Appl Intell (2012) 36:442–453
DOI 10.1007/s10489-010-0270-z

A pattern recognition based intelligent search method and two assignment problem case studies

Jingpeng Li · Edmund K. Burke · Rong Qu

Published online: 31 December 2010

Abstract Numerous papers based on various search methods across a wide variety of applications have appeared in the literature over recent years. Most of these methods apply the following same approach to address the problems at hand: at each iteration of the search, they first apply their search methods to generate new solutions, then they calculate the objective values (or costs) by taking some constraints into account, and finally they use some strategies to determine the acceptance or rejection of these solutions based upon the calculated objective values. However, the premise of this paper is that calculating the exact objective value of every resulting solution is not a must, particularly for highly constrained problems where such a calculation is costly and the feasible regions are small and disconnected. Furthermore, we believe that for newly-generated solutions, evaluating the quality purely by their objective values is sometimes not the most efficient approach. In many combinatorial problems, there are poor-cost solutions where possibly just one component is misplaced and all others work well. Although these poor-cost solutions can be the intermediate states towards the search of a high quality solution, any cost-oriented criteria for solution acceptance would deem them as inferior and consequently probably suggest a rejection. To address the above issues, we propose a pattern recognition-based framework with the target of designing more intelligent and more flexible search systems. The role of pattern recognition is to classify the quality of resulting solutions, based on the solution structure rather than the solution cost. Hence, the general contributions of this work are in the line of "insights" and recommendations. Two real-world cases of the assignment problem, i.e. the hospital personnel scheduling and educational timetabling, are used as the case studies. For each case, we apply neural networks as the tool for pattern recognition. In addition, we present our theoretical and experimental results in terms of runtime speedup.

Keywords Neural networks · Assignment problem · Personnel scheduling · Exam timetabling · Search method

J. Li (✉) · E.K. Burke · R. Qu
School of Computer Science, The University of Nottingham, Nottingham, NG8 1BB, UK
e-mail: jpl@cs.nott.ac.uk

E.K. Burke
e-mail: ekb@cs.nott.ac.uk

R. Qu
e-mail: rxq@cs.nott.ac.uk

1 Introduction

The assignment problem is a fundamental combinatorial optimization problem. In its most general form, the problem is to find the best possible way of assigning a number of tasks (e.g. duties or jobs) to a number of agents (e.g. employees or machines) such that each task is assigned to exactly one agent, subject to some constraints. Martello and Toth [28], Cattrysse and Van Wassenhove [11] gave an extensive review of the problem. Fisher et al. [15] proved that the problem is NP-hard, and Osman [29] presented a survey on its many real-life applications, ranging from the well known staff assignment problem or the assignment of jobs to parallel machines, to less well known applications, such as the frequency assignment problem in satellite communications [37].

Basically, the problem consists of two parts: the underlying combinatorial structure of the assignment and an objective function modeling the "best way". Due to the NP-hard nature of the problem, existing exact algorithms are

only effective in certain instances. For more difficult highly-constrained problems, exact algorithms can only solve instances with up to a few hundred decision variables before the search trees grow prohibitively large. Hence, larger-sized complex problems are often tackled by heuristics and meta-heuristics, in an effort to obtain near optimal solutions within reasonable time. The newly-emerging hyper-heuristics [6] are often employed when raising the level of generality is an aim.

Numerous papers based on various search methods have appeared in the literature. They are largely characterized by the following broad approach. Firstly, obtain a solution, then evaluate the solution quality by an objective function which may take a variety of constraints into account. Secondly, the objective value is used to check whether the resulting solution is improved or not, so that further search directions can be determined. Most simple heuristics only accept improved solutions, while some more advanced approaches, such as tabu search [18] and simulated annealing [20], may accept non-improved solutions under certain circumstances. An introduction to different search methodologies is given in [5].

Based on our past studies on two types of staff assignment problem, namely transportation driver scheduling [25, 26] and hospital nurse rostering [7, 8], we have noticed that there might be no need to calculate the objective values for every resulting solution, particularly for highly-constrained problems, where calculations of objective values are costly and the regions containing feasible solutions are scattered and small. For example, in nurse rostering [8, 24], we found that the close neighbors of an infeasible solution are mostly infeasible, while the close neighbors of a feasible solution are very likely to be feasible. In addition, we have noticed that for the resulting solutions, there should be alternative ways to evaluate their quality apart from the conventional method of calculating the objective values. For example, when encountering schedules that look obviously infeasible or inefficient, an experienced human scheduler will never waste time to evaluate them by calculating the exact objective values, because human beings have the ability to learn from experience and make decisions by simply recognizing patterns hidden in good/bad schedules.

Mimicking the human ability to distinguish patterns, a pattern recognition system [38] can be applied here to do a similar thing: it deems a generated solution as a pattern and each of the solution components as a feature, and then infers whether the quality of this solution is good or not without calculating the detailed objective value. Such a question with a 'yes-or-no' answer, depending on the values of some input variables, can be regarded as a decision problem. If the answer is yes, then the resulting solution should be accepted; otherwise it should be rejected.

Numerous developments in neural networks have demonstrated that they are good at pattern recognition [2, 22]. Also, because of their potential ability to mimic human intelligence, neural networks have represented a popular and active research area over the past few years, with the aim that artificial intelligence systems will eventually perform some of the tasks requiring human intelligence and compensate for some of the human weakness in doing such jobs. The ability of a neural network to learn from experience and identify previously learned data, even in the presence of noise and distortion in the input pattern, has made it an excellent candidate for pattern recognition study within the context of the aims of this paper.

In this research, we apply neural networks to recognize good patterns (or assignmenting) among all the generated assignments, based on the two case studies of the real-world assignment problem (i.e. hospital staff assignment and educational exam timetable). For each case, we introduce the methods of applying neural networks, and discuss our theoretical and experimental results in terms of runtime speedup. We finally present a concluding discussion and outline some future research directions.

2 A case study of staff assignment

2.1 Problem description

The task of assignment, especially when it affects the performance of people, is a very complex endeavor. Satisfying a variety of needs and requirements while maintaining high standards for efficiency and effectiveness is often difficult and challenging. Staff assignment is the process of constructing work timetables or duty rosters for an organization. There are many types of staff assignment, such as nurse rostering [3, 10], audit staff scheduling [35], call center rostering [17], tour scheduling [1], airline crew scheduling [21] and transportation driver scheduling [23]. In this section, we implement a case study on nurse rostering to illustrate how to apply pattern recognition to the decision making of a general search system.

Nurse rostering has been a well-studied problem over the past several decades due to its complexity in scientific research and its importance in providing healthcare. For a comprehensive discussion of the various approaches that have appeared in the literature, see the survey papers by Sitompul and Randhawa [36], Cheang [12], Burke et al. [3]. In brief, this problem can be regarded as a type of staff assignment problem in which the workload (i.e. shifts of different types) needs to be assigned to nurses periodically, subject to a number of constraints. Some constraints are those generated by physical resource restrictions and legislation which must be satisfied in order to have a feasible schedule, while the other constraints represent requirements which are desirable but not obligatory. Such requirements are often used to evaluate the quality of feasible schedules.

	1							2							3							4							5			
2003 Jan	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	
Nurse 1				D	D	D	D			L	L	L	N	N				E	E	L	L	L					D	D	D	D	D	50
Nurse 2	D	D	D			E	E	E	L	L				D	L	L	L					L	L	N	N	N			E	E	E	20
Nurse 3	N	N							E	E			E	E	E									L	L	L					E	0
Nurse 4	D	D	N	N	N					E	E	E	D				E	E	E	L	L	L						E	E	E	L	20
Nurse 5				D	D	D	L	L	L				E	E	D	D	D			D	D	D	D	D			D	D	D	D		40
Nurse 6			E	E	E	L	L	L			E	E	D	D	N	N							E	E	D	D	L	L				10
Nurse 7	E	E	E					D	D	N	N	N				E	E	D	D	D	D				E	E	E	D	L	L		20
Nurse 8	L	L				L	L								E	E	D			E	E						N	N				0
Nurse 9	D	D						E	E	D			L	L									L	L				L	L			0
Nurse 10			D	L	L	L				D	D	D	L	L				D	D	L	L	N	N				E	E	D	L	L	10
Nurse 11			L	L	L	N	N				D	D	D	L	L	L				E	E	E	L	L			E	E	L	L		10
Nurse 12	L	L	L			E	E	L	L	L				D	L	L	L			D	D	D	D	D					N	N	N	20
Nurse 13	E	E	E				D	D	D	D					D	D	N	N	N			E	E	E	D	D	D			D	D	20
Nurse 14	L	L	L					E	E	E	L	L	L				L	L	L	N	N				E	E	L	L			D	20
Nurse 15				E	E	D	D	N	N						E	E	E	L	L			E	E	E	L	L	L				E	40
Nurse 16	E	E	D			E	E	D	D				E	E	D	D	D			E	E	D	D	D					E	E	L	20
																																Employees' Penalty 300

Fig. 1 An example schedule of nurse rostering. ∗ Working hours: 07:00–16:00 (*early* shift 'E'), 08:00–17:00 (*day* shift 'D'), 14:00–23:00 (*late* shift 'L'), and 23:00–07:00 (*night* shift 'N')

A large collection of test instances for the staff assignment problem are available for public download at http://www.cs.nott.ac.uk/~tec/NRP/ [8, 13], most of which relate to real-world nurse rostering. For each test instance, a spreadsheet of published results, along with at least one visualizable schedule, are provided. Figure 1 shows a typical schedule of nurse rostering. This schedule comes from the situation of intensive care units in a Dutch hospital (i.e. the ORTEC01 instance). This problem involves assigning four types of shifts (i.e. shifts of '*early*', '*day*', '*late*' and '*night*') to 16 nurses of different working contracts, within a planning period of 31 days. For a better visualization, shift types are differentiated by color, each of which covers a specific time period. For example, an *early* shift (displayed in red rectangles) covers the hours from 7:00 to 16:00.

Although there are a large number of variations on legal regulations and individual preferences depending on different countries and hospitals, typical issues in nurse rostering concern coverage demand, day-off requirements, weekend-off requirements, and minimum/maximum workforce requirements [3]. In more detail, Table 1 lists some of the most commonly-imposed constraints.

Refined heuristics or meta-heuristics are widely employed on this problem. If using the traditional cost-oriented search methodology, once a new solution x is obtained, it needs to be evaluated by taking all the constraints into account. The overall evaluation function $G(x)$ takes the form of

$$G(x) = \sum_{i=1}^{c} w_i g_i(x), \tag{1}$$

where c is the number of constraints considered, w_i is the importance weights of constraint i, and $g_i(x)$ is the evaluation function for constraint i.

Among the constraints listed in Table 1, the ones involving shift sequences or shift type succession need extra computational effort. For the purposes of illumination, we only give the formulations of a few constraints, and use the following notations:

m = number of days during the planning period;

n = number of nurses;

g = number of nurse grades;

u = number of shift types;

$x_{ijk} = 1$ if nurse i is assigned shift type k on day j, 0 otherwise;

$q_{il} = 1$ if nurse i is of grade l, 0 otherwise.

The constraint of daily demand (i.e. number of nurses) of each shift type and each grade can be illustrated as follows

$$g_1(x) = \sum_{l=1}^{g} \sum_{j=1}^{m} \sum_{k=1}^{u} \left| \sum_{i=1}^{n} x_{ijk} q_{il} - d_{jkl} \right|, \tag{2}$$

where d_{jkl} is the demand of nurses with grade l to cover type k on day j. The computational complexity of calculating the satisfaction of this constraint is

$$C_1 = (2n+2)gmu = O(gmnu). \tag{3}$$

Table 1 Constraints commonly imposed in nurse rostering problems

Constraint	Example
Daily demand on shift type and grade	4 'D' shifts of grade-II required on 02/06/2008
Max shifts per day	Max 1 shift a day
Max/min total shifts	Max 25 and/or min 16 shifts
Max/min total shift of a type	Max 10 and/or min 6 'N' shifts
Max/min consecutive shifts	Max 6 and/or min 2 consecutive shifts
Max/min consecutive shifts of a type	Max 5 and/or min 2 consecutive 'E' shifts
Shift type successions	Let '_' denote empty day. (1) D: _,D, N; (2) L: _,L, E; (3) E: _,E, N; (4) N: _,N
Min free days after a series of shifts of a type	Min 2 free days after a series of 'N' shifts
Complete weekends	No shift or two shifts in weekends
Max number of working weekends	Max 5 working weekends
Max consecutive working weekends	Max 3 consecutive working weekends
Max/min working hours between two dates	Max 80 and/or min 40 hours between 2/06/2008 and 15/06/2008
Requested shifts on/off	No 'N' shifts for nurse 6

The constraint of maximizing the number of consecutive working days can be represented by

$$g_2(x) = \sum_{i=1}^{n} \sum_{r=1}^{m-m_1} \max\left\{0, \sum_{j=r}^{r+m_1} \sum_{k=1}^{u} x_{ijk} - m_1\right\}, \tag{4}$$

where m_1 is the upper bound of consecutive working days. The computational complexity of calculating the satisfaction of this constraint is

$$C_2 = (m - m_1)(m_1 u + 2)n = O(m_1(m - m_1)nu). \tag{5}$$

The constraint of maximizing the number of consecutive shifts of individual types can be given as

$$g_3(x) = \sum_{i=1}^{n} \sum_{r=1}^{m-c_k} \sum_{k=1}^{u} \max\left\{0, \sum_{j=r}^{r+c_k} x_{ijk} - c_k\right\}, \tag{6}$$

where c_k is the upper bound of consecutive shifts of type k. The computational complexity of calculating the satisfaction of this constraint is

$$C_3 = (m - m_2)(m_2 + 2)nu = O(m_2(m - m_2)nu), \tag{7}$$

where $m_2 = \sum_{k=1}^{u} c_k/u$.

The constraint of avoiding certain shift type successions (e.g. 'D' shift followed by 'E' shift) can be formulated as

$$g_4(x) = \sum_{i=1}^{n} \sum_{j=1}^{m-1} \sum_{k_1=1}^{u} \sum_{k_2=1}^{u} \max\{0, x_{ijk_1} + x_{i(j+1)k_2} - 1\}. \tag{8}$$

The computational complexity of calculating the satisfaction of this constraint is

$$C_4 = 3(m - 1)nu^2 = O(mnu^2). \tag{9}$$

The calculations of the remaining constraints are relatively easy, each of which has a computational complexity of $O(mnu)$. Assuming the number of such constraints we need to consider is v, the complexity of calculating those easier constraints is $O(mnuv)$. Hence, the overall computational complexity for evaluating a schedule under all constraints is

$$\begin{aligned} C &= gm(2n + 2)u + (m - m_1)(m_1 u + 2)n \\ &\quad + (m_2 + 2)(m - m_2)nu + 3(m - 1)nu^2 + mnuv \\ &= O((gm + m_1(m - m_1) \\ &\quad + m_2(m - m_2) + mu + mv)nu). \end{aligned} \tag{10}$$

2.2 Pattern recognition by neural networks

The goal of this research is to find an additional way to predict the quality of solutions without calculating their actual objective values, whilst making the (meta-)heuristic search speed up or at least not slow down. We apply neural networks to achieve this goal, and in particular, we study the degree to which the evaluation by pattern recognition could be speeded up.

We use the nurse rostering problem to implement our study. A schedule can be regarded as a pattern with $m \times n$ features, where m is the number of days in the planning period and n is the number of employees to be scheduled. Each feature corresponds to a specific assignment at the i-th 'day/employee' slot denoted by a variable x_i, $i \in \{1, \ldots, m \times n\}$. A variable y is used to determine whether the quality of this schedule is 'good' or 'bad', based on the value of feature vector $(x_1, \ldots, x_{m \times n})$ rather than the objective value $G(x)$ formulated in (1). If the value of y indicates 'good', then the schedule should be accepted; otherwise it

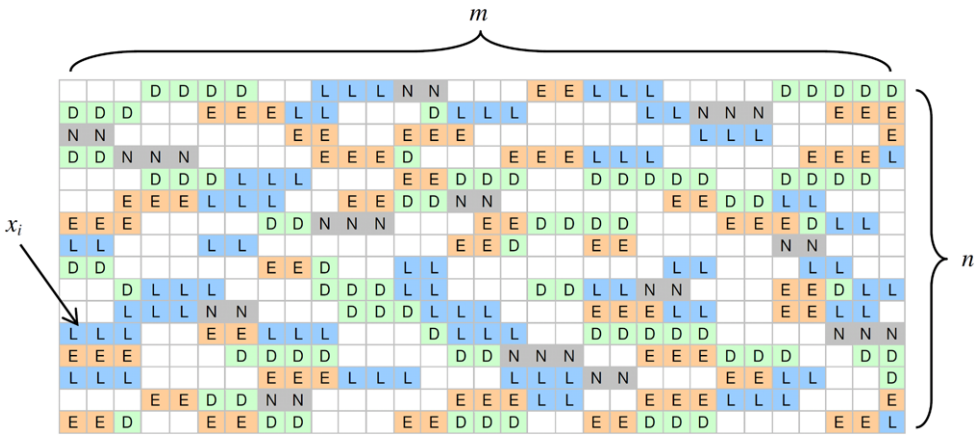

Fig. 2 The understanding of a schedule from a pattern recognition point of view

should be rejected. Figure 2 illustrates how to treat a schedule as a pattern from this point of view. For other types of assignment problems, their solutions could be expressed in similar ways.

We use neural networks to determine classifications on the resulting schedules. A neural network consists of some basic components called neurons [33]. The neurons are arranged according to a certain model, and are linked together by interconnections represented by the values of a set of weights. The network is then trained by a chosen algorithm so that a set of inputs will produce the desired output. During the training process, the weights are adjusted to better represent the relationship between the inputs and the final output. When the training is carried out, based on certain criteria, the training results of a network are represented by the final values of the weights. Once a network is properly formed and trained, it has the ability to generalize the knowledge it has learned and when a similar new instance is encountered, it can derive the appropriate result. The response of a properly trained network has the capacity to be insensitive to minor noise in its input. This generalization ability is the key reason why a neural network approach to pattern-recognition in a (meta-)heuristic search is particularly appropriate.

To develop a pattern recognition system, a set of training instances (found by whatever search methods) need to be collected first. Each instance should consist of both the input and output values. The input values correspond to the specific arrangements at 'day/employee' slots which compose a whole schedule, and the output value corresponds to the binary outcome about the overall schedule quality. The binary outcome can be obtained in the following way: comparing the objective value of each schedule with a threshold value, if the objective value is higher than the threshold, the output is 0 (rejected); otherwise the output is 1 (accepted).

There are many types of neural networks including multilayer perceptron networks, probabilistic neural networks, general regression neural networks, radial basis function networks, cascade correlation network and recurrent network, among which the multilayer perceptron network is one of the most widely used. Hence, in this paper we apply it and simply term it "neural network". The back-propagation training algorithm is frequently used for supervised learning, as it has been widely applied in pattern-recognition [16]. Note that the multilayer perception network with a back-propagation algorithm might not be the best solution.

In more detail, the training processes by a 2-layer neural network are implemented as follows:

I. *Initialization*. Set the learning rate α. Randomly initialize vector W comprising weights for all neuron connections (i.e. w_{ik} and w_k) and vector θ comprising threshold levels for neurons at the non-input layers.

II. *Provision of Training Examples*. Provide the network with an epoch of training examples denoted by $\{(x, d)\}$, with the input vector $x = (x_1, \ldots, x_{m\times n})$ applied to the input layer of neurons and the desired output d presented to the output layer consisting of only one neuron. For each training example, perform the sequence of forward and backward computation as described under points III and IV, respectively.

III. *Forward Computation*. Assume the use of a learning function denoted as φ.

i. Calculate the actual outputs of p neurons at the middle layer as

$$h_k = \varphi\left(\sum_{i=1}^{m\times n} w_{ik}x_i - \theta_k\right), \quad \text{for } k = 1, \ldots, p. \quad (11)$$

ii. Calculate the final output of the neuron at the output layer as

$$y = \varphi\left(\sum_{k=1}^{p} w_k h_k - \theta\right). \quad (12)$$

iii. Calculate the error signal as

$$e = d - y. \quad (13)$$

IV. *Backward Computation*.

i. Calculate the error gradient of the neurons at the output layer as

$$\delta = e\varphi'\left(\sum_{k=1}^{p} w_k h_k - \theta\right). \tag{14}$$

ii. Calculate the error gradients of neurons at the middle layer as

$$\delta_k = w_k \delta \varphi'\left(\sum_{i=1}^{m\times n} w_{ik} x_i - \theta_k\right). \tag{15}$$

iii. Modify W and θ of the output layer as

$$w_k = w_k + \alpha h_k \delta, \quad \text{for } k = 1, \ldots, p. \tag{16}$$

$$\theta = \theta - \alpha\delta. \tag{17}$$

iv. Modify W and θ of the middle layer as

$$w_{ik} = w_{ik} + \alpha x_i \delta_k. \tag{18}$$

$$\theta_k = \theta_k - \alpha\delta_k. \tag{19}$$

V. *Iteration*. Iterate the forward computation and backward computations under points III and IV by providing new epochs of training examples to the network until a predefined number, s, of iterations is reached.

The computational complexity of an algorithm is usually measured in terms of the number of multiplications, additions and the storage requirement involved in its implementation. A learning algorithm can be thought of as being computationally efficient when its computational complexity is polynomial in the number of adjustable parameters that are to be updated from one iteration to the next. On this basis, we can say that the training process by back-propagation is computationally efficient. This is particularly the case when using the algorithm to train a multilayer neural network, its computational complexity is linear in the total number of weights contained in the network. This important property can be verified by examining the computations involved in the forward and backward passes.

In the forward pass, the only computations involving the weights are those that relate to calculating the actual outputs of the various neurons in the network. Here we see in (11)–(13) that these computations are all linear in the weights of the network, with the computational complexity being

$$C_{11}^* = 2mnp + 3p + 3 = O(mnp). \tag{20}$$

In the backward pass, the only computations involving the weights are those that relate to the local gradient of the output and middle neurons (as shown in (14) and (15)) and the updating of the weights and thresholds themselves (as shown in (16)–(19)). We also see that these computations are all linear in the weights and thresholds of the network. Here again, the computational complexity of the backward pass is

$$C_{12}^* = 6mnp + 5p = O(mnp). \tag{21}$$

For a back-propagation algorithm with s iterations of training examples, the conclusion is therefore that the computational complexity is

$$C_1^* = (C_{11}^* + C_{12}^*)s = 8mnps + 8ps + 3s = O(mnps). \tag{22}$$

After the network is trained, the validating processes is given as follows:

I. To each network, apply its trained values in vectors W and θ.

II. Input a new pattern, i.e. vector $x' = (x'_1, \ldots, x'_{m\times n})$.

III. Calculate the output vector $(h'_1, \ldots, h'_k)$ of the middle layer as

$$h'_k = \varphi\left(\sum_{i=1}^{m\times n} w_{ik} x'_i - \theta_k\right), \quad \text{for } k = 1, \ldots, p. \tag{23}$$

IV. Calculate the output y' of the output layer as

$$y' = \varphi\left(\sum_{k=1}^{p} w_k h'_k - \theta\right). \tag{24}$$

One might notice that (23) and (24) are very similar to (11) and (12). In fact, validating on new instances is implemented in the same way as a forward pass of the back-propagation algorithm. Hence, if validating t unknown new instances, the computational complexity is

$$C_2^* = 2mnpt + 3pt + 2t = O(mnpt). \tag{25}$$

Hence, the computational complexity for the entire processes of training and validating is

$$\begin{aligned} C^* = C_1^* + C_2^* &= (8s + 2t)mnp + (8s + 3t)p + 3s + 2t \\ &= O(mnp(s + t)). \end{aligned} \tag{26}$$

It has been shown mathematically that a 2-layer neural network can accurately reproduce any differentiable function, provided the number of neurons in the hidden layer is unlimited. However, increasing the number of neurons increases the number of weights that must be estimated in the network, which in turn increases the execution time for the network. Under the situation where the execution time is an important consideration factor, a neural network without the hidden layer is often used. For such a 1-layer neural network, the

computational complexity for the training and the validating reduces to

$$C^* = C_1^* + C_2^* = O(mn(s+t)). \tag{27}$$

Compared to the cost-oriented evaluation which calculates the violations of all constraints, the evaluation of t new solutions by a 2-layer or a 1-layer neural network has the following speedup ratio:

$$r = \begin{cases} O\left(\dfrac{p(s/t+1)}{(g+m_1+m_2-(m_1^2+m_2^2)/m+u+v)u}\right), \\ \text{if using a 2-layer network;} \\ O\left(\dfrac{s/t+1}{(g+m_1+m_2-(m_1^2+m_2^2)/m+u+v)u}\right), \\ \text{if using a 1-layer network.} \end{cases} \tag{28}$$

From (28), we can also see that the computational complexity of the cost-oriented evaluation is fixed depending on the size of the problem, while the complexity of the structure-oriented evaluation by neural networks is variable depending on the parameter settings of p, s and t.

2.3 Experimental results

For the case of staff assignment, we show in the above sections that embedding neural networks, especially 1-layer neural networks, into the search may achieve runtime improvement. The availability of this approach is still based on the assumption that a neural network could find the hidden patterns in large datasets and make efficient classifications, indicated by high classification rates (i.e. the percentages of the validating samples are correctly classified). To confirm this assumption, we implement our proposed two types of neural networks on a large set of intermediate solutions (i.e. schedules) generated by a hybrid variable neighborhood approach presented in Burke et al. [8], in which 12 real world instances of nurse rostering collected from a Dutch hospital are used.

The large set of intermediate schedules is used as the sample set for training and validating. The exact objective values of all those schedules are known, as they have been calculated in Burke et al. [8]. According to these objective values, we are able to classify each of the schedules beforehand into "good" schedule or "bad" schedule corresponding to accepting the schedule or rejecting the schedule, respectively.

Table 2 lists the size of each instance, with each instance corresponding to a calendar month from January to December in 2003. The information on problem size includes the number of constraints and the number of variables appearing in its integer programming formulation. As mentioned at the beginning, the problem sizes are too large to be solved efficiently by any existing exact methods.

Table 2 also shows the details of the sample sets for training and testing, and summarizes the classification rates (i.e. the percentages of the cases that are correctly classified) for the neural networks with and without a hidden layer. For

Table 2 Problem sizes and classification results

Data	Problem size		Sample info				Classification rate		CPU time (s)		
	Constraint	Variable	|Input|[a]	|Sample|[b]	PC#[c]	PC%[d]	2-layer	1-layer	2-layer	1-layer	Normal
Jan	9206	7915	496	4000	89	74.6	88.6%	86.8%	2.86	0.53	3.83
Feb	8437	7316	448	4000	91	73.6	80.7%	79.5%	2.54	0.40	3.66
Mar	9059	7830	496	4000	99	75.5	81.3%	78.8%	2.67	0.58	3.80
Apr	8787	7616	480	4000	80	79.5	90.5%	88.4%	3.83	0.55	3.76
May	9218	7935	496	4000	95	72.4	87.9%	85.9%	5.05	0.61	3.77
Jun	8836	7641	480	4000	102	73.3	84.2%	82.7%	4.67	0.56	3.74
Jul	9090	7831	496	4000	100	74.6	83.6%	82.7%	4.31	0.59	3.78
Aug	9298	8019	496	4000	108	70.0	84.2%	80.2%	4.24	0.63	3.89
Sep	8723	7564	480	4000	83	79.6	82.8%	81.5%.	3.75	0.49	3.75
Oct	9154	7883	496	4000	96	73.6	85.9%	82.8%	4.10	0.62	3.79
Nov	9059	7830	480	4000	89	79.2	85.5%	83.9%	3.81	0.60	3.76
Dec	9026	7805	496	4000	77	78.4	84.9%	83.8%	3.34	0.61	3.73
Ave.	**8991**	**7765**	**487**	**4000**	**92**	**75.4**	**85.0%**	**83.2%**	**3.76**	**0.56**	**3.77**

[a] "|Input|"—the number of input variables (i.e. the number of 'day/employee' slots)

[b] "|Sample|"—the total number of samples used for training and testing

[c] "PC#"—the number of principal components whose eigenvalues are 1 or greater

[d] "PC%"—the percent of variance accounted for the number of PC# principal components

each problem instance, the number of samples for training and validating is fixed at 4000 (with 2000 "good" solutions and 2000 "bad" solutions calculated beforehand). For each neural network, we use a random collection of 30% samples to train the neural networks, and then use the remaining 70% samples (i.e. $t = 2800$) to validate the performance of the trained networks. To investigate the possibility of data reduction, we present the summary results of principal component analysis (see the columns of "PC#" and "PC%"). The last three columns display the CPU time (in seconds) needed for the 2-layer neural network, the 1-layer neural network and the normal method by calculating exact objective values, respectively. The computer we used for the experiments was an Intel Core 2 Duo 1.86 GHz PC with 2.0 GB of RAM.

For consistency, the parameter setting of the neural network is the same for each problem instance. For the 2-layer neural network, a fixed number of 10 neurons (i.e. $p = 10$) are set in the hidden layer. Each neural network uses a sigmoid activation function, and has the same learning rate of 0.4. The threshold for classification (i.e. distinguishing a solution being "good" or "bad") is 0.50. We set the termination condition to be a maximum 100 number of data passes of the training samples (i.e. $s = 100 \times 4000 \times 0.30$), or a minimum 0.001 relative change in training error ratio. The input variables that are constant in the training sample are excluded from the analysis.

The results in columns "PC#" and "PC%" show the potential of data reduction for our future implementation of feature extraction as, on average, 19% of the variables in the raw data account for 75% of the total variance. Table 2 also demonstrates the existence of some global patterns in the schedules of nurse rostering due to the high classification rates obtained. In general, the neural network with a hidden layer performs slightly better than the neural network without a hidden layer, with an average classification rate of 85% for the former versus 83% for the latter. Considering the number of input variables (487 on average) and the size of the sample set (4000 for all instances), the classification rate should fluctuate at 50% for all instances if no pattern could be detected. Note that the classification rates would differ if using another set for training and testing.

With respect to the CPU time, compared to the traditional evaluation method, there is no obvious runtime improvement if using the 2-layer neural network. However, there is a (roughly) 7 times speedup if using the 1-layer neural network. The experimental runtime results provide a more straightforward estimation about our theoretical result derived in (28). Of course, the neural networks would further speed up if we reduce the number of training samples and/or the number of data passes of those training samples, s. In addition, the neural network may be even faster if a smaller set of good features is successfully extracted and is used to replace the original data set.

3 Case study of educational timetabling

3.1 Problem description

Timetabling has attracted a significant level of research interest since the 1960's. The general timetabling problem comes in many different guises such as sports timetabling [14], transportation timetabling [23] and educational timetabling [4, 27, 30–32, 34]. Educational timetabling problems are probably the most widely studied. This is partly because it is one of the most important administrative activities that take place several times a year in all academic institutions.

A general exam timetabling problem can be considered to be the process of assigning a set of events (i.e. exams) into a limited number of timeslots subject to a set of constraints. In doing this, some constraints must be satisfied under any circumstances (so called *hard constraints*). A typical example is when two exams with common students involved cannot be scheduled into the same timeslot. In addition, there is also a set of desirable constraints (so called *soft constraints*), which may be violated if necessary. A typical example is when exams taken by common students should be spread out over the available timeslots so that students do not have to sit two exams that are too close to each other. Solutions with no violations of hard constraints are called *feasible* solutions. How much the soft constraints are satisfied gives an indication of how good the solutions (timetables) are.

To facilitate our time complexity analysis on the above algorithm, we use the following notations:

m—number of exams;

n—number of timeslots for scheduling all exams;

S—number of students;

S_i—set of students registered to exam i;

E_k—set of exams to which student k enrolled;

$\mathbf{A}$—an $m \times m$ matrix where $\mathbf{A}_{i,j} = 1$ if exam i conflicts with exam j, $\mathbf{A}_{i,j} = 0$ otherwise;

$\mathbf{A}'$—an $m \times m$ matrix where $\mathbf{A}'_{i,j} = |S_i \cap S_j|$ if exam i conflicts with exam j (i.e. the number of students that take both exam i and exam j), $\mathbf{A}'_{i,j} = 0$ otherwise;

α—conflict density of $\mathbf{A}$, calculated as $\alpha = \sum_{i=1}^{m}\sum_{j=1}^{m} A_{i,j}/m^2$ (i.e. the percentage of the non-zero elements in $\mathbf{A}$);

$\mathbf{B}$—an $n \times m$ matrix where $\mathbf{B}_{i,j} = 1$ if timeslot i is assigned to exam j, $\mathbf{B}_{i,j} = 0$ otherwise. $\mathbf{B}$ represents a complete solution, satisfying $m = \sum_{i=1}^{m}\sum_{j=1}^{n} B_{i,j}$.

The following objective function is used in Burke et al. [9] and many other papers in the literature to calculate the cost of an obtained feasible solution x:

$$\text{Minimize} \quad G(x) = \left(\sum_{i=0}^{4} w_i \times N_i\right) \Big/ S, \tag{29}$$

where $w_i = 2^{4-i} | i \in \{0, \ldots, 4\}$ is the weight that represents the importance of scheduling exams with common students

of i timeslots away in n, $N_i|i \in \{0, \ldots, 4\}$ is the number of students that sit two exams of i timeslots away.

Deriving the time complexity of evaluating a feasible candidate solution by (1) is not straightforward. To facilitate this, we introduce another ancillary $m \times m$ matrix $\mathbf{D}$, where

$$\mathbf{D}_{i,j} = \begin{cases} 2^{4-t}, & \text{if } 0 \le t \le 4; \\ \infty, & \text{otherwise.} \end{cases} \tag{30}$$

t denotes the timeslot distance between exam i and exam j in matrix $\mathbf{B}$. Then

$$t = |i_1 - i_2| - 1, \tag{31}$$

satisfying $B_{i_1,i} = 1$ and $B_{i_2,j} = 1$. Hence, (29) can be rewritten as

$$\text{Minimize} \quad G'(x) = \sum_{i=1}^{m} \sum_{j=1}^{m} (D_{i,j} \times A'_{i,j})/S. \tag{32}$$

Once a new solution is generated, the values in matrix $\mathbf{D}$ (i.e. $D_{i,j}$) need to be updated, while the values in matrix $\mathbf{A}'$ are constant which means we can calculate and store these values in advance. The complexity to calculate the t value in (30) is determined by the complexity of identifying i and j in (31), which is between 1 (best case) and n (worst case) depending on the solutions generated. Hence, in our following complexity analysis for the timetabling problem, we do not calculate the exact number of operations and only discuss the situation of the average case.

On average, the complexity of finding the t value in (30) is $O(n)$, and the complexity of (32) is

$$C_S = O(m^2 n). \tag{33}$$

However, in most real-world instances, there is a certain number of elements in matrix $\mathbf{A}'$ with '0'-values. Obviously the density of $\mathbf{A}'$ is α, the same as that of matrix $\mathbf{A}$. Hence, we only need to calculate the (i, j)-pairs of $\mathbf{D}$, where $\mathbf{A}'_{i,j} \neq 0$. We can reduce the complexity of evaluating a resulting solution to

$$C_S = O(\alpha m^2 n). \tag{34}$$

3.2 Pattern recognition by neural networks

We apply neural networks to study the possibility of predicting the quality of timetabling solutions without calculating their actual objective values. We also study the degree to which the evaluation by neural networks could be speeded up.

For the timetable problem, a schedule can be regarded as a pattern with m features, where m is the number of exams to be scheduled. Each feature corresponds to an assignment of exam i at a specific time slot, denoted by a variable $x_i \in \{1, \ldots, n\}|i \in \{1, \ldots, m\}$ where n is the number of available time slots. A dependent variable y is used to determine whether the quality of this schedule is 'good' or 'bad', based on the value of feature vector $(x_1, \ldots, x_m)$ rather than the objective value $G(x)$ formulated in (29). If the value of y indicates 'good', then the schedule should be accepted; otherwise it should be rejected.

The neural network approach used to classify the timetabling solutions is trained via the standard 2-layer backpropagation algorithm described in Sect. 2.2. For a 2-layer network with s iterations of training examples, the computational complexity is

$$C_1^* = O(mps), \tag{35}$$

where p is the number of neurons in the middle layer.

When validating t unknown new instances, the computational complexity is

$$C_2^* = 2mpt + 3pt + 2t = O(mpt). \tag{36}$$

Hence, the computational complexity for the entire processes of training and validating is

$$C^* = C_1^* + C_2^* = O(mp(s + t)). \tag{37}$$

Obviously, if using a 1-layer neural network, the computational complexity for the training and the validating reduces to

$$C^* = C_1^* + C_2^* = O(m(s + t)). \tag{38}$$

Compared to the cost-oriented evaluation which calculates the violations of all constraints, the evaluation of t new solutions by a 2-layer or a 1-layer neural network with s iterations of training samples has the following speedup rate:

$$r = \begin{cases} \dfrac{O(mp(s+t))}{O(\alpha m^2 nt)} = O\left(\dfrac{p(1+s/t)}{\alpha mn}\right), \\ \quad \text{if using a 2-layer network;} \\ \dfrac{O(m(s+t))}{O(\alpha m^2 nt)} = O\left(\dfrac{1+s/t}{\alpha mn}\right), \\ \quad \text{if using a 1-layer network.} \end{cases} \tag{39}$$

From (39), we can still find that the computational complexity of the solution evaluation is fixed depending on the problem size, while the complexity of the structure-oriented evaluation by neural networks is variable depending on the network parameter settings.

3.3 Experimental results

For the exam timetabling problem, we show in (39) that embedding neural networks into the search may achieve significant runtime improvement, especially for a simple neural

Table 3 Problem characteristics and classification results

Data	Problem size			Sample info				Classification rate		CPU time (s)		
	Exam	Slot	Density	|Input|[a]	|Sample|[b]	PC#[c]	PC%[d]	2-layer	1-layer	2-layer	1-layer	Normal
car91	682	35	0.13	682	4000	171	62.4	95.9	94.3	52.88	6.87	221.39
car92	543	32	0.14	543	4000	141	60.6	95.8	95.6	40.75	5.37	142.82
ear83	190	24	0.27	190	4000	49	53.8	94.5	93.8	10.07	1.34	22.05
hec92	81	18	0.42	81	4000	20	51.9	96.5	95.2	0.31	0.06	5.20
kfu93	461	20	0.06	461	4000	112	59.6	94.8	96.3	16.75	2.25	27.38
Lse91	381	18	0.06	381	4000	90	58.4	96.0	94.1	12.43	1.28	18.08
sta83	139	13	0.14	139	4000	47	59.1	95.2	94.9	7.91	0.18	5.17
tre92	261	23	0.18	261	4000	53	53.8	91.4	89.6	16.80	0.16	28.72
uta93	622	35	0.13	622	4000	156	61.7	88.9	88.8	47.13	5.46	170.38
ute92	184	10	0.08	184	4000	53	57.9	92.8	92.6	4.08	0.14	5.56
yor83	181	21	0.29	181	4000	35	51.5	94.1	93.9	10.15	1.17	20.62
Ave.	**339**	**23**	**0.17**	**339**	**4000**	**84**	**57.3**	**94.1**	**93.6**	**19.93**	**2.21**	**60.67**

[a] "|Input|"—the number of independent variables, i.e. the number of exams

[b] "|Sample|"—the number of samples or cases

[c] "PC#"—the number of principal components whose eigenvalues are 1 or greater

[d] "PC%"—the percent of variance accounted for the number of PC# principal components

network without the hidden layer. The availability of this approach is based on the assumption that neural networks could find the hidden pattern in the solutions (i.e. the time tables) and make efficient classifications. In this section, we verify the above assumption experimentally, based on the solutions of a set of internationally accepted benchmark exam timetabling problems. These problems are real-world problems that have been tested by many approaches in the literature (see [32]).

Table 3 displays the problem characteristics (including the number of exams, the number of available time slots and the problem conflict density) and the classification results by two types of neural networks (i.e. with and without the hidden layer) on 11 exam timetabling problems. The problem size ranges from 81 to 682 exams and from 10 to 35 time slots. The density of the conflict matrix, i.e. the ratio of the number of conflicting exams over the overall number of exams, ranges from 0.06 to 0.42. For each neural network, the number of independent variables is equal to the number of exams, and the number of cases for training and validating is fixed at 4000 (with 2000 "good" solutions and 2000 "bad" solutions whose exact objective values are calculated beforehand). The last three columns display the CPU time (in seconds) needed for the 2-layer neural network, the 1-layer neural network and the normal evaluation method without the aid of any neural network, respectively. The computer used for our experiments was an Intel Core 2 Duo 1.86 GHz PC with 2.0 GB of RAM.

Columns 7 and 8 (i.e. columns "PC#" and "PC%") display the summary results of principal component analysis. On average, 25% of the variables in the raw data account for 57% of the total variance. Although the results are not as prominent as the ones in Table 2 for the nurse rostering problem, they still indicate some potential for implementing a data reduction process (i.e. feature extraction). Columns 9–10 list the classification results generated by the two types of neural networks, and we use the same neural network parameters as in Sect. 2.3.

In terms of classification rate, the 2-layer neural network and the 1-layer neural network have similar performance, and they all have an average rate as high as 94%. The results show that the neural networks are very efficient in classifying the solutions of exam timetabling problems without undertaking the calculation of exact objective values for every resulting solution. In terms of the CPU time, evaluation by the 2-layer neural network is roughly 3 times faster than the traditional evaluation method, while the 1-layer one is nearly 30 times faster. The experimental runtime results are in line with our theoretical result derived in (39).

4 Conclusions and future research

For most assignment problems, the solutions are comprised of components (i.e. unit assignments). This means that pattern recognition could be applied to increase the level of intelligence when designing an automatic search system. The primary goal of pattern recognition is to make supervised classifications, in which input patterns (i.e. resulting solutions) can be identified as a member of predefined classes

(i.e. 'good' or 'bad' in this context). Hence, pattern recognition provides an alternative way to evaluate the quality of solutions without calculating their exact objective values.

Experimental results on two difficult real-world assignment problems, the hospital personnel scheduling problem and the educational timetabling problem, demonstrates the efficiency of the neural network as a pattern recognition tool to make classifications. Complexity analysis based on the above two problems confirms that for highly-constrained problems, pattern recognition by neural networks (especially the single-layer neural network) could speed up the search process significantly. It is also suggested that the supposition could be extended to other assignment problems, although the degrees to which patterns could be recognized would vary depending on the types of considered problem and the sample set.

The major contribution of our work is that, as far as we are aware, this is the first time that the idea of using pattern recognition to evaluate the quality of solutions has been presented. This idea is problem-independent, and could be incorporated to any heuristic, meta-heuristic or hyper-heuristic search.

Another contribution of our work is that, to deal with the numerous solutions generated during the search, we propose a new idea of structure-oriented evaluation as a useful addition to the traditional cost-oriented evaluation. The idea is based on the observation that, due to the nature of combinatorial problems, there are many poor-cost solutions where maybe just one component is misplaced and all others work well. Under these circumstances, the cost-based selection operators would regard these solutions as inferior and consequently suggest rejecting them even when they contain outstanding building blocks. Hence, these seemingly inferior solutions may not have an appropriate chance of surviving during the search.

Our work opens a wide area for future research. Firstly, we are looking at the implementation of a feature extraction phase [19] to further speed up the classification process. Pattern recognition normally consists of two phases of feature extraction and pattern classification, and in this research we only consider the latter phase. For large assignment problems, computational difficulties exist if all possible features are used for learning directly.

Secondly, this work sheds light on the development of more flexible and more intelligent decision support systems. With the aid of pattern recognition, a general search system could be developed in such a way: once new data (or new patterns) are accumulated to a certain amount, a "pattern recognition" processor would be invoked automatically. The lifecycle of the processor and percentage of patterns entering the processor could be controlled by a number of parameters, which can be adjusted adaptively or by an evolutionary approach depending on the changing situations during the search. With the search in progress, the processor would be suspended while the search stays in the regions with similar landscapes. It would be revived when the search enters the promising regions with different landscapes, and would be refreshed when the search has obtained some new elite solutions.

Acknowledgements The work was funded by the UK's Engineering and Physical Sciences Research Council (EPSRC), under grant EP/D061571/1.

References

1. Alfares HK (2004) Survey, categorization, and comparison of recent tour scheduling literature. Ann Oper Res 127:145–175
2. Bishop CM (2005) Neural networks for pattern recognition. Oxford University Publisher, Oxford
3. Burke EK, De Causmaecker P, Vanden Berghe G, Landeghem H (2004) The state of the art of nurse rostering. J Sched 7:441–499
4. Burke EK, De Werra D, Kingston J (2004) Applications to timetabling. In: Cross J, Yellen J (eds) Handbook of graph theory. Chapman Hall/CRC Press, London/Boca Raton, pp 445–474. Sect. 5.6
5. Edmund EK, Graham K (eds) (2005) Search methodologies: introductory tutorials in optimization and decision support techniques. Springer, Berlin
6. Burke EK, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of meta-heuristics. Kluwer Academic, Norwell, pp 457–474
7. Burke EK, Li J, Qu R (2010) A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. Eur J Oper Res 2003:484–493
8. Burke EK, Curtois T, Post G, Qu R, Veltman B (2008) A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. Eur J Oper Res 188:330–341
9. Burke EK, McCollum B, Meisel A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. Eur J Oper Res 176:177–192
10. Burke EK, Cowling P, De Causmaecker P, Vanden Berghe G (2001) A memetic approach to the nurse rostering problem. Appl Intell 15:199–214
11. Cattrysse D, Van Wassenhove LN (1992) A survey of algorithms for the generalized assignment problem. Eur J Oper Res 60:260–272
12. Cheang B, Li H, Lim A, Rodrigues B (2003) Nurse rostering problems—a bibliographic survey. Eur J Oper Res 151:447–460
13. Curtois T (2007) Novel heuristic and metaheuristic approaches to the automated scheduling of healthcare personnel. PhD thesis, School of Computer Science, University of Nottingham
14. Easton K, Nemhauser G, Trick M (2004) Sports scheduling. In: Leung J (ed) Handbook of scheduling: algorithms, models, and performance analysis, Chap 52. CRC Press, Boca Raton
15. Fisher M, Jaikumar R, Van Wassenhove L (1986) A multiplier adjustment method for the generalized assignment problem. Manag Sci 32:1095–1103
16. Haykin S (1998) Neural networks: a comprehensive foundation, 2nd edn. Prentice Hall, New York
17. Gans N, Koole G, Mandelbaum A (2003) Telephone call centers: tutorial, review, and research prospects. Manuf Serv Oper Manag 5:79–141
18. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Norwell

19. Guyon I, Gunn S, Nikravesh M, Zadeh LA (2006) Feature extraction: foundations and applications. Springer, Berlin
20. Kirkpatrick S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680
21. Kohl N, Karisch SE (2004) Airline crew rostering: problem types, modeling, and optimization. Ann Oper Res 127:223–257
22. Kulkarni AD, Cavanaugh CD (2000) Fuzzy neural network models for classification. Appl Intell 12:207–215
23. Kwan RSK (2004) Bus and train driver scheduling. In: Leung J (ed) Handbook of scheduling: algorithms, models, and performance analysis, Chap 51. CRC Press, Boca Raton
24. Li JJ, Aickelin U, Burke EK (2009) Self-adjusting search for hospital personnel scheduling. INFORMS J Comput 21:468–479
25. Li J, Kwan RSK (2003) A fuzzy genetic algorithm for driver scheduling. Eur J Oper Res 147:334–344
26. Li J, Kwan RSK (2005) A self-adjusting algorithm for driver scheduling. J Heuristics 11:351–367
27. Mansour N, Isahakian V, Ghalayini I (2009) Scatter search technique for exam timetabling. Appl Intell. doi:10.1007/s10489-009-0196-5
28. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, New York
29. Osman IH (1995) Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches. OR Spektrum 17:211–225
30. Petrovic S, Burke EK (2004) University timetabling. In: Leung J (ed) Handbook of scheduling: algorithms, models, and performance analysis, Chap 45. CRC Press, Boca Raton
31. Qu R, Burke EK (2009) Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. J Oper Res Soc 60:1273–1285
32. Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2009) A survey of search methodologies and automated system development for examination timetabling. J Sched 12:55–89
33. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds) Parallel distributed processing: explorations in the microstructure of cognition, vol 1. MIT Press, Cambridge, pp 318–362
34. Schaerf A (1999) A survey of automated timetabling. Artif Intell Rev 13:87–127
35. Salewski F, Bottcher L, Drex LA (1996) Operational audittask assignment and staff scheduling. OR Spektrum 18:29–41
36. Sitompul D, Randhawa S (1990) Nurse scheduling models: a state-of-the-art review. J Soc Health Syst 2:62–72
37. Salcedo-Sanz S, Bousoño-Calzón C (2005) A hybrid neural-genetic algorithm for the frequency assignment problem in satellite communications. Appl Intell 22:207–217
38. Theodoridis S, Koutroumbas K (2006) Pattern recognition. Academic Press, San Diego