.), *Lecture notes in computer science: Vol. 1153. Practice and theory of automated timetabling* (pp. 46–75). Berlin: Springer.