Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 667–691
computers & operations research
www.elsevier.com/locate/cor

A genetic algorithm approach to a nurse rerostering problem

Margarida Moz[a,b,*], Margarida Vaz Pato[a,b]

[a] *Departamento de Matemática, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Rua do Quelhas, 6, 1200-781 Lisboa, Portugal*
[b] *Centro de Investigação Operacional, Universidade de Lisboa, Portugal*

Available online 22 April 2005

Abstract

The nurse rerostering problem occurs when one or more nurses cannot work in shifts that were previously assigned to her or them. If no pool of reserve nurses exists to replace those absent, then the current roster must be rebuilt. This new roster must comply with the labour rules and institutional constraints. Moreover, it must be as similar as possible to the current one. The present paper describes constructive heuristics, besides several versions of genetic algorithms based on specific encoding and operators for sequencing problems applied to the nurse rerostering problem, defined with hard constraints. In the genetic algorithms described, each individual in the population is associated with a pair of chromosomes, representing permutations of tasks and nurses. Those permutations are used as input to a procedure that generates rosters. The fitness of individuals is given by the similarity between the roster generated from the permutations and the current one. The authors developed several versions of the genetic algorithm, whose difference lay in the encoding of permutations and in the genetic operators used for each encoding. These heuristics were tested with real data from a Lisbon hospital and yielded good quality solutions.

Scope and purpose The research reported is part of a project designed to develop a system for the management of nurse schedules for implementation in Portuguese public hospitals. The specific problem of rebuilding nurse schedules is addressed when unexpected staff absences arise. The complexity of the problem led the authors to design heuristic procedures. The tests performed so far with real data have shown that the algorithms attain good quality solutions at a computing time within the bounds stipulated by the hospital.

Keywords: Nurse scheduling; Rerostering; Heuristics; Genetic algorithms; OR applications in health care

* Corresponding author. Departamento de Matemática, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Rua do Quelhas, 6, 1200-781 Lisboa, Portugal. Fax: +351 21 3922781.
E-mail address: mmoz@iseg.utl.pt (M. Moz).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.03.019

1. Introduction

In hospital units that operate 24 hours a day, work is organised into shifts. Periodically, a plan of work for the next working period—a roster—is announced by the head nurse. The problem of nurse rerostering occurs when at least one nurse notifies that she is unable to perform one or more shifts of her current schedule. If there is no reserve pool of nurses to replace those absent, the problem is solved by rebuilding the current roster, thus altering the schedules of other nurses. The altering of previously announced schedules is a sensitive issue. Since nurses tend to organise their private lives in accordance with their expected duties, any change in the announced schedules may create personal inconveniences to some of them. Building the new roster gives rise to a new problem—the rerostering problem—where the objective is to determine a new feasible roster that minimises the number of shift changes with regard to the current one.

Although staff scheduling and rostering problems are widely addressed in the literature, as shown in the bibliographic surveys by Baker [1], Tien and Kamiyama [2], Siferd and Benton [3], Wren [4] and, more recently, by Cheang et al. [5], the rerostering problem has, to the authors' knowledge, received much less attention. Warner [6], in addressing a nurse scheduling problem, refers to the existence of unexpected absences, but assumes there is always a reserve pool of nurses to replace those absent. Weil et al. [7] identify the problem in a French hospital and suggest a constraint programming approach that successively generates feasible rosters. However, they do not optimise any criterion, thus leaving the selection of a solution to a decision-maker.

In a different, albeit related context, the paper of Cumming et al. [8] mentions the problem of adjusting school timetables in response to unexpected events, while, as far as possible, preserving the former structure.

In previous approaches to the rerostering problem occurring in certain units of a Lisbon public hospital, the authors presented a constructive heuristic and a number of ILP formulations, along with results obtained from a computational experiment (see [9,10]).

Despite the good results obtained, in terms of computational time and solution quality, in solving one of the ILP models with standard optimisation software, further research was devoted to the development of heuristics. Two main reasons justify the use of heuristics in this context: (a) since the problem is NP-hard, as proved in [11] (see Appendix A), particularly high dimensioned instances may occur where exact methods would break down; (b) as heuristic approaches are more flexible, they enable one to more easily handle specific features to be found in real cases, besides sporadic alterations in these features.

The present paper reports on this research into heuristics designed to be incorporated in a computational system for nurse rostering and rerostering in Portuguese hospital units. It presents an improved version of the constructive heuristic in [12], as well as several versions of a genetic algorithm (GA). The constructive heuristics, based on the sequential assignment of tasks to nurses from an input comprising pairs of permutations for nurses and tasks, have pointed to the possibility of using a GA to search the two spaces of permutations. Each pair of permutations to be encoded in a pair of chromosomes would then be used as input for a procedure that would generate the solutions to the original problem. One such indirect solution approach with precedents in Davis [13] was also recently used in a nurse scheduling problem by Aickelin and Dowsland [14]. While sharing the indirect solution idea, the GA described here, and developed in the context of [11], not only differs from the one mentioned above concerning the problem addressed, but also differs in the coding of the two spaces of permutations through pairs of chromosomes, and in the decoder that transforms each pair of chromosomes into a solution to the problem.

In Section 2 of this paper, the authors describe the nurse rerostering problem defined with hard constraints in relation to the particular real situation addressed. Section 3 is devoted to the constructive heuristics. Section 4 presents the GA, and Section 5 introduces different versions of this GA, with specific encoding and operators for sequencing problems. Section 6 reports on the results obtained in computational experiments with all versions of the heuristics. The instances tested were generated using data from two units of the same hospital, and statistical tests were performed to compare the quality of the heuristics' solutions. Finally, some conclusions are outlined in Section 7.

2. Description of the problem

The hospital in question operates 24 hours a day. Some units organise nurses' work into three eight-hour shifts—the day, evening and night shift, whereas others offer more flexible starting times for the nurses' daily tasks. Health care is provided by a fixed group of nurses in each unit. Every month each unit's head nurse notifies each nurse of her schedule for the next 28-day planning period. A nurse schedule is made up of sequences of shifts—one shift per day, where one task is assigned to that nurse—and days off. The present paper is based on the data concerning two hospital units, which work in three eight-hour shifts, therefore a task means an eight-hour working period, corresponding to a shift, to be assigned to one and only one nurse, and the number of tasks of a specific shift is equal to the minimum number of nurses required to work that shift.

The set of all nurse schedules of the unit is called a roster. The roster must comply with the rules of the labour contracts, with the hospital administration's internal regulations and, as far as possible, must satisfy the preferences of nurses for certain sequences of shifts/days off.

These regulations may be summarised by the following constraints:

(i) each nurse has an eight-hour working day in a single shift;
(ii) some nurses have an average weekly workload of 35 hours, while others have 42 hours;
(iii) nurses must rest at least 16 hours between two consecutive shifts; therefore some shift sequences are not feasible (e.g. work on an evening shift should not be followed by work on a night shift or day shift; as shown in Fig. 1, other sequences are also infeasible);
(iv) nurses must enjoy a minimum number of days' leave in every seven day sequence, depending on their weekly workload (2 days off for those with a 35 hour week, and 1 day off for a 42 hour week);
(v) nurses who should not be working on certain shifts and/or on certain days cannot be assigned to tasks pertaining to those shifts and/or days.

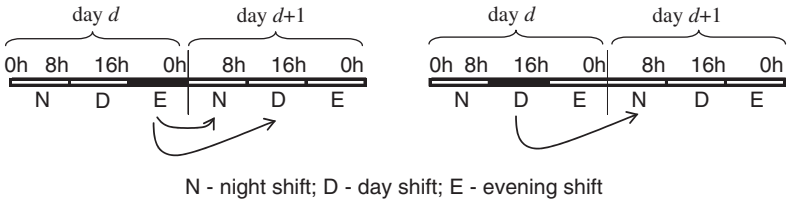


Fig. 1. Infeasible sequences of shifts on consecutive days.

In practice, after a roster has been made known, for unexpected reasons nurses often announce that they will be unable to perform some of their tasks. Since there is no reserve pool of nurses to replace those absent, the roster must be rebuilt.

The problem of nurse rerostering consists in building a new roster applicable to the first day of absences until the last day of the rostering period. The new roster must satisfy all the constraints to which the current one is subject and, obviously, it must not assign tasks to absent nurses. In this rerostering context, the goal is to find a new feasible roster as similar as possible to the current one, that is, one that ensures a minimum number of shift swaps in the nurse schedules.

Apart from the abovementioned (hard) constraints, there are other rules invoked by the head nurse, albeit in a very flexible manner, depending on the context. For example, sometimes it is felt that nurses should not work on a night shift on consecutive days. However, often, due to a temporary shortage of available personnel, there is no alternative, and the rule is not enforced. This type of rule, which may be modelled as soft constraint, has not, as yet, been taken into consideration. Nevertheless, solutions for the rerostering problem that minimise the dissimilarity with the current roster will tend to cover those constraints when satisfied by the current roster.

It is also worth noting that when the number of nursing hours arising from the absences fails to meet the total number required to ensure a minimum level of health care standards in the units, the problem is not one of rerostering. A situation such as this could not be solved by the head nurse. Such decisions must be made at hospital administration level.

3. Constructive heuristics

The search for feasible solutions to this nurse rerostering problem in a short computational time began with the design of a constructive heuristic. This heuristic is based on the classical assignment problem, with additional constraints that forbid tasks in some sequences of shifts on consecutive days for the same nurse, and enforce the minimum number of days' leave in a seven-day period.

The heuristic reassigns all tasks to nurses, from the first day of absences to the last day of the planning period. Example 3.1 illustrates a list of tasks for a rerostering instance.

Example 3.1. Consider the current roster for a seven-day period (Table 1), requiring one nurse for each day shift, in a hospital unit with a fixed group of five nurses.

The rerostering problem in this small-sized unit would, for instance, occur if nurse 3 could not work on the night shift on day 5. To rebuild this roster one would have to reassign the nine tasks, required on days 5, 6 and 7 (see Table 2), to the five nurses, while satisfying all the constraints.

Starting with a random ordering of all tasks for the rerostering period, the iterative step of the constructive heuristic attributes each task to one nurse, while respecting the constraints. First, it tries to assign the task to a nurse already scheduled in the current roster to a task on the same shift on the same day by verifying a set of hierarchically ordered rules, while imposing all the constraints of the problem. If, when respecting this order, a task cannot be assigned to any nurse, a backtracking procedure is undertaken to search for a position hitherto never occupied by this task, to avoid sequences that have already been tested (possible ties are broken by using a lexicographic decreasing order of the positions in the list of tasks). Example 3.2 illustrates the backtracking procedure. Once again, the iterative step is performed with this

Table 1
A current seven-day roster for a five-nurse unit

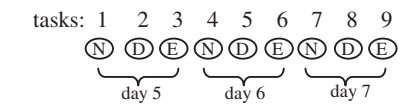
Day	1	2	3	4	5	6	7	ts
Nurse 1	D	E	O	N	O	D	E	5
Nurse 2	O	N	O	D	E	O	N	4
Nurse 3	O	D	E	O	N	O	D	4
Nurse 4	E	O	N	O	D	E	O	4
Nurse 5	N	O	D	E	O	N	O	4
nN	1	1	1	1	1	1	1	
nD	1	1	1	1	1	1	1	
nE	1	1	1	1	1	1	1	

O - day off, ts - number of assigned shifts, nN, nD, nE - number of nurses assigned in the current roster to night/day/evening shifts.

Table 2
The roster in Table 1 after nurse 3 announces her absence

Day	1	2	3	4	5	6	7	ts
Nurse 1	D	E	O	N	O	D	E	5
Nurse 2	O	N	O	D	E	O	N	4
Nurse 3	O	D	E	O	▲	O	D	4
Nurse 4	E	O	N	O	D	E	O	4
Nurse 5	N	O	D	E	O	N	O	4
nN	1	1	1	1	1	1	1	
nD	1	1	1	1	1	1	1	
nE	1	1	1	1	1	1	1	

▲ - absence
nN, nD, nE - number of nurses required in night / day / evening shifts



new ordering until all tasks are assigned, and a new roster is thus obtained. Otherwise, the algorithm stops without finding any feasible assignment. The time complexity of this constructive heuristic is $O(NT^2)$, where N is the number of nurses of the hospital unit and T is the number of tasks to be assigned.

Example 3.2. Consider the ordered list of the three tasks [$a\ b\ c$] to be assigned to three nurses. First, the procedure assigns a to a given nurse. It then assigns b to another nurse. Now suppose that task c cannot be assigned to the third nurse. The backtracking procedure 'unassigns' task b and succeeds in assigning task c to the second nurse. Then it fails to assign b to the third nurse. At this point the procedure searches for a previous position for task b, bearing in mind that it is pointless to try the second nurse for task b once more, as sequence a, b, c has already been unsuccessfully tested.

Table 3
Description of the constructive heuristic

Step 1 (initialisation):
All the tasks required in the rerostering period are randomly ordered in a list and considered to be assigned to no nurse; all the nurses are randomly ordered in CH_{alea}, or not in CH_{nat}
Step 2 (iterative):
2.1. While respecting the following hierarchical rules, assign the first non-assigned task in the list:
Rule 1: to a nurse who was scheduled in the current roster to a task in the same shift on the same day, if no constraint of the problem is violated; otherwise, try using Rule 2;
Rule 2: to a nurse who was not scheduled in the current roster for a task on the same shift on the same day, **but** to whom that assignment is compatible with the tasks assigned to that nurse on the previous day **and** the day after in the current roster, and if no constraint of the problem is violated; otherwise, try using Rule 3;
Rule 3: to a nurse who was not scheduled in the current roster to a task in the same shift on the same day, **but** to whom that assignment is compatible with the tasks assigned to that nurse on the previous day **or** the day after in the current roster, and if no constraint of the problem is violated; otherwise, try using Rule 4;
Rule 4: arbitrarily, to a nurse if no constraint of the problem is violated
If the task concerned was assigned to a nurse, go to Step 3
2.2. Backtrack in the list to search for a position that the task has never occupied; if such a position is found, insert the task in that position, go to 2.1. Otherwise, STOP
Step 3 (stopping criterion)
If there remains a task to assign in the list, go to Step 2. If all tasks are assigned, STOP

Two versions of the constructive heuristic were implemented and tested. In the first version, tasks are assigned to nurses in accordance with the natural order of their indexes (CH_{nat}) [12]. In the second one, not only are the tasks but the nurses also are randomly ordered in the initialisation step (CH_{alea}). This new version arose from the need to improve the diversity of solutions obtained when successively running the heuristic. The detailed description of this heuristic is presented in Table 3.

Since the quality of the solutions obtained through these procedures depends on the ordering of the tasks in the list, as well as on the ordering of the nurses, the search for better quality solutions may be performed by repeating the execution of the heuristic, with different orderings, over a specified amount of time. This amounts to a random search in the space of the permutations of task (and nurse) indexes. Clearly this procedure may lead to an improvement in the feasible solutions found. However, as an alternative, a guided search of such spaces was also implemented by developing GA. Both approaches were tested and the results are referred to in Section 6.2.

As is known, GA have been successfully applied in nurse rostering and scheduling problems, as indicated in [5].

4. Genetic algorithm—general description

In the construction of a GA the first issue to deal with is solution encoding. Two types of solution encoding may be used: direct encoding, which directly represents a solution of the rerostering problem and indirect encoding, the option in this paper. Direct encoding was not implemented as, almost

Table 4
Description of the genetic algorithm (GA)

Step 1 (initialisation)
Randomly generate the population of dimension *n_pop* for generation 1
$t \leftarrow 1$
Step 2 (evaluation)
Assign a fitness value to each individual of the population in generation t
Step 3 (iterative)
3.1. Selection—Copy *n_pop* individuals to the mating pool
3.2. Crossover and mutation—Randomly match pairs of individuals from the mating pool to form new individuals and submit them to mutation
3.3. Create the population for generation $t + 1$
$t \leftarrow t + 1$
Step 4 (stopping criterion)
If a stopping criterion is met, STOP. Otherwise, go to Step 2

invariably, the crossover operator acting on such chromosomes gives rise to infeasible rosters. Imposing feasibility to these rosters involves the resolution of problems which are as complex as the rerostering problem itself. Previous experience in nurse scheduling in the public hospital under analysis proved that to rebuild an infeasible roster is much more difficult than building the roster itself [9].

The idea of using a GA to search in a permutation space, when permutations are used as input to heuristics that solve the original problem, was inspired by an approach to the colouring problem developed by Davis [13].

In the GA described in this paper, each individual in the population is associated with a pair of chromosomes: one of them represents a permutation of the list of tasks, and the other a permutation of the list of nurses. In order to obtain a solution to the problem—a new roster—from an individual, a procedure, similar to the assigning process of the constructive heuristic (Step 2.1), is performed, where the two permutations corresponding to that individual act as input.

The algorithm starts with a population of *n_pop* individuals, which is randomly generated. Each individual, associated with a new roster, has a fitness value related to the dissimilarity between the current and the new roster. This new roster is the solution of the rerostering problem obtained from the permutations of that individual. Three operators—selection, crossover and mutation—are applied to the individuals in the population to produce a new population, which replaces the old one in the next generation. This process is iterated until a stopping criterion is met. Table 4 summarises the GA.

From the population of each generation, the selection operator copies *n_pop* individuals to a mating pool by using the roulette wheel selection procedure, see e.g. [15], according to which the probability of entering the mating pool is directly proportional to the fitness of the individual. Individuals in the mating pool are randomly matched, each pair (the parents) being recombined by applying a crossover operator to the respective chromosomes, thus generating two new individuals (two children). After recombination, the mutation operator submits the children's chromosomes to possible random alterations and includes them in the population for the next generation.

5. Genetic algorithm—implementation details

Several versions of this basic GA were developed. The differences among them lay in the encoding of permutations and in the genetic operators used for each encoding.

5.1. Permutation encoding

Two different ways of encoding permutations, i.e. chromosomes, were implemented: the natural encoding, which represents permutations by integer vectors, and the random keys encoding, which represents permutations by real vectors, as presented in [16, p. 176, 17]. Example 5.1 describes the decoding of random keys into permutations of integers.

Example 5.1. Random keys encoding.

Consider a problem with six tasks and the respective vector of six real numbers (0.50 0.02 0.10 0.03 0.25 0.09). A correspondence is established between each real number and a task. In this case, 0.50 would correspond to task 1, 0.02 to task 2, and so on. By sorting the components of the vector, (0.02 0.03 0.09 0.10 0.25 0.50) is obtained. This sorted vector, together with the above correspondence, yields the sequence of tasks by applying the following procedure: for all i $(i = 1, \ldots, 6)$, the task associated with the number in position i, in the sorted vector, is placed in position i of the sequence. For instance, 0.02, the first number in the sorted vector, corresponds to task 2. Hence, task 2 will be the first in the sequence; 0.03 corresponds to task 4, so task 4 will be the second one, and so on. For the abovementioned real vector, the encoded permutation will be (2,4,6,3,5,1), identical to the one directly obtained through natural encoding.

5.2. Crossover and mutation operators

As mentioned above, each individual is represented by two chromosomes. When two individuals in the mating pool are paired, the chromosome representing the permutation of the tasks of one is recombined with the corresponding chromosome of the other. The same occurs with the chromosomes representing the permutation of nurses. Hence, the operators described below apply in the same way to either pair of chromosomes. Pairs of chromosomes in the mating pool are modified through a crossover procedure with probability p_cross.

As usual, the simple crossover operator (one-point crossover) consists of randomly choosing a crossover point and then recombining the pieces of a pair of chromosomes to form two new chromosomes. The simple crossover is compatible with the random keys encoding, though it generally fails to preserve the permutation when dealing with natural encoding. Hence, for natural encoding, special crossover operators must be used. From the literature, the partially matched crossover (PMX) and the order based crossover (OX), see [13,15], were chosen to be implemented and tested: PMX tries to preserve the absolute positions of the integers, whereas OX tends to preserve the relative positions.

These special operators require two crossover points. Given two parent chromosomes, A and B, child chromosome A' will inherit from A the sub-sequence between these two points and child chromosome B', from B, the respective sub-sequence. The elements of A' and B' outside the two points are copied

from the other parent chromosome, while trying to preserve its position under the PMX operator (see Example 5.2), or by trying to preserve its order under the OX operator (see Example 5.3).

Example 5.2. PMX operator.

Consider parent chromosomes A and B with two crossover points (..|...|..),

$$A = (3\,5\,|\,2\,10\,8\,4\,|\,6\,1\,7\,9) \quad \text{and} \quad B = (8\,9\,|\,6\,4\,2\,3\,|\,7\,10\,5\,1).$$

Then A' and B' initially inherit the sub-sequences between the two points, from A and B respectively, to yield

$$A' = (\_\,\_\,|\,2\,10\,8\,4\,|\,\_\,\_\,\_\,\_) \quad \text{and} \quad B' = (\_\,\_\,|\,6\,4\,2\,3\,|\,\_\,\_\,\_\,\_).$$

Afterwards, the other values are copied from the other parent chromosome in the same position, if they are not yet present,

$$A' = (\_\,9\,|\,2\,10\,8\,4\,|\,7\,\_\,5\,1) \quad \text{and} \quad B' = (\_\,5\,|\,6\,4\,2\,3\,|\,\_\,1\,7\,9).$$

For those positions outside the two crossover points, with values already present in the sequence, components are assigned through a procedure based on the correspondence $6 \leftrightarrow 2$, $4 \leftrightarrow 10$, $2 \leftrightarrow 8$, $3 \leftrightarrow 4$. As for the first position of A', the value 8 (from B) cannot be assigned to it, because it is already present in the place of 2, but 2 is already present in the place of 6, which is not yet present ($8 \rightarrow 2 \rightarrow 6$). The number 6 can therefore be assigned to position 1 of A'. This procedure is iterated until all the positions are filled, thus obtaining

$$A' = (6\,9\,|\,2\,10\,8\,4\,|\,7\,3\,5\,1) \quad \text{and} \quad B' = (10\,5\,|\,6\,4\,2\,3\,|\,8\,1\,7\,9).$$

Example 5.3. OX operator.

Given parent chromosomes A and B and two OX points (..|...|..),

$$A = (3\,5\,|\,2\,10\,8\,4\,|\,6\,1\,7\,9) \quad \text{and} \quad B = (8\,9\,|\,6\,4\,2\,3\,|\,7\,10\,5\,1),$$

chromosomes A' and B' initially inherit the sub-sequences between the two points, from A and B, respectively, to yield

$$A' = (\_\,\_\,|2\,10\,8\,4\,|\,\_\,\_\,\_\,\_) \quad \text{and} \quad B' = (\_\,\_\,|\,6\,4\,2\,3\,|\,\_\,\_\,\_\,\_).$$

Next, where the positions outside the two points are concerned, beginning after the second point, the values are copied in keeping with the order in which they appear in the other parent, and in a circular way, if not yet present. This procedure gives rise to the child chromosomes:

$$A' = (6\,3\,|\,2\,10\,8\,4\,|\,7\,5\,1\,9) \quad \text{and} \quad B' = (10\,8\,|\,6\,4\,2\,3\,|\,1\,7\,9\,5).$$

After applying crossover, the mutation operator acts on the pairs of chromosomes of the individuals in the mating pool.

The simple mutation operates on each component of all chromosomes by randomly changing its value with a given (usually small) probability. Over random keys coded chromosomes a random change of the real number of a single component in a chromosome amounts to changing the position of a task in the

permutation represented by that chromosome. However, with natural encoding the simple mutation gives rise to a sequence of integers that may not represent a permutation. Therefore, another procedure was used: a chromosome is picked for mutation with probability *p_mut*, and two randomly chosen components of the vector are swapped. This special type of mutation used for natural encoding was tested also for random keys encoding, yielding worse results than those obtained with simple mutation. In the computational experiments reported in Section 6, only the results with simple mutation are presented, for the case of random keys encoding. Since, in natural encoding, the mutation acts over a whole chromosome and, in random keys encoding, simple mutation operates on each component of the chromosome, the magnitude of $p - mut$ has been correspondingly adjusted.

5.3. Decoding and fitness evaluation

The decoding process, developed to obtain a new roster from a pair of permutations encoded through the respective pair of chromosomes, characterizing an individual, consists of assigning tasks to nurses, by using the procedure described in Step 2.1 of the constructive heuristic. When a feasible solution is formed, that is a new roster, the solution value is made equal to the dissimilarity between this roster and the current roster. For a pair of permutations that do not lead to a feasible solution, the solution value is obtained by adding a penalisation value p, for each non-assigned task, to the value of the dissimilarity.

The GA evaluation step amounts to obtaining the *n_pop* solutions (new rosters and, possibly, some non-feasible solutions), corresponding to the *n_pop* individuals of the current generation's population, from the respective encoded pairs of permutations, and transforming the value of each solution into a fitness value. The fitness function to be maximised is given by $fitness = M - value$, where *value* quantifies the dissimilarity, penalised or otherwise, and M is a sufficiently large number to guarantee that *fitness* always assumes non-negative values. All the implemented versions of GA include the transformation of fitness according to the linear scaling procedure, proposed by Goldberg [15], thus avoiding premature convergence and allowing a significant diversity within the population, in the late phase of the execution process. In this linear scaling procedure, upper and lower bounds are adaptively determined in each generation by the current best and worst fitness values.

5.4. Elitism and stopping criterion

As stated above, selection was implemented using the roulette wheel operator. However, in the light of previous experience with GA and theoretical results on convergence pointing to the advantage of elitist criteria (see e.g. [18] or [19]), in all versions of the algorithms developed for the nurse rerostering problem under study, one individual in the new population was replaced by the fittest of the previous generation.

According to the stopping criterion implemented, the execution stops after *max_gen* generations or after *n_stop* generations without improving the best fitness value.

5.5. Hybridisation

Several versions of the above described GA with the different types of encoding were developed and tested: GA_{PMX} and GA_{OX}, with natural encoding and GA_{RK} with random keys encoding.

Since feasible solutions were hard to obtain when GA_{PMX} and GA_{OX} were applied to some instances, hybridisation with the random version of the constructive heuristic was implemented.

When running each of these versions of the algorithm, if *n_gen* generations pass without registering the presence of an individual corresponding to a (feasible) roster, then a pseudo-random uniform number in the interval (0,1) is drawn for each individual of that generation. If this number is less than *p_hybrid* (the probability of hybridisation) the constructive heuristic is triggered off, taking as input the pair of chromosomes of the respective individual. GA_{PMX} and GA_{OX} hybridised led to two new versions: HH_{PMX} and HH_{OX}.

6. Computational experiments

6.1. Description of the experiments

In order to test and compare the performance of the various heuristics, computational experiments[1] were performed with 68 instances. These instances were generated by randomly creating absences over real rosters from two units of the hospital under study, involving 19 and 32 nurses, respectively.[2]

The test instances are classified into four groups according to the first day of absences and thus the length of the rerostering period. Group I includes smaller instances, where the first day of absences occurs during the fourth week of the 28-day rostering period, group II, during the third week and so on. For the 32-nurses' unit a fifth group was also included, with instances of a larger dimension. Table 5 presents some features of the instances: the first day of absences (column 2 and column 6, respectively for the 19-nurses' unit and the 32-nurses' units), the number of absent nurses (columns 3 and 7) and the number of days with absences (columns 4 and 8).

To select the values of the GA parameters, a preliminary study was carried out on a subset of particularly difficult instances. Here, in some initial tests, the five GA versions more frequently reached solutions with less quality than the best solutions obtained with the constructive heuristic (see details in [11]). For each combination of parameter values, six runs of each GA version were then executed with these instances. The values selected for the parameters are displayed in Table 8 in Appendix B.

The main computational experiments consisted in executing twenty runs with each version of the heuristics for each instance. Tables 10–15 in Appendix B reveal the best and worst solution values (optimum values are in bold) obtained in these twenty runs, along with the total computational time of the repeated runs.

6.2. Analysis of the results

The computational results are summarised in Table 6. This table refers to 67 instances[3] and contains the number of feasible solutions and number of optimal solutions[4] obtained for the test instances with

[1] All the procedures were coded in Pascal (Borland Delphi, version 5.0). The algorithms ran on a PC with a 2.6 GHZ and 256 Mb RAM Intel Pentium IV processor.

[2] A detailed description of the procedure used to generate the test instances may be found in [11].

[3] The table excludes one of the test instances (II.7_19) that proved to be one with no feasible solution.

[4] The optimal solutions were obtained by solving an ILP formulation of the problem with CPLEX 7.0 (see Table 9 in Appendix B).

Table 5
Features of the instances

1	2	3	4	5	6	7	8
	19-nurses' unit				32-nurses' unit		
Instance	First day of absences	Number of absent nurses	Number of days with absences	Instance	First day of absences	Number of absent nurses	Number of days with abssences
I.1_19	27	2	2	I.1_32	27	2	2
I.2_19	26	2	3	I.2_32	26	2	3
I.3_19	24	3	5	I.3_32	24	3	5
I.4_19	22	1	1	I.4_32	22	1	1
I.5_19	24	7	5	I.5_32	24	4	5
I.6_19	22	2	7	I.6_32	25	9	4
I.7_19	22	4	7	I.7_32	26	10	4
I.8_19	22	1	1	I.8_32	26	10	3
II.1_19	20	1	1	II.1_32	20	1	1
II.2_19	20	1	1	II.2_32	20	1	1
II.3_19	19	1	3	II.3_32	19	1	3
II.4_19	16	1	8	II.4_32	16	1	8
II.5_19	20	1	6	II.5_32	17	8	12
II.6_19	18	6	11	II.6_32	20	6	9
II.7_19	20	5	9	II.7_32	18	4	8
II.8_19	16	1	4	II.8_32	16	3	8
III.1_19	13	5	3	III.1_32	13	5	3
III.2_19	12	4	2	III.2_32	12	4	2
III.3_19	11	4	3	III.3_32	11	4	4
III.4_19	9	3	4	III.4_32	9	3	4
III.5_19	8	3	17	III.5_32	13	11	15
III.6_19	9	5	12	III.6_32	11	12	18
III.7_19	12	3	9	III.7_32	9	8	20
III.8_19	10	1	8	III.8_32	14	10	15
IV.1_19	6	2	2	IV.1_32	6	2	2
IV.2_19	5	3	3	IV.2_32	5	3	3
IV.3_19	4	2	2	IV.3_32	4	2	2
IV.4_19	1	3	5	IV.4_32	1	3	5
1V.5_19	2	3	15	IV.5_32	5	3	14
IV.6_19	5	4	12	IV.6_32	6	1	2
IV.7_19	1	5	12	IV.7_32	4	2	13
IV.8_19	1	3	13	IV.8_32	3	9	26
				V.l_32	1	9	28
				V.2_32	1	10	28
				V.3_32	1	9	28
				V.4_32	1	15	28

Table 6
Computational results—67 instances and 20 runs

Heuristic versions	Number of feasible solutions	Number of optimal solutions	Average % gap (number of sub-optimal solutions)	Average time (s)
CH_{nat}	67	34	21.70 (33)	0.85
CH_{alea}	67	39	19.90 (28)	0.75
GA_{OX}	63	59	10.90 (4)	1130.81
GA_{PMX}	66	60	8.04 (6)	1143.23
GA_{RK}	67	60	7.51 (7)	1104.85
HH_{OX}	67	60	13.94 (7)	1149.25
HH_{PMX}	67	60	7.40 (7)	1123.97

the repeated runs of each heuristic, together with the average percentage gap of the solution values in relation to the optimum values for the sub-optimal solutions only, and the average computational time for twenty runs of the respective heuristic.

In terms of computational time, the two versions of the constructive heuristic performed very well. In fact, the total time in seconds consumed by the twenty runs of either version is on average 0.85 and 0.75 (Table 6) and varies between 0 and 30 seconds (see Tables 10 and 11, in Appendix B). As for solution quality, CH_{alea} outperforms CH_{nat} since it obtained both a larger number of feasible and optimal solutions. This may be due to the fact that CH_{nat} deals with the list of nurses, ordered by seniority, given by the head nurse, and is therefore less flexible to schedule. Since the computational times are very similar, CH_{alea} is clearly the best option.

However, despite the disadvantage of longer computational times, all versions of the GA, except GA_{OX}, produced a larger number of feasible and optimal solutions when compared with the constructive heuristics. If one considers the non-hybridised versions alone, as seen in Table 6, GA_{RK} performed slightly better in terms of solution quality. Hybridisation of GA_{PMX}, that is HH_{PMX}, shows a similar behaviour, albeit at the cost of a longer computational time.

The GA performance superiority over the constructive heuristic was also tested in another experiment that consisted in running CH_{alea} successively until it reached the total running time of HH_{PMX} per instance (see [11]). In this test, CH_{alea} never outperformed HH_{PMX}, and gave worse results in 10% of the instances.

Furthermore, a non-parametric statistical test was carried out to more accurately compare the quality of the solutions obtained by the different heuristics. The classical binomial test [20] was chosen, in order

Table 7
Binomial test reulsts

Heruistic versions paired-up	Units	Number of observations	Number of "-"	p-value** (2-tailed)
CH_{alea}/CH_{nat}	19-nurses	490	247	0.892
	32-nurses	562	419	0.000
GA_{PMX}/GA_{OX}	19-nurses	130	127	0.000
	32-nurses	*	*	*
GA_{RK}/GA_{OX}	19-nurses	162	148	0.000
	32-nurses	32	20	0.215
GR_{RK}/GA_{PMX}	19-nurses	129	52	0.034
	32-nurses	32	20	0.215
HH_{PMX}/HH_{OX}	19-nurses	151	140	0.000
	32-nurses	20	18	0.000
GA_{RK}/HH_{PMX}	19-nurses	114	35	0.000
	32-nurses	32	6	0.001
GA_{PMX}/HH_{PMX}	19-nurses	100	37	0.012
	32-nurses	20	0	0.000

*There are not enough valid cases to perform binomial test.
**When the number of observations is greater than 20, the figure represents the asymptotic significance, otherwise is the exact significance.

to handle the situations in which the heuristics did not come up with feasible solutions, hence treated as non-numerical outcomes.

In this statistical study, the two hospital units are analysed separately. Heuristic versions are paired up and, for each run per instance, the respective solution values are compared. When a tie occurs the observation is dropped, which explains why the number of valid pairs observed differs from test to test. The study considers the null hypothesis (H_0), stating that both heuristics achieve similar behaviour. This is summarised in Table 7 in which column 3 displays the number of valid observations for each heuristic pair and each hospital unit and column 4 counts the number of occurrences where the solution value obtained through heuristic A is less than the respective value from heuristic B, for the pair A/B. The last column of this table shows the significance results.

For the case of the 32-unit's instances, the binomial test at a significance level of $a = 0.05$ confirms that CH_{alea} outperforms CH_{nat}, although for the 19-unit's case they register similar performances. From the pair-wise comparison of GA_{OX}, GA_{PMX} and GA_{RK}, for the 19-nurses' unit, it may be concluded that the null hypothesis is always rejected, that is, genetic heuristics display different performances. As for the other unit, the test accepted the null hypothesis, thus confirming equivalent behaviour of the heuristics. Between the hybridised versions, for both units, H_0 was rejected and HH_{PMX} gave better results, as seen in columns 5 and 4 of Table 7.

Finally, as GA_{OX} showed the worst performance (see column 4, Table 7), the test was applied to the two other non-hybridised GA versions, each one paired with HH_{PMX}. At the same level of significance,

the test again rejected the null hypothesis and, from Table 7, one can see that HH_{PMX} behaved better in terms of solution quality.

Overall, the behaviour of the genetic-based algorithms, in particular HH_{PMX}, may be considered satisfactory. The solution quality is good, and the computational time required is well below the two-hour limit stipulated by the head nurses.

7. Conclusion

Previous research on the nurse rerostering problem in a Lisbon hospital had already been carried out, using a simple constructive heuristic that sequentially assigns tasks to nurses, using a permutation of tasks as input. At the time, it was conjectured that a GA search in the space of both permutations might lead to a substantial improvement in solution quality. As a result, the previous constructive heuristic was slightly altered and hybridised with a GA. The genetic procedure is basically a classical version [13], though developed according to the specifications of the rerostering problem under study, namely, the assignment of two chromosomes to each individual of the population.

The computational tests carried out confirmed the above conjecture. Notwithstanding longer computational times, the GA clearly outperformed the constructive heuristic in terms of solution quality, without exceeding acceptable time limits, thus suggesting that these algorithms may act as an effective heuristic device to solve the practical problem addressed by the authors. In particular, one of the genetic versions hybridised with the random version of the constructive heuristic yielded the best overall results.

In the near future, further work will be pursued to compare the performance of the GA in the search of the permutations' spaces with local search procedures, and genetic/local search hybrids. Moreover, the problem of rerostering must be tackled with both hard and soft constraints by developing multi-objective versions of the heuristics.

Following the direction of the overall project of a computational system to manage nurse schedules in Portuguese public hospitals, those heuristic approaches that have proved successful for the purpose of rerostering will also be developed and tested for the rostering problem.

Acknowledgements

The authors are grateful to the anonymous referees and to the editors for their valuable suggestions and comments. They also thank the head nurses of the units studied for their patient collaboration and readiness in providing the required data, as well as João Manuel Andrade e Silva from ISEG (Universidade Técnica de Lisboa) for his technical assistance on statistical analysis. This research was supported in part by POCTI/ISFL/152.

Appendix A

In order to study the computational complexity of the nurse rerostering problem, this problem is formulated as one of determining a set of n disjoint paths of minimum total cost in a multi-level directed network.

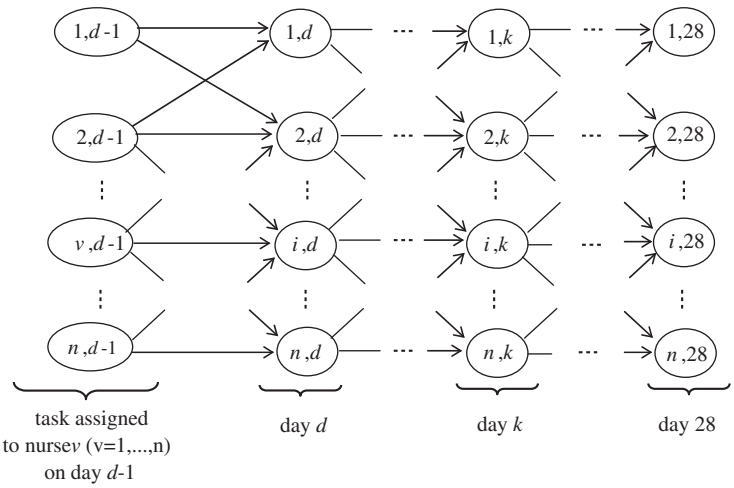


Fig. 2. Directed $(30 - d)$-level acyclic network for the rerostering problem.

Consider a hospital unit with n nurses working under the conditions defined in Section 2. Moreover, suppose that the scheduling period of 28 days has already begun, that nurses have been working up to day $(d - 1)$ according to the current roster and that day d is the first day of absences. Hence, the network for the rerostering problem has $30 - d$ levels, each of which containing n nodes. In the first level, each node is associated with a nurse. In each of the following levels, as many as the number of days of the rerostering period, the n nodes represent the n tasks (work or 'day off' tasks) to be assigned in each day.

The arcs of the network, from level d to level 28, represent feasible sequences of tasks on consecutive days. However, an arc from a node of the first level to a node of level d exists if the nurse represented by the former node performed a task on day $d - 1$ compatible with the task represented by the second node. Fig. 2 displays the directed $(30 - d)$-level acyclic network for the rerostering problem. Note that a path in this network, from a node of the first level to a node of the last level represents the schedule of a specific nurse, which is the one associated with the origin of the path.

The rerostering feasibility problem consists in verifying if there exists a set of n disjoint paths in this network, while respecting the additional constraints that impose that paths must go through at least 2 (or 1) day-off nodes in each sequence of seven levels.

Proposition 1. *The rerostering feasibility problem is* NP-*complete*.

Proof. First, one will show that the feasibility problem addressed belongs to the class NP, then that it possesses a particular case, the restrict problem, which is identical to a well-known NP-complete problem, the three-dimensional matching problem or 3-*DM* (see [21]).

The rerostering feasibility problem is in NP, since a non-deterministic algorithm for this problem need only propose n schedules and check, in polynomial time, the feasibility of the respective roster. As for the restrict problem, it is obtained by considering the first day of absences occurring on the penultimate day of the period, that is $d = 27$. This problem is identical to the 3-*DM*, which is NP-complete [21, p. 50]. □

However, the rerostering problem is an optimisation problem, whose objective is to minimise the dissimilarity between the current and the new roster. This dissimilarity can be expressed in network terminology by defining costs associated with each arc of the multi-level network, which depend on the path that passes through that arc, in other words, on the respective nurse. Thus, associated with the network's arc $((i, k), (j, k + 1))$ and for each nurse v the parameter c^v_{ikj} is defined with the following values and meanings:

$$c^v_{ikj} = \begin{cases} 0 & \text{if, in the current roster, nurse } v \text{ performs a task on the same shift as that of task } j \text{ on day } k+1; \\ M & \text{if, in the new roster, nurse } v \text{ cannot be assigned to task } i \text{ on day } k \text{ or to task } j \text{ on day } k+1 \text{ (either due to absence or to the fact that she can only work on specific shifts, which are not those of task } i \text{ or } j\text{), where } M \text{ is a sufficiently large positive number;} \\ 1 & \text{otherwise (in the current roster, nurse } v \text{ does not perform a task on the same shift as that of task } j \text{ on day } k+1\text{).} \end{cases}$$

As usual, the cost of a path from node v of the first level to a node of the last level is fixed as equal to the sum of the costs of the arcs in that path. Thus, if not penalised with M costs, it indicates the number of swaps between shifts assigned to nurse v in the current and new rosters. The total cost of a roster is the sum of the costs of the respective n disjoint paths.

In this context, the rerostering (optimisation) problem can be formulated as that of determining n disjoint paths on the above described network with a minimal total cost, while satisfying the additional constraints imposing 1 or 2 days off in each sequence of seven days. Because this optimisation problem is at least as difficult as the corresponding feasibility problem, the following result comes straightforwardly:

Proposition 2. *The rerostering problem is* NP*-hard.*

Appendix B

Selected values for parameters, optimum values and computational results are given in Tables 8–15.

Table 8
Selected value for the parameters for each GA version

Parameters	Meaning	GA_{PMX}	GA_{OX}	GA_{RK}	HH_{OX}	HH_{PMX}
n_pop	Population length	400	400	200	400	400
p_cross	Probability of crossover	0.6	0.6	0.6	0.6	0.6
p_mut	Probability of mutation	0.1	0.05	0.005	0.05	0.1
p	Penalisation value	*T*	*T*	*T*	*T*	*T*
max_gen	Maximum number of generations	2000	2000	2000	2000	2000
n_stop	Maximum number of generations without improvement	400	400	400	400	400
n_gen	Number of generations before hybridisation	—	—	—	200	200
p_hybrid	Probability of hybridisation	—	—	—	0.001	0.001

Table 9
Optimum values obtained by solving an ILP formulation of the rerostering problem [10]

Instance	Optimum value	Instance	Optimum value	Instance	Optimum value	Instance	Optimum value
19-nurses' unit							
I.l_19	3	II.l_19	1	III.1_19	7	IV.1_9	9
I.2_19	2	II.2_19	0	III.2_19	12	IV.2_19	12
I.3_19	9	II.3_19	5	III.3_19	13	IV.3_19	10
I.4_19	2	II.4_19	12	III.4_19	7	IV.4_19	34
I.5_19	20	II.5_19	6	III.5_19	27	IV.5_9	18
I.6_19	8	II.6_19	17	III.6_19	27	IV.6_9	23
I.7_19	20	II.7_19	*	III.7_19	19	IV.7_19	9
I.8_19	2	II.8_19	5	III.8_19	11	IV.8_19	10
32-nurses' unit							
I.l_32	5	II.1_32	5	III.1_32	13	IV.1_32	14
I.2_32	5	II.2_32	5	III.2_32	13	IV.2_32	15
I.3_32	8	II.3_32	7	III.3_32	15	IV.3_32	14
1.4_32	3	II.4_32	13	III.4_32	14	IV.4_32	20
I.5_32	10	II.5_32	20	III.5_32	25	IV.5_32	21
I.6_32	15	II.6_32	24	III.6_32	44	IV.6_32	11
I.7_32	9	II.7_32	10	III.7_32	25	1V.7_32	20
I.8_32	10	I1.8_32	11	IIII.8_32	30	IV.8_32	30
V.1_32	14	V.2_32	27	V.3_32	28	V.4_32	117

*Problem without feasible solutions.

Table 10
Computational results—constructive heuristics—19-nurses' units

1	2	3	4	5	6	7
	CH_{nat}			CH_{alea}		
Instance	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution	
I.1_19	**3**	3	0.10	**3**	4	0.17
I.2_19	**2**	2	0.02	**2**	4	0.02
I.3_19	**9**	13	0.02	**9**	15	0
I.4_19	**2**	4	0	**2**	8	0
I.5_19	22	26	0.12	22	27	0.10
I.6_19	11	22	0.04	9	19	0.05
I.7_19	26	Infeasible	3.05	25	Infeasible	3.40
I.8_19	**2**	7	0.04	**2**	5	0.04
II.1_19	**1**	5	0	**1**	6	0
II.2_19	**0**	0	0	**0**	0	0.02
II.3_19	**5**	14	0	**5**	14	0.03
II.4_19	13	24	0.08	14	21	0.07
II.5_19	**6**	13	0	7	16	0.02
II.6_19	20	29	0.18	24	35	0.07
II.7_19	Infeasible	Infeasible	3.85	Infeasible	Infeasible	4.23
II.8_19	6	11	0.04	**5**	10	0.07
III.1_19	**7**	13	0.08	**7**	15	0.04
III.2_19	14	23	0.35	14	21	0.29
III.3_19	16	31	0.54	17	30	0.43
III.4_19	9	15	0.02	8	19	0.04
III.5_19	36	63	2.93	35	58	0.89
III.6_19	36	58	5.34	38	53	3.29
III.7_19	21	35	0.77	**19**	38	0.99
III.8_19	17	24	0.08	12	31	0.05
IV.1_19	12	22	0.10	12	21	0.19
IV.2_19	15	23	0.13	15	24	0.16
IV.3_19	14	24	0.19	14	23	0.16
IV.4_19	40	56	10.24	39	54	6.72
IV.5_19	26	42	0.36	26	46	0.25
IV.6_19	32	46	2.87	29	45	1.51
IV.7_19	13	22	0.14	11	23	0.12
IV.8_19	14	29	0.08	12	25	0.09

Best and worst solution values and total time of 20 runs for each instance.

Table 11
Computational results—constructive heuristics—32-nurses' unit

1	2	3	4	5	6	7
Instance	CH_{nat}			CH_{alea}		
	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution	
I.1_32	**5**	7	0	**5**	7	0.02
I.2_32	**5**	6	0	**5**	7	0.02
I.3_32	**8**	10	0.02	**8**	10	0.02
I.4_32	**3**	3	0.04	**3**	5	0.05
I.5_32	**10**	11	0	**10**	14	0.02
I.6_32	16	19	0.02	16	22	0
I.7_32	**9**	12	0	**9**	11	0
I.8_32	**10**	13	0	**10**	12	0
II.1_32	**5**	8	0.04	**5**	7	0
II.2_32	**5**	8	0	**5**	7	0
II.3_32	7	13	0	**7**	11	0.02
II.4_32	**13**	19	0.06	**13**	18	0.02
II.5_32	21	28	0.02	**20**	23	0.02
II.6_32	25	36	0.06	**24**	33	0.02
II.7_32	**10**	17	0	**10**	14	0.02
II.8_32	**11**	15	0	**11**	13	0
III.1_32	**13**	19	0.02	**13**	16	0.03
III.2_32	**13**	22	0.02	**13**	19	0
III.3_32	**15**	21	0.06	**15**	20	0.03
III.4_32	**14**	18	0	**14**	19	0.04
III.5_32	26	31	0.06	**25**	31	0
III.6_32	48	61	0.23	45	53	0.14
III.7_32	26	29	0.04	26	33	0.02
III.8_32	32	41	0.11	31	40	0.04
IV.1_32	**14**	18	0.02	**14**	20	0.02
IV.2_32	**15**	20	0.06	**15**	18	0.04
IV.3_32	**14**	23	0	**14**	19	0.02
IV.4_32	21	29	0.06	**20**	29	0.04
IV.5_32	**21**	28	0.04	**21**	28	0.05
IV.6_32	**11**	15	0.04	**11**	15	0.02
IV.7_32	**20**	24	0.02	**20**	27	0.02
IV.8_32	**30**	39	0.04	**30**	36	0.03
V.1_32	17	28	0.04	15	27	0.02
V.2_32	30	47	0.35	30	47	0.28
V.3_32	31	51	0.29	31	49	0.28
V.4_32	141	166	27.42	144	166	29.66

Best and worst solution values and total time of 20 runs for each instance.

Table 12
Computational results—GA versions without hybridisation—19-nurses' unit

1	2	3	4	5	6	7	8	9	10
	GA_{OX}			GA_{PMX}			GA_{RK}		
Instance	Value		Time (s)	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution		Best solution	Worst solution	
I.1_19	**3**	3	1771.13	**3**	3	162.96	**3**	3	233.60
I.2_19	**2**	2	197.17	**2**	2	188.63	**2**	2	255.97
I.3_19	**9**	9	293.37	**9**	9	281.55	**9**	9	336.12
I.4_19	**2**	2	359.13	**2**	2	355.34	**2**	2	381.18
I.5_19	22	Infeasible	396.99	**20**	Infeasible	452.33	**20**	22	415.21
I.6_19	9	9	373.47	9	9	380.91	9	9	405.90
I.7_19	Infeasible	Infeasible	515.02	22	Infeasible	614.61	22	25	666.75
I.8_19	**2**	2	337.71	**2**	2	353.44	**2**	2	387.27
II.1_19	**1**	1	403.44	**1**	1	420.42	**1**	1	439.49
II.2_19	**0**	0	367.41	**0**	0	384.21	**0**	0	437.84
II.3_19	**5**	5	473.50	**5**	5	483.16	**5**	5	481.20
II.4_19	**12**	12	628.49	**12**	12	642.69	**12**	12	609.10
II.5_19	**6**	6	430.21	**6**	6	434.34	**6**	6	450.05
II.6_19	**17**	17	698.30	**17**	17	606.24	**17**	19	620.07
II.7_19	Infeasible	Infeasible	912.86	Infeasible	Infeasible	1116.05	Infeasible	Infeasible	1247.74
II.8_19	**5**	5	594.49	**5**	5	606.97	**5**	5	586.77
III.1_19	**7**	7	721.52	**7**	7	729.13	**7**	7	692.27
III.2_19	**12**	14	1023.76	**12**	13	1171.13	**12**	13	940.16
III.3_19	**13**	15	1183.46	**13**	14	1097.03	**13**	14	1075.32
III.4_19	**7**	7	900.01	**7**	7	912.35	**7**	7	857.03
III.5_19	28	Infeasible	1880.27	28	29	1506.30	28	35	2051.26
III.6_19	Infeasible	Infeasible	1718.47	29	Infeasible	1806.10	27	34	1622.35
III.7_19	**19**	20	1245.86	**19**	19	892.99	**19**	19	1026.12
III.8_19	**11**	11	884.74	**11**	11	886.01	**11**	11	830.63
IV.1_19	**9**	9	1051.50	**9**	9	1050.70	**9**	9	1039.20
IV.2_19	**12**	12	1220.82	**12**	12	1138.62	**12**	12	1291.24
IV.3_19	**10**	10	1264.19	**10**	10	1184.96	**10**	11	1152.11
IV.4_19	Infeasible	Infeasible	2605.65	36	Infeasible	2401.36	38	39	2247.07
IV.5_19	**18**	24	2218.13	**18**	19	1519.20	**18**	19	1885.31
IV.6_19	27	Infeasible	2115.57	25	Infeasible	2208.26	24	30	1963.49
IV.7_19	**9**	9	1284.89	**9**	9	1295.11	**9**	9	1212.84
IV.8_19	**10**	10	1345.07	**10**	10	1300.47	**10**	10	1376.57

Best and worst solution values and total time of 20 runs for each instance.

Table 13
Computational results—GA versions with hybridisation—19-nurses' unit

1	2	3	4	5	6	7
Instance	HH_{OX}			HH_{PMX}		
	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution	
I.1_19	**3**	3	147.35	**3**	3	163.5
I.2_19	**2**	2	170.95	**2**	2	186.66
I.3_19	**9**	9	271.00	**9**	9	279.98
I.4_19	**2**	2	344.39	**2**	2	355.70
I.5_19	**20**	23	545.49	**20**	21	453.79
I.6_19	9	9	379.25	9	9	380.75
I.7_19	22	28	728.67	21	25	717.40
I.8_19	**2**	2	347.40	**2**	2	354.50
II.1_19	**1**	1	411.04	**1**	1	420.24
II.2_19	**0**	0	347.88	**0**	0	384.42
II.3_19	**5**	5	485.05	**5**	**5**	485.90
II.4_19	**12**	12	644.16	**12**	12	644.55
II.5_19	**6**	6	438.39	**6**	6	437.51
II.6_19	**17**	17	724.06	**17**	17	624.03
II.7_19	Infeasible	Infeasible	1808.28	Infeasible	Infeasible	2199.36
II.8_19	**5**	5	607.01	**5**	5	608.02
III.1_19	**7**	7	729.27	**7**	7	729.30
III.2_19	**12**	13	1094.38	**12**	14	1002.30
III.3_19	**13**	15	1138.14	**13**	14	1138.48
III.4_19	**7**	7	917.09	**7**	7	917.48
III.5_19	29	52	2254.51	28	30	1570.93
III.6_19	30	48	2154.15	28	37	1729.36
III.7_19	**19**	21	1300.00	**19**	19	912.13
III.8_19	**11**	11	899.54	**11**	11	891.56
IV.1_19	**9**	9	1060.93	**9**	9	1044.69
IV.2_19	**12**	12	1234.05	**12**	12	1140.94
IV.3_19	**10**	10	1281.05	**10**	10	1197.91
IV.4_19	38	46	2988.85	37	39	2528.40
IV.5_19	**18**	24	2048.30	**18**	21	1485.12
IV.6_19	28	42	2585.34	25	32	2509.28
IV.7_19	**9**	9	1315.23	**9**	9	1271.72
IV.8_19	**10**	10	1391.41	**10**	10	1307.64

Best and worst solution values and total of 20 runs for each instance.

Table 14
Computational results—GA versions without hybridisation—32-nurses' unit

1	2	3	4	5	6	7	8	9	10
	GA_{OX}			GA_{PMX}			GA_{RK}		
Instance	Value		Time (s)	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution		Best solution	Worst solution	
I.1_32	**5**	5	189.29	**5**	5	210.58	**5**	5	273.88
I.2_32	**5**	5	233.60	**5**	5	247.11	**5**	5	317.47
I.3_32	**8**	8	358.25	**8**	8	378.80	**8**	8	414.69
I.4_32	**3**	3	498.29	**3**	3	503.44	**3**	3	531.98
I.5_32	**10**	10	367.02	**10**	10	384.41	**10**	10	415.50
I.6_32	**15**	15	329.59	**15**	15	346.60	**15**	15	386.17
I.7_32	**9**	9	244.46	**9**	9	268.16	**9**	9	327.64
I.8_32	**10**	10	251.16	**10**	10	263.13	**10**	10	323.93
II.1_32	**5**	5	630.29	**5**	5	647.81	**5**	5	630.34
II.2_32	**5**	5	629.63	**5**	5	649.23	**5**	5	637.20
II.3_32	**7**	7	701.89	**7**	7	720.48	**7**	7	682.43
II.4_32	**13**	13	926.56	**13**	13	930.65	**13**	13	874.38
II.5_32	**20**	20	899.82	**20**	20	906.02	**20**	20	835.08
II.6_32	**24**	24	708.88	**24**	24	734.63	**24**	24	684.08
II.7_32	**10**	10	769.81	**10**	10	785.88	**10**	10	742.99
II.8_32	**11**	11	923.07	**11**	11	935.76	**11**	11	893.52
III.1_32	**13**	13	1120.55	**13**	13	1140.81	**13**	13	1040.15
III.2_32	**13**	13	1190.78	**13**	13	1207.82	**13**	13	1094.29
III.3_32	**15**	15	1254.15	**15**	15	1274.74	**15**	15	1164.23
III.4_32	**14**	14	1400.11	**14**	14	1415.54	**14**	14	1323.77
III.5_32	**25**	25	1166.02	**25**	25	1173.57	**25**	25	1062.53
III.6_32	**44**	44	1534.35	**44**	44	1463.45	**44**	45	1392.85
III.7_32	**25**	25	1542.99	**25**	25	1551.70	**25**	26	1482.01
III.8_32	**30**	30	1163.90	**30**	30	1157.74	**30**	30	1034.84
IV.1_32	**14**	14	1619.41	**14**	14	1598.38	**14**	14	1519.24
IV.2_32	**15**	15	1694.17	**15**	15	1682.00	**15**	15	1593.58
IV.3_32	**14**	14	1774.94	**14**	14	1759.73	**14**	14	1691.58
IV.4_32	**20**	20	2051.07	**20**	20	2008.45	**20**	20	1921.01
IV.5_32	**21**	21	1723.28	**21**	21	1706.33	**21**	21	1630.99
IV.6_32	**11**	11	1603.16	**11**	11	1581.36	**11**	11	1526.02
IV.7_32	**20**	20	1779.52	**20**	20	1768.20	**20**	20	1715.68
IV.8_32	**30**	30	1928.91	**30**	30	1907.85	**30**	30	1830.07
V.1_32	**14**	14	2096.66	**14**	14	2039.48	**14**	14	1930.94
V.2_32	**27**	27	2509.87	**27**	27	2244.96	**27**	27	2588.24
V.3_32	**28**	28	2627.37	**28**	28	2233.54	**28**	28	2214.59
V.4_32	Infeasible	Infeasible	6411.71	Infeasible	Infeasible	9300.55	129	161	7327.75

Best and worst solution values and total time of 20 runs for each instance.

Table 15
Computational results—GA versions with hybridisation—32-nurses' unit

1	2	3	4	5	6	7
Instance	HH_{OX}			HH_{PMX}		
	Value		Time (s)	Value		Time (s)
	Best solution	Worst solution		Best solution	Worst solution	
I.1_32	**5**	5	186.77	**5**	5	210.74
I.2_32	**5**	5	229.67	**5**	5	241.89
I.3_32	**8**	8	360.87	**8**	8	372.27
I.4_32	**3**	3	508.67	**3**	3	499.36
I.5_32	**10**	10	367.44	**10**	10	376.00
I.6_32	**15**	15	326.51	**15**	15	348.78
I.7_32	**9**	9	246.41	**9**	9	261.49
I.8_32	**10**	10	244.74	**10**	10	264.32
II.1_32	**5**	5	628.98	**5**	5	626.37
II.2_32	**5**	5	635.10	**5**	5	633.84
II.3_32	**7**	7	710.68	**7**	7	707.69
II.4_32	**13**	13	939.87	**13**	13	921.69
II.5_32	**20**	20	894.63	**20**	20	900.99
II.6_32	**24**	24	724.12	**24**	24	725.85
II.7_32	**10**	10	778.75	**10**	10	782.62
II.8_32	**11**	11	929.45	**11**	11	923.21
III.1_32	**13**	13	1137.13	**13**	13	1122.19
III.2_32	**13**	13	1217.00	**13**	13	1183.45
III.3_32	**15**	15	1282.53	**15**	15	1250.79
III.4_32	**14**	14	1437.99	**14**	14	1390.03
III.5_32	**25**	25	1197.30	**25**	25	1177.45
III.6_32	**44**	44	1663.31	**44**	44	1499.19
III.7_32	**25**	25	1593.01	**25**	25	1601.16
III.8_32	**30**	30	1194.95	**30**	30	1169.97
IV.1_32	**14**	14	1654.95	**14**	14	1613.55
IV.2_32	**15**	15	1769.08	**15**	15	1701.50
IV.3_32	**14**	14	1841.18	**14**	14	1785.02
IV.4_32	**20**	20	2132.88	**20**	20	2049.01
IV.5_32	**21**	21	1787.70	**21**	21	1726.80
IV.6_32	**11**	11	1647.42	**11**	11	1620.95
IV.7_32	**20**	20	1856.37	**20**	20	1786.50
IV.8_32	**30**	30	1995.74	**30**	30	1928.47
V.1_32	**14**	14	2167.46	**14**	14	2071.26
V.2_32	**27**	27	2587.85	**27**	27	2278.80
V.3_32	**28**	28	2495.24	**28**	28	2281.24
V.4_32	144	166	4616.67	128	155	7397.17

Best and worst solution values and total time of 20 runs for each instance.

References

[1] Baker KR. Workforce allocation in cyclic scheduling problems: a survey. Operational Research Quarterly 1976;27(1, ii):155–67.

[2] Tien JM, Kamiyama A. On manpower scheduling algorithms. SIAM Review 1992;24(3):275–87.

[3] Siferd SP, Benton WC. Workforce staffing and scheduling: hospital nursing specific models. European Journal of Operational Research 1992;60:233–46.

[4] Wren A. Scheduling, timetabling and rostering—a special relationship? In: Proceedings of the first international conference on the practice and theory of automated timetabling (ICPTAT'95), 1995. p. 474–95.

[5] Cheang B, Li H, Lim A, Rodrigues B. Nurse rostering problems—a bibliographic survey. European Journal of Operational Research 2003;151(3):447–60.

[6] Warner DM. Scheduling nursing personnel according to nursing preference: a mathematical programming approach. Operations Research 1976;24(5):842–56.

[7] Weil G, Heus K, François P, Poujade M. Constraint programming for nurse scheduling. IEEE Engineering in Medicine and Biology 1995;14:417–22.

[8] Cumming A, Paechter B, Rankin R. Post-publication timetabling. In: PATAT 2000 conference (extended abstracts), 2000. p. 107–8.

[9] Moz M, Almeida MT. Nurse scheduling—a mathematical programming based approach. Working paper 3/97, Centro de Investigação Operacional, Universidade de Lisboa; 1997.

[10] Moz M, Pato MV. Solving the problem of rerostering nurse schedules with hard constraints: new multicommodity flow models. Annals of Operations Research 2004;128:179–97.

[11] Moz M, Técnicas de Investigação Operacional Aplicadas a um Problema de Escalonamento de Pessoal em Contexto Hospitalar. PhD dissertation, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Portugal, 2003.

[12] Moz M, Pato MV. An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals of Operations Research 2003;119:285–301.

[13] Davis L. Handbook of genetic algorithms. New York: Van Nostrand Reinhold; 1991.

[14] Aickelin U, Dowsland KA. An indirect genetic algorithm for a nurse scheduling problem. Computers and Operations Research 2004;31(5):761–78.

[15] Goldberg D. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.

[16] Michalewicz Z. Genetic algorithms+data structures = evolution programs. Berlin: Springer; 1992.

[17] Bean JC. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 1994;6(2):154–60.

[18] Rudolph G. Convergence properties of canonical genetic algorithms. IEEE Transactions on Neural Networks 1994;5(1): 96–101.

[19] Reeves C, Rowe J. Genetic algorithms—principles and perspectives—a guide to GA theory. Boston: Kluwer Academic Publishers; 2003.

[20] Siegel S, Catellan Jr NJ. Nonparametric statistics for the behavioral sciences. 2nd ed., International Editions, New York: McGraw-Hill; 1988.

[21] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco: W.F. Freeman; 1979.