Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results

Kate A. Smith[1*], David Abramson[2], and David Duke[2]

[1]School of Business Systems, and
[2]School of Computer Science and Software Engineering
P. O. Box 63B, Monash University
Victoria 3800
Australia

Abstract

This paper considers the use of discrete Hopfield neural networks for solving school timetabling problems. Two alternative formulations are provided for the problem: a standard Hopfield-Tank approach, and a more compact formulation which allows the Hopfield network to be competitive with swapping heuristics. It is demonstrated how these formulations can lead to different results. The Hopfield network dynamics are also modified to allow it to be competitive with other metaheuristics by incorporating controlled stochasticities. These modifications do not complicate the algorithm, making it possible to implement our Hopfield network in hardware. The neural network results are evaluated on benchmark data sets and are compared to results obtained using greedy search, simulated annealing and tabu search.

Keywords: Hopfield neural networks, timetabling, combinatorial optimization, simulated annealing, tabu search

[*] Corresponding author: Associate Professor Kate Smith, School of Business Systems, P. O. Box 63B, Monash University, Victoria 3800, Australia. Telephone: +61 3 9905 5800, Fax: +61 3 9905 9422, email: kate.smith@infotech.monash.edu.au

1. Introduction

For well over a decade, researchers have been trying to use Hopfield neural networks to solve difficult combinatorial optimisation problems (Smith, 1999). The idea of using Hopfield neural networks for this purpose initially seemed promising (Hopfield & Tank, 1985), but was questioned several years later when the initial results could not be reproduced and some severe limitations of the approach were reported (Wilson & Pawley, 1988). Hopfield networks, in their original form, suffer from being a gradient descent technique incapable of escaping local minima, and are further limited by the fact that their penalty parameter approach to solving constrained optimisation problems frequently results in infeasible or poor quality solutions. Their advantages however include hardware implementability which can potentially lead to rapid solutions if these limitations can be overcome. Over the last decade researchers have modified the approach in an attempt to overcome these limitations (Van den Bout & Miller, 1989; Kamgar-Parsi & Kamgar-Parsi, 1987; Smith, Palaniswami & Krishnamoorthy, 1996), but the field has never truly recovered from its controversial beginnings. In fact, Osman & Laporte (1996) stated in a survey of metaheuristic techniques that while neural networks are a powerful technique for solving prediction, classification and pattern recognition problems, they "are not competitive with the best metaheuristics from the operations research literature when applied to combinatorial optimisation problems".

Recent developments in the field have demonstrated that modified Hopfield neural networks can complete effectively with metaheuristics when solving practical combinatorial optimisation problems (Smith, Palaniswami & Krishnamoorthy, 1998; Papageorgiou, Likas & Stafylopatis, 1998). That is, by incorporating hill-climbing modifications, the quality of the solutions produced by the Hopfield network can match and even outperform solutions found by metaheuristic techniques such as simulated annealing. Unfortunately, such modified Hopfield networks tend not to be competitive in terms of computation effort compared to other metaheuristics. Further, many of the modifications that have been made to the Hopfield network have not retained the efficient hardware implementability of the original model, eliminating the possibility of future computational efficiency though hardware implementation.

This paper proposes a new modified Hopfield neural network that is competitive with metaheuristics in terms of both solution quality and computational effort. In addition, the modifications are fully implementable in hardware, enabling future gains in computational efficiency to be attained. The competitiveness of this Hopfield network is demonstrated by solving a set of benchmarked timetabling problems.

Timetabling is a classical operations research problem that finds broad application in many areas of business, industry, and public administration. While most research has focused on university lecture and examination timetabling (Mehta, 1981; White & Haddad, 1983), no doubt due to the involvement of operations research academics in the timetabling activities of their own institutions, there are many other practical application areas of timetabling including staff rostering (Bianco, Bielli, Mingozzi & Ricciardelli, 1992; Hsu, Lin & Hu, 1994), school timetabling (Abramson, 1991; Wright, 1996), interview scheduling (Tsuchiya & Takefuji, 1997; Mausser, Magazine & Moore, 1996), and many other constraint satisfaction problems. From its simplest form as a class-teacher problem through to complex rostering problems, timetabling has also served as a useful benchmark problem for testing metaheuristics due to its

NP-complete nature, and its relationship to the broad class of generalised quadratic assignment problems.

This paper proposes the use of Hopfield neural networks for solving school timetabling problems. Previous studies using neural networks for timetabling problems are surprisingly scarce, no doubt a consequence of the poor reputation neural networks have had for solving the well-studied Travelling Salesman Problem (Smith, 1999). In fact, the two recent volumes of Practice and Theory of Automated Timetabling (PATAT) conference proceedings (Burke & Ross, 1996; Burke & Carter, 1998) contain no papers applying neural networks to any timetabling problem. Aside from the contributions this paper makes to the neural network literature, as discussed above, this paper therefore makes an important contribution to the available literature on timetabling. We show how two different formulation of a school timetabling problem can be mapped onto the Hopfield neural network, and demonstrate that the performance of the Hopfield network depends on the formulation used. Comparisons with other metaheuristic approaches including simulated annealing and tabu search are also conducted, where solution quality and computation time of each technique are examined.

In its most general form the school timetabling problem involves the assignment of a teacher, class and room to a particular time-slot or period in such a way that no person is assigned to more than one room at any time, and no room has more than one class and teacher in it at any time. Such a timetable is considered to be feasible meaning clash-free, but not necessarily ideal. There could also be a number of context-specific constraints which may need to be considered for the timetable to become operational.

Examples of such constraints will be considered later in this paper. This general framework encompasses school timetabling as well as course and examination scheduling in universities. The underlying assumption in this simple model is that we are timetabling classes rather than individual students. A class is defined as a group of students studying an identical course of subjects and who are never separated, so that they move as a group and can be considered as a single entity.

The simplest form of school timetabling problem, the class-teacher model as presented by de Werra (1985), is as follows:

Let C be a set of C classes and T be a set of T teachers. We are given a C × T matrix of requirements **R** = (R_{ij}) where R_{ij} is the number of periods teacher j is required to meet class i for the duration of the timetable (containing a total of P periods in a set P). Suppose we define a boolean variable

$$x_{ijk} = \begin{cases} 1 \text{ } \textit{if class i and teacher j are assigned to period k} \\ 0 \text{ } \textit{otherwise} \end{cases} \quad (1)$$

with the additional condition that each period is of equal duration. This simple model is then equivalent to solving the constraint satisfaction problem (TT1):

$$\sum_{k=1}^{P} x_{ijk} = R_{ij} \quad (\forall i \in \mathsf{C}; j \in \mathsf{T}), \quad (2)$$

$$\sum_{j=1}^{T} x_{ijk} \le 1 \quad (\forall i \in \mathsf{C}; k \in \mathsf{P}), \quad (3)$$

$$\sum_{i=1}^{C} x_{ijk} \le 1 \quad (\forall j \in \mathsf{T}; k \in \mathsf{P}), \quad (4)$$

$$x_{ijk} = 0 \text{ } or \text{ } 1 \quad (\forall i \in \mathsf{C}; j \in \mathsf{T}; k \in \mathsf{P}). \quad (5)$$

Equation (2) ensures that all requirements are satisfied, while equations (3) and (4) prevent class and teacher clashes respectively. Equation (5) is required to ensure the integrality of the solution.

To this model, several researchers have associated a bipartite multigraph G = $(C, T, \hat{R})$ and used edge-colouring techniques to find conflict-free assignments (de Werra, 1985; Schreuder & van der Velde, 1984). The vertices of the graph are the classes and teachers, while each vertex c_i and vertex t_j are linked by R_{ij} parallel edges. The problem is equivalent to assigning a colour to each edge of G (from a choice of P colours) in such a way that no two adjacent (or parallel) edges have the same colour. Network flow techniques (Krarup, 1982) have been used to solve such simple class-teacher problems exactly for small sized graphs. It has been shown however, that such problems are NP-complete (Even, Itai & Shamir, 1976) and so network flow and edge-colouring techniques are quite unsuitable for solving larger sized class-teacher problems.

Naturally, when we increase the dimensions and complexity of the problem by considering room allocations and other realistic constraints, exact approaches must be replaced by heuristic approaches in order to obtain a useful solution for a reasonable amount of computational effort. Various metaheuristic techniques have been used with great success over the last decade or so, including simulated annealing (Abramson, 1991; Abramson & Dang, 1993; Elmohamed, Coddington & Fox, 1998; Dowsland, 1998), tabu search (Dowsland, 1998; Costa, 1994; Hertz, 1991), constraint logic programming (Kang & White, 1992; White & Zhang, 1998; Gueret, Jussien, Boizumault & Prins, 1996), genetic algorithms (Ross & Corne, 1995; Burke, Elliman & Weare, 1995; Ross, Hart & Corne, 1998; Erben & Keppler, 1996), and other heuristic search procedures (Wright, 1996; Ferland & Lavoie, 1992). While a few of these previous studies use theoretical benchmark data sets designed to evaluate the comparative power of different techniques (Abramson & Dang, 1993; Ferland & Lavoie, 1992), most are concerned with proving the power of a certain technique by solving realistic school and university timetabling problems (Abramson, 1991; Wright, 1996; Elmohamed et al., 1998; Costa, 1994; Hertz, 1991; Kang & White, 1992; Ross & Corne, 1995; Burke et al., 1995; Nepal, Melville & Ally, 1998). We refer the interested reader to Carter and Laporte (1998) for a detailed survey of timetabling problems and metaheuristic approaches to solving them.

The approach taken in this paper is to use the same benchmark data sets used by Abramson and Dang (1993), available from the OR-Library at Imperial College London, to compare the performance of our modified neural networks with other metaheuristic approaches for timetabling problems with room allocations. Neural networks have the advantage over these alternative approaches that they are an inherently parallel approach, designed to be accelerated by hardware implementation. Our previous work has indicated that the speed-up obtained when implementing Hopfield neural networks in hardware is at least one order of magnitude over an optimised software implementation (Abramson, Smith, Logothetis & Duke, 1998). While the present study does not include any acceleration of the neural network performance through hardware implementation, the potential of the approach in this regard is the motivation for studying the comparative performance of the Hopfield network against other metaheuristic techniques. If the neural network results can be shown to be comparable in quality to the best metaheuristic approaches, then the investment in hardware implementation of the neural network is justified.

Previous research in applying neural networks to timetabling (Gislen, Peterson & Soderberg, 1989; Mausser & Magazine, 1996; Pellerin & Herault, 1994; Kovacic, 1993; Lim & Loe, 1991; Gianoglio, 1990) has been inconclusive when attempting to address this research question, since these studies have not compared their results to other metaheuristic approaches. The neural network approach considered here is also different to those used in the previous studies. This difference stems from the alternative formulation used to encode the problem, as well as the (fully hardware implementable) modifications that have been made to the standard Hopfield network dynamics to ensure competitiveness with metaheuristic techniques such as simulated annealing.

The contributions of this paper are thus two-fold. To the neural network literature we present a modified Hopfield network, that retains its hardware implementability, and is shown to be competitive with metaheuristics on a benchmarked combinatorial optimisation problem. This competitiveness is measured in terms of both solution quality and computational efficiency. The advantages of the new model are demonstrated by comparing the Hopfield network with and without the proposed modifications. To the timetabling literature we contribute a study showing how two different formulations of a school timetabling problem can be mapped onto an effective Hopfield neural network, and yield different results. This study also contributes a comparative analysis with metaheuristic techniques which is rarely present in other papers applying neural network to timetabling problems. Extensions to other timetabling problems including side constraints such as teacher preferences, availabilities, and room capacities are also considered.

2. Timetabling Formulations

In this section we expand the simple class-teacher problem described in Section 1 to consider venue or room allocation. Two formulations are presented to model the class-teacher-venue problem: one using a logical extension of the class-teacher formulation (TT1), and another which reformulates the problem using the ideas of swapping heuristics. The result is a more compact second formulation which is intended to be more competitive with other metaheuristic such as simulated annealing that can be designed to swap static rows of a solution matrix, rather than flip elements of a solution matrix.

Let V be a set of V venues (rooms) and C and T be the set of classes and teachers as before. The requirements matrix is now of dimension C × T × V where R_{ijk} is the number of periods teacher j is to meet class i in venue k for the duration of the timetable. Suppose we define a boolean variable

$$x_{ijkl} = \begin{cases} 1 \text{ } \textit{if class i and teacher j are assigned to venue k in period l} \\ 0 \text{ } \textit{otherwise} \end{cases} \qquad (6)$$

Then the formulation can be extended to include venues by satisfying the constraints (TT2):

$$\sum_{l=1}^{P} x_{ijkl} = R_{ijk} \quad (\forall i \in \mathsf{C}; j \in \mathsf{T}; k \in \mathsf{V}), \tag{7}$$

$$\sum_{j=1}^{T}\sum_{k=1}^{V} x_{ijkl} \leq 1 \quad (\forall i \in \mathsf{C}; l \in \mathsf{P}), \tag{8}$$

$$\sum_{i=1}^{C}\sum_{k=1}^{V} x_{ijkl} \leq 1 \quad (\forall j \in \mathsf{T}; l \in \mathsf{P}), \tag{9}$$

$$\sum_{i=1}^{C}\sum_{j=1}^{T} x_{ijkl} \leq 1 \quad (\forall k \in \mathsf{V}; l \in \mathsf{P}), \tag{10}$$

$$x_{ijkl} = 0 \; or \; 1 \quad (\forall i \in \mathsf{C}; j \in \mathsf{T}; k \in \mathsf{V}; l \in \mathsf{P}). \tag{11}$$

This formulation is a logical extension of the class-teacher model (TT1), so that equation (7) ensures the requirements are satisfied, while equations (8), (9), and (10) prevent class, teacher, and venue clashes respectively. Again, equation (11) is required to ensure the integrality of the solutions. Unfortunately, the dimensions of this formulation are rather large for practical sized problems. For this reason, techniques that use this formulation are unlikely to be competitive with heuristics that treat the problem as one of mapping a triplet (class, teacher, venue) to a period. This formulation is nonetheless the one arrived at when extending the approach of Hopfield and Tank (1985) to solve the class-teacher-venue problem.

In order to generate neural network results that are competitive with simulated annealing and other approaches that have used the idea of swapping the positioning of triplets in a timetable (Abramson, 1991), we now propose an alternative timetabling formulation that is similar in concept, and better suited for mapping onto a Hopfield neural network. Suppose there are τ triplets that need to be timetabled across the set of P periods. Each triplet takes the form *class i meets teacher j in venue k.* Each unique triplet will appear R_{ijk} times in the set of all triplets $\Im$, so that the total number of triplets is determined by:

$$\tau = \sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V} R_{ijk} \tag{12}$$

The task of assigning each triplet in the set $\Im$ to a unique period can then be formulated using the boolean variable:

$$x_{tp} = \begin{cases} 1 & \textit{if triplet t is assigned to period p} \\ 0 & \textit{otherwise} \end{cases} \tag{13}$$

In order to identify possible clashes, we need to be able to determine if a given class, teacher, or venue appear in a given triplet. Thus we define the constant matrices $\hat{\mathbf{C}}, \hat{\mathbf{T}}, \text{and } \hat{\mathbf{V}}$, which function as look-up tables:

$$\hat{C}_{it} = \begin{cases} 1 & \textit{if class i is in triplet t} \\ 0 & \textit{otherwise} \end{cases} \tag{14}$$

$$\hat{T}_{jt} = \begin{cases} 1 & \textit{if teacher j is in triplet t} \\ 0 & \textit{otherwise} \end{cases} \tag{15}$$

$$\hat{V}_{kt} = \begin{cases} 1 & \textit{if venue k is in triplet t} \\ 0 & \textit{otherwise} \end{cases} \tag{16}$$

The solution matrix **x** then defines a valid solution to the timetabling problem if the following constraints are satisfied (TT3):

$$\sum_{p=1}^{P} x_{tp} = 1 \quad (\forall t \in \Im), \tag{17}$$

$$\sum_{t=1}^{\tau} x_{tp}\hat{C}_{it} \leq 1 \quad (\forall i \in \mathsf{C}; p \in \mathsf{P}), \tag{18}$$

$$\sum_{t=1}^{\tau} x_{tp}\hat{T}_{jt} \leq 1 \quad (\forall j \in \mathsf{T}; p \in \mathsf{P}), \tag{19}$$

$$\sum_{t=1}^{\tau} x_{tp}\hat{V}_{kt} \leq 1 \quad (\forall k \in \mathsf{V}; p \in \mathsf{P}), \tag{20}$$

$$x_{tp} = 0 \; or \; 1 \quad (\forall t \in \Im; p \in \mathsf{P}). \tag{21}$$

Equation (17) ensures each triplet appears exactly once in the timetable, while equations (18), (19), and (20) prevent class, teacher, and venue clashes, and equation (21) ensures the integrality of the solutions. Clearly this formulation is more compact than TT2, due to the reduction of dimensionality. While the formulation TT2 involves solution matrices of dimension C×T×V×P, TT3 reduces the dimension to τ×P. As will be seen later when the data sets are discussed, as C, T, and V grow linearly, so too does τ.

It should be noted that both formulations TT2 and TT3 are concerned only with minimising clashes. Other objective functions such as maximising preferences could be considered by including additional terms. These will be discussed in section 5.

3. Hopfield Neural Networks for Timetabling

In this section we describe how Hopfield neural networks can be used to solve the timetabling formulations presented in Section 2. The dynamics of the Hopfield neural network are first outlined, followed by a brief discussion of how they can be used to solve combinatorial optimisation problems. We then demonstrate how the two (class-teacher-venue) timetabling formulations TT2 and TT3 can be mapped onto the

Hopfield neural network by deriving the network weights. Finally, some modifications to the dynamics of the Hopfield neural network are proposed to enable what is essentially a gradient descent technique to become competitive with metaheuristics.

3.1 Hopfield Neural Networks

Hopfield neural networks (Hopfield, 1982; Hopfield, 1984) are a biologically inspired mathematical tool that can be used to solve difficult optimisation problems. Their advantage over more traditional optimisation techniques lies in their potential for rapid computational power when implemented in electronic hardware, and the inherent parallelism of the network.

There are two types of Hopfield networks, discrete and continuous models, which permit different values for neuron states. Biological modelling of the human brain is attempted by utilising a fully inter-connected system of N neurons. Neuron i has internal state u_i and output level v_i (which can be either binary valued in the discrete model or real valued bounded by 0 and 1 in the continuous model). The internal state u_i incorporates a bias current (or additional input) denoted by I_i, and the weighted sums of outputs from all other neurons. The weights, which determine the strength of the connections from neuron i to j, are given by W_{ij} . The relationship between the internal state of a neuron and its output level is detemined by an activation function $g(u_i)$. The nature of this activation function depends on whether the Hopfield network is discrete or continuous. Commonly,

$$v_i = g(u_i) = \frac{1}{2}(1 + \tanh(\frac{u_i}{\lambda})) \tag{22}$$

is used for the continuous model, where λ is a parameter used to control the slope (or gain) of the activation function. For discrete Hopfield networks, the activation function is usually a discrete threshold function:

$$v_i = g(u_i) = \begin{cases} 1 & if\, u_i > 0 \\ 0 & if\, u_i \le 0. \end{cases} \tag{23}$$

The neurons update themselves (either sequentially or in parallel) according to the following rule:

$$u_i(t+1) = u_i(t) + \Delta t(\sum_{j=1}^{N} W_{ij} v_j + I_i) \tag{24}$$

$$v_i(t+1) = g(u_i) \tag{25}$$

(where Δt is a constant time-step), and in doing so, the network of neurons will converge to a local minimum of the following energy function over time:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij} v_i v_j - \sum_{i=1}^{N} I_i v_i \tag{26}$$

provided the weights are symmetric $W_{ij} = W_{ji}$. If neurons are updated in parallel (or synchronously) then the possibility of convergence to a two-cycle exists. Both of the network states which comprise the two-cycle will be local minima of the energy function however.

The discrete model has an advantage over the continuous model in terms of the number of updates required to converge to a local minimum and the speed of executing each iteration. For this reason, and others related to hardware constraints (which have been discussed in Abramson et al., 1998), we have chosen to use a discrete Hopfield network for solving the timetabling problems formulated in the previous section. We have also chosen to update the neurons in a parallel operation rather than sequentially since it is our ultimate intention to solve large scale problems as rapidly as possible. Parallel implementation involves calculating all of the $\boldsymbol{u}$ updates then all of the $\boldsymbol{v}$ updates, as opposed to the sequential update which calculate the u and v update for each neuron one at a time.

Hopfield and Tank (1985) showed that if a 0-1 optimisation problem can be expressed in terms of an energy function of the form given by (26), a Hopfield network can be used to find locally optimal solutions of the energy function, which may translate to the local minimum solutions of an optimisation problem, provided the network parameters (weights and external bias) have been selected appropriately for the problem. Typically, the network energy function is made equivalent to the objective function of the optimisation problem which is to be minimised, while the constraints of the problem are included in the energy function as penalty terms. The network parameters can then be inferred by comparison with the standard energy function given by (26). The weights of the network, W_{ij}, are the coefficients of the quadratic terms $v_i v_j$, and the external bias currents, I_i for each neuron i, are the coefficients of the linear terms v_i in the chosen energy function.

The network can be initialised by setting the activity level v_i of each neuron to an unbiased state. Random and asynchronous updating of the network according to equations (24) and (25) will then allow a minimum energy state to be attained, since the energy level never increases during state transitions (Hopfield, 1982). The proof of this involves showing that:

$$\Delta E = -(\sum_{j=1}^{N} W_{ij} v_j + I_i)\Delta v_i \leq 0$$

since if $\sum_{j=1}^{N} W_{ij} v_j + I_i < 0$ then $\Delta u_i < 0$ and $\Delta v_i \leq 0$

and if $\sum_{j=1}^{N} W_{ij} v_j + I_i \geq 0$ then $\Delta u_i \geq 0$ and $\Delta v_i \geq 0$.

The derivation of the weights and external biases for the two timetabling formulations (TT2 and TT3) are provided in the following section.

3.2 Mapping Timetabling onto Hopfield Neural Networks

As discussed in the preceding section, in order to map an optimisation problem onto a Hopfield neural network, we first need to express the cost and constraints of the problem as penalty terms in a single function. Focusing first on TT2, this function can be constructed as:

$$E = \frac{\alpha}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}(\sum_{l=1}^{P} x_{ijkl} - R_{ij})^2 + \frac{\beta}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P}\sum_{\substack{j'=1 \\ j'\neq j}}^{T}\sum_{\substack{k'=1 \\ k'\neq k}}^{V} x_{ijkl}x_{ij'k'l}$$
$$+\frac{\chi}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P}\sum_{\substack{i'=1 \\ i'\neq i}}^{C}\sum_{\substack{k'=1 \\ k'\neq k}}^{V} x_{ijkl}x_{i'jk'l} + \frac{\gamma}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P}\sum_{\substack{i'=1 \\ i'\neq i}}^{C}\sum_{\substack{j'=1 \\ j'\neq j}}^{T} x_{ijkl}x_{i'j'kl} \qquad (27)$$

where α, β, χ, and γ are the penalty parameters chosen to equally balance the terms of the function. The first term of equation (27) is zero only if the requirements' constraint (7) is satisfied, and is positive otherwise. The remaining three terms each encourage satisfaction of the constraints (8), (9), and (10) by counting pairwise products and returning a non-zero value if two "1"s are found in the same dimension of the solution matrix **x**, representing a clash.

A Hopfield network can be designed to minimise this function, provided the network weights and external bias are determined appropriately. A neuron v_{ijkl} is made equivalent to the solution matrix element x_{ijkl}, and an eight-dimensional matrix of weights connecting all neurons v_{ijkl} to all other neurons $v_{i'j'k'l'}$ is constructed as well as a four-dimensional matrix of external biases. The energy function for this Hopfield network is

$$E = -\frac{1}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P}\sum_{i'=1}^{C}\sum_{j'=1}^{T}\sum_{k'=1}^{V}\sum_{l'=1}^{P} W_{ijkl,i'j'k'l'} v_{ijkl} v_{i'j'k'l'} - \sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P} I_{ijkl} v_{ijkl} \tag{28}$$

which is generalised from (26) to consider the four-dimensional nature of each neuron. After expanding and rearranging the function (27), comparison with the energy function (28) reveals that the two equations are equivalent provided

$$\begin{aligned} W_{ijkl,i'j'k'l'} &= -\alpha\delta_{ii'}\delta_{jj'}\delta_{kk'} - \beta\delta_{ii'}(1-\delta_{jj'})(1-\delta_{kk'})\delta_{ll'} \\ &\quad - \chi(1-\delta_{ii'})\delta_{jj'}(1-\delta_{kk'})\delta_{ll'} - \gamma(1-\delta_{ii'})(1-\delta_{jj'})\delta_{kk'}\delta_{ll'} \end{aligned} \tag{29}$$

$$I_{ijkl} = \alpha R_{ijk} \tag{30}$$

where $\delta_{yy'}$ is the Kronecker-Delta function, $\delta_{yy'}=1$ if $y = y'$ and 0 otherwise.

There is an additional constant term of $\frac{\alpha}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}R_{ijk}^2$ needed to demonstrate equivalence, however this term can be ignored since it does not affect the location of the local and global minima of the original function (27), and the Hopfield energy function (28) does not include a constant term.

The second class-teacher-venue formulation, TT3, can similarly be mapped onto a Hopfield neural network. Using this formulation, the dimensions of each neuron are smaller, as are the dimensions of the connecting weights and external biases. For TT3, a penalty function can be constructed which will be minimised when all constraints are satisfied. One such function is

$$E = \frac{\alpha}{2}\sum_{t=1}^{\tau}(\sum_{p=1}^{P}x_{tp} - 1)^2 + \frac{\beta}{2}\sum_{i=1}^{C}\sum_{p=1}^{P}\sum_{t=1}^{\tau}\sum_{\substack{t'=1 \\ t'\neq t}}^{\tau}x_{tp}\hat{C}_{it}x_{t'p}\hat{C}_{it'}$$
$$+\frac{\chi}{2}\sum_{j=1}^{T}\sum_{p=1}^{P}\sum_{t=1}^{\tau}\sum_{\substack{t'=1 \\ t'\neq t}}^{\tau}x_{tp}\hat{T}_{jt}x_{t'p}\hat{T}_{jt'} + \frac{\gamma}{2}\sum_{k=1}^{V}\sum_{p=1}^{P}\sum_{t=1}^{\tau}\sum_{\substack{t'=1 \\ t'\neq t}}^{\tau}x_{tp}\hat{V}_{kt}x_{t'p}\hat{V}_{kt'} \tag{31}$$

The first term of (31) will be zero if each triplet is represented exactly once in the timetable. The second term attempts to minimise class clashes by inspecting all triplets involving a given class in a given period, and summing any non-zero pairwise products found. This term will be zero if the timetable is clash-free for all classes in all periods. Similarly, the third and fourth terms attempt to minimise teacher and venue clashes.

To map this optimisation problem onto a Hopfield neural network, the solution matrix ***x*** is first renamed as a matrix of binary neurons ***v***, and the Hopfield energy function will be a function of v_{tp}, $W_{tp,t'p'}$, and I_{tp}. Rearranging this function in a similar manner to that described for TT2, and recognising the coefficients of the quadratic terms $v_{tp}v_{t'p'}$ to be the weights, and the coefficients of the linear terms v_{tp} to be the external biases, gives the following parameters:

$$W_{tp,t'p'} = -\alpha\delta_{tt'} - \beta(1-\delta_{tt'})\delta_{pp'}(\sum_{i=1}^{C}\hat{C}_{it}\hat{C}_{it'}) - \chi(1-\delta_{tt'})\delta_{pp'}(\sum_{j=1}^{T}\hat{T}_{jt}\hat{T}_{jt'})$$
$$-\gamma(1-\delta_{tt'})\delta_{pp'}(\sum_{k=1}^{V}\hat{V}_{kt}\hat{V}_{kt'}) \qquad (32)$$
$$I_{tp} = \alpha \qquad (33)$$

Thus, equations (29) and (30) define the network parameters for formulation TT2, and equations (32) and (33) for formulation TT3. Already it can be seen that the reduction in the dimensionality of the neurons, as well as the weight matrix and external bias matrix, makes TT3 considerably more efficient. The number of neurons required is also much reduced using TT3 for the larger timetabling problems solved in the following section, although this does depend on the number of requirements (triplets) in the timetable.

While such large weight matrices can normally slow down the computation of the Hopfield network, we have used several techniques to speed-up the computation of the network parameters before the updating algorithm begins. Firstly, since the weight matrices are very sparce, we reduce the number of times we need to enter the repetition loop required to calculate each weight matrix element by acknowledging that if all of the Kronecker-Delta calculation for a particular neuron pair are zero, then the weight matrix element will also be zero, and there is no need to perform the calculation. Further, we only store the non-zero weights, and use pointers to reference these weights. The result is a much faster initialisation of the weight matrix. Other opportunities to optimise the code have been created by using boolean logic when summing binary valued variables. Our aim here is to produce the fastest possible software implementation so that later comparisons with hardware implementation can be unaffected by differences not directly related to the implementation.

3.3 Modifications to the Hopfield Neural Network

The standard Hopfield neural network updates according to equations (24) and (25), which minimises both the Hopfield energy function and the equivalent optimisation penalty function. The nature of this minimisation is steepest gradient descent, which naturally causes some difficulty with convergence to local minima of the energy function. Further difficulties arise when we consider the nature of these local minima. Since the optimisation problem is represented using a series of penalty terms, it is difficult to equally balance these terms through optimal choice of the penalty parameters, in our case α, β, χ, and γ. If the α term is too large relative to the remaining three terms, our timetable is likely to contain all requirements or triplets, but there may be clashes between classes, teachers, venues, or all three. Likewise, if the α term is too small relative to the remaining three terms, the timetable is likely to be clash-free since requirements or triplets will be omitted to reduce clashes. These local minima states translate into an ineffective timetable. In order for the timetable to be satisfactory, a global minima must be obtained, corresponding to a cost value of zero in equations (27) or (31).

Clearly, even if the penalty parameters are chosen appropriately to equally balance the terms of the optimisation problem, we are still faced with the problem caused by the gradient descent dynamics of the update rule. In this section, we present some modifications to these dynamics to allow local minima to be escaped. These modifications are fully implementable in hardware, ensuring that through future research we can benefit from both improved quality and speed of solutions.

Our modifications are based on the observation that the Hopfield neural network works hardest and most productively (as measured by a rapidly decreasing energy function) during the first few iterations. An *iteration* is defined as one complete update of all N neurons according to equations (24) and (25). During the first few iterations, the energy function decreases rapidly, and improvements slow thereafter, until after around 50 iterations a local minima is often encountered and the network oscillates between neuron states that all have the same energy value. We therefore terminate the updating after a small number of iterations, introduce stochasticity into the neuron states to perturb the solution, renormalise the u values to 0 or 1 based on the current v values, and restart the descent. This procedure is repeated for a certain number of *descents*. The stochasticity at the end of each iteration is created by randomly choosing a neuron and assigning its state v a value of 0 or 1. A *threshold* is used to control the stochasticity, so that if the threshold is 0.5, then there is an equal chance of a neuron state becoming 0 or 1, but if the threshold was increased to say 0.9, then there is only a 10% chance that the neuron will be assigned a value of 1. In order to allow a particular descent to terminate prematurely if a local minima has already been encountered, we also introduce a measure called *stability* which counts the number of changed neuron states in an iteration $\sum_{k=1}^{N} \Delta v_k$ or $\Delta \mathbf{v}$. If the stability measure is below some pre-defined value, the Hopfield network can terminate the current descent, flip a neuron, and continue with another descent. This creates a dynamic number of iterations, and enables the total number of iterations to be substantially reduced where possible.

Thus, the modifications to the network dynamics described above combine to contribute to an efficient wandering of the search space in short bursts of gradient descent, using controlled stability criteria and random perturbations to escape local minima. The parameters to be determined are the penalty parameters α, β, χ, and γ, as well as:

1. number of iterations per descent
2. the number of descents
3. the stability criteria for premature termination of an iteration
4. the number of neurons allowed to flip at the end of an iteration
5. the threshold to determine the probability of assigning a neuron a value of 1 when flipped
6. the total number of runs performed from random initialisation of $\boldsymbol{u}$ and $\boldsymbol{v}$.

The discrete time-step in equation (24) is fixed at $\Delta t = 1$ since multiplication needs to be avoided for fast hardware implementation. All other multiplications can be handled using conditional adds since the neuron states are binary valued (Abramson et al., 1998).

The complete modified Hopfield network algorithm can thus be summarised by the following pseudo-code:

```
initialise $\boldsymbol{u}$ (randomly around zero)
calculate $\boldsymbol{v}$ using equation (23)

for i = 1 to *descents*
    for j = 1 to *iterations*
        for k = 1 to $N$
            update $u_k$ using equation (24)
            update $v_k$ using equation (23)
        next k
        calculate *stability* = $\sum_{k=1}^{N} \Delta v_k$
        if *stability criteria* is satisfied, then exit j loop
    next j
    randomly choose a neuron, and *flip* according to *threshold*
    renormalise $\boldsymbol{u}$ ($u_k \rightarrow 1$ if $u_k > 0$; $u_k \rightarrow 0$ if $u_k < 0$)
next i
```

4. Comparative Study

In this section we describe the data sets used, and present the results of the Hopfield neural network using the two formulations TT2 and TT3. We also compare the results to those obtained when none of our modifications to the Hopfield network dynamics are used to demonstrate the effect they have on solution cost and time, and compare the results to simulated annealing, tabu search and greedy search to provide a comparison with other metaheuristics.

4.1 Data Sets

The data sets used in this paper are the same ones benchmarked by Abramson and Dang (1993), and are publically available from the OR-Library at Imperial College London (http://www.ms.ic.ac.uk). Problems of various sizes have been randomly constructed so that the timetable is totally constrained, ie. each period contains every class, teacher, and venue. Since the solution is generated we know there is a clash-free solution to each problem, but the lack of flexibility in the final solution makes it extremely difficult to find. Table 1 shows the characteristics of the five data sets used in this study.

<<INSERT TABLE 1 HERE>>

While these problems are relatively small in terms of class, teacher, and venues sizes, they are dense as shown by the number of triplets, and are thus extremely difficult to solve (the "hd" in the problem name stands for "hard"). Considering that the purpose of this paper is to show the solving ability of our modified neural network on hard problems, these benchmarked data sets are most appropriate. They are also large enough to demonstate differences between the efficiencies of the formulations and techniques. It should be noted that while real school timetabling problems are larger than the problems solved here, their complexity is typically significantly reduced. For example, Nepal et al. (1998) describe a real school timetabling problem involving 31 teachers, 35 classes, 18 venues, 50 periods, however the density of the problem was such that only 9 of the 18 venues needed to be used in order to find a clash-free solution. The timetabling problems we solve in this section are completely constrained, and a clash-free solution is difficult to find. Thus we are focused on testing the neural network's capabilities as an optimisation technique more than solving a practical problem (larger but not as dense).

4.2 Neural Network Formulations

It is clear from the description of the data sets in Table 1 that as the problem size increases, the number of neurons required for formulation TT2 varies from 1920 to 15360, while for formulation TT3 the number of neurons is initially larger at 3600 but only increases to 7200. So the advantage of using formulation TT3 for these data sets is already clear. The number of weights required for TT2 solving problem hdtt8 is 235,929,600 whereas only 51,840,000 weights are required for TT3.

Figure 1 provides a comparison of the two formulations across the 5 data sets. These results are generated based on an average of 20 runs, using the following parameter combinations: $\alpha = 3$, $\beta = \chi = \gamma = 1$, iterations = 10, descents = 500, flips = 1 or 2, threshold = 0.85, and stability is defined as $\Delta \mathbf{v} < 20$ during an iteration (less than 20 changes across all neurons). Figure 1a demonstrates the relationship between computational effort and the formulation, which is affected by the number of neurons. Figure 1b shows the average cost of the solutions over the 20 runs, for each formulation and different problem sizes. This cost is the value of the functions (27) and (31) for formulations TT2 and TT3 respectively. A cost of zero corresponds to a clash-free timetable. Figure 1c demonstrates the percentage of the 20 runs that resulted in a zero-cost solution.

<<INSERT FIGURE 1a, 1b, 1c HERE>>

Figure 1a shows that formulation TT2 requires considerably more CPU seconds to produce 500 descents for larger problems than does TT3, which is merely a reflection of the number of neurons involved in each formulation. Thus formulation TT3 scales better than TT2. Figure 1b demonstrates that formulation TT3 also produces a better average cost within 500 descents for larger problems using the chosen parameter combinations, with a slightly higher rate of finding perfect solutions as shown in Figure 1c. It should be noted that the chosen parameter combination, which is used for both TT2 and TT3, was developed to demonstrate good performance on larger problems. All neural network results reported in this paper are produced using a common set of parameters. By varying the parameters we can find solutions that improve the results for the smaller problems, but our focus here is on producing good solutions for larger timetabling problems.

4.3 Effect of Modifications to Hopfield Network Dynamics

In order to determine the genuine effect of the modifications we have made to the Hopfield network dynamics, we have conducted a series of experiments where we compare the performance of our modified Hopfield network to one where no flips randomly perturb the solution after each descent. The random perturbations have proven to be invaluable in helping the solution escape local minima and continue to explore the search space. The approach used to escape local minima is also more easily implementable in hardware compared to our previous approaches (Smith et al., 1996; Smith et al., 1998) which were designed for continuous Hopfield networks and are considerably slower to execute than the current discrete Hopfield network.

Figure 2 demonstrates how the procedure of randomly flipping 1 or 2 neurons at the end of every descent can help to produce a better cost over time. This figure is based on a single run using formulation TT3 to solve problem hdtt8. Similar graphs are obtained using different runs, different problems sizes and formulation TT2. The graph with no flips does not undertake a pure descent since we are still renormalising the ***u*** values at the end of every descent, but it is clear that the random flipping procedure is beneficial to obtaining a zero cost solution.

<<INSERT FIGURE 2 HERE>>

4.4 Comparison with Metaheuristics

In order to gauge the effectiveness of the Hopfield neural network results, we now present a comparative performance against simulated annealing, tabu search, and a greedy search algorithm, across all five test problems. The results are compared in terms of both average cost and average CPU seconds for a set of 20 random initialisations. In the following sections, we discuss the metaheuristic approaches used for comparison, as well as the particular parameter combinations used for the reported results.

4.4.1 *Simulated Annealing Approach*

Two different simulated annealing implementations of the timetabling problem are used for comparison. The first is the implementation of Abramson and Dang (1993) using the triplet mapping approach first discussed in Abramson (1991). Triplets are removed and added to the timetable, and the change in cost is used to determine the next transition. Furthermore, classes, teachers, and venues can all be removed or added to triplets via swap and move operators. The work extended upon the earlier work of Abramson (1991) by providing a new computer architecture designed to speed up the execution of the simulated annealing algorithm. Abramson and Dang (1993) presented the results obtained on the test problems shown in Table 1 for a new hardware accelerator, and compared the results with their simulated annealing algorithm implemented in software on a variety of machines (including a CRAY Y/MP, Sun Sparc Station 1+, and PC486). This comparison demonstrated that orders of magnitude increase in performance could be obtained with their hardware accelerator. We use their results, which we refer to as SA1, to compare both the quality and speed of the neural network solutions later in this section.

The second simulated annealing approach that we use for comparison is the metaheuristic based solver of Randall, Abramson & Wild (1999). This method uses an efficient linked list based representation of a combinatorial optimisation problem, rather than using binary matrices to encode solutions to problems. For the timetabling problem, the solution is encoded using the integer matrix **X** where X_{ij} stores the triplet number corresponding to the jth list element in period i. As an example, if period 7 contains triplets numbered 25, 49, and 121, then $X_{71}=25$, $X_{72}=49$, and $X_{73}=121$. The lengths of these lists are dynamic, with many operators available to allow transitions: swap, move, reposition, inversion, add, drop, change. The probability of any of these operators being chosen is adaptive, so that if a transition operator is producing high quality solutions, its transition probability is raised. The simulated annealing search engine implements Connolly's Q8-7 cooling schedule as it has been shown to be quite successful (Abramson & Randall, 1999; Connolly, 1990; Connolly, 1992). The cooling schedule is based on reheating the temperature a number of times throughout the search. Both the number of times that this reheating occurs as well as the interval between reheats can be altered. The simulated annealing engine also has an option that allows it to perform as many reheats as possible in order to try to reach a given solution quality. The results presented later in this section utilise (10000*τ) iterations between reheats, with the reheating procedure continuing until either a cost of zero or 2750 seconds is reached. We refer the interested reader to Randall et al. (1999) and Abramson & Randall (1999) for further details of the metaheuristic solver.

4.4.2 *Tabu Search and Greedy Search Approaches*

The tabu search and greedy search results used for comparison in this study have also been produced using the metaheuristic solver of Randall et al. (1999). The tabu search engine allows the tabu list size to be set. This parameter governs the number of tabu search iterations for which a particular transition stays tabu. The tabu list is implemented as a matrix as this approach has been adopted by a number of other researchers (Osman, 1993; Osman, 1995; Taillard, 1991) and has been shown to be quite effective. The tabu list in this approach records whether a particular element has been placed on a certain sub-list. For efficiently searching the neighbours at each iteration, the system implements a simple candidate list strategy (Glover & Laguna, 1997) based on the probabilistic selection of neighbours. For instance, if the probability is set to 1, then all neighbours are tested, however, if the probability is set to 0.5, then each neighbour has a 50% chance of being evaluated. For the results presented in the following section, a probability of 0.5 was used to generate the tabu search results, while a probability of 1.0 was used to generate the greedy search results. These were found to produce the best results from the set of {0.1, 0.5, 1.0}.

4.4.3 *Results*

Table 2 presents the comparative results for all problem sizes. NN-TT2 and NN-TT3 are our Hopfield neural networks solving formulations TT2 and TT3 respectively. SA1 is the simulated annealing approach of Abramson and Dang (1993) implemented using their hardware accelerator. SA2, TS, and GS are the results from using the metaheuristic solver of Randall et al. (1999) using the simulated annealing, tabu search, and greedy search engines respectively. The results reported in Table 2 are based on a set of 20 runs, and record the cost of the best solution found (*best cost*), the average cost of the 20 runs (*average cost*), and the average CPU time in seconds over the 20 runs (*av CPU (secs)*). All techniques, with the exception of SA1, were run on a 333MHz Pentium II with 128MB RAM. The best average cost for each problem is highlighted in bold.

<<INSERT TABLE 2 HERE>>

The results demonstrate clearly that NN-TT3 is able to compete effectively with SA2, the best of the metaheuristics, and far outperforms the other techniques in terms of both solution quality and speed. The results from NN-TT3 on the largest problem hdtt8 are only beaten in terms of required CPU seconds by greedy search, but the solution quality produced by greedy search is poor. This provides further evidence of the difficulty of these benchmark problems, and the need for hill-climbing techniques. On the larger problems (hdtt6 - hdtt8), NN-TT3 produces better average costs than

SA2, with the average CPU time for hdtt8 faster than SA2, indicating the superior scaling properties of NN-TT3 over SA2. Naturally when we implement the neural networks in hardware, at least one order of magnitude speed-up will be obtained (Abramson et al., 1998), reducing the computational requirements considerably. Similarly, both neural network formulations produce better results for the larger problems than SA1. As discussed earlier in this section, better neural network performance can be obtained for the smaller problem through different choice of parameters. For example, reducing the threshold to zero enables improved solution to hdtt4 with an average cost of 0.0 in 19.1 seconds for TT2 and an average cost of 0.0 in 42.6 seconds for TT3. The neural network results presented in Table 2 are all based on the same set of parameters however, and we have chosen the parameters to demonstrate the improved performance for the larger problems.

The results of Randall et al. (1999) for the tabu search algorithm shown in Table 2 are somewhat surprising, and we might conclude that either the linked list based representation used for these results does not suit tabu search, or the incorporation of further enhancements (Glover & Laguna, 1997) to the algorithm are required.

5. Extensions to Other Timetabling Problems

The focus of this paper has been on demonstrating the optimisation capabilities of our modified Hopfield neural network by solving a set of difficult benchmarked school timetabling problems. The school timetabling problem was chosen as a classical operations research problem, versatile as a generalisation of the quadratic assignment problem, and for the existence of challenging data sets that are both publicly available and benchmarked. Our results show that the modified neural network can compete effectively and efficiently with metaheuristics such as simulated annealing and tabu search, benchmarked and previously published on the same data sets.

While our focus has not been on providing a solution technique for solving large and practical timetabling problems, it is natural to question the applicability of our modified Hopfield neural network to other timetabling problems: larger problems, as well as variations on the timetabling problem considered here.

The availability of other data sets for the problem described in this paper, class-teacher timetabling with room allocations, is limited. According to a recent survey by Carter and Laporte (1998), there are only a limited number of practical problems that have been published fitting this description (Abramson, 1991; Costa, 1994; Nepal et al., 1998; Chan, Lau & Sheung, 1998). The sizes of these problems vary from 15 to 101 classes, 15 to 65 teachers, and 12 to 24 rooms. Other studies solve practical problems but do not present the details of the problem size (Colorni, Dorigo & Maniezzo, 1992; Cooper & Kingston, 1993). Compared to the artificial data sets we have solved, which force each period to utilise each class, teacher, and room, typical practical school timetabling problems are less dense. This corresponds to a reduced number of triplets per period. The benefits of formulation TT3 when solving practical timetabling problems is further illustrated when we consider that the number of neurons required for TT3 is $\tau \times P$, whereas for formulation TT2 it is $C \times T \times V \times P$. Our experiments have revealed that formulation TT3 can easily solve problems with over 1000 requirements, exceeding the theoretical maximum number of requirements for many of the practical school timetabling problems mentioned above.

We focus now on the issue of the applicability of this approach to solving related timetabling problems, such as those with side constraints including teacher preferences, and room capacity constraints. Any additional constraints can typically be handled by incorporating further penalty terms in the objective function. For example, suppose teacher j is unavailable or would prefer not to teach during a set of periods `Lj`. Then $\Pi_{jl}=1$ for all $j \in \mathrm{T}$ and $l \in$ `Lj`, and the additional term added to the objective function of formulation TT2 is $\frac{\varphi}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P} x_{ijkl}\Pi_{jl}$. For this constraint, φ is the penalty parameter set to determine the relative importance of this term. A similar term can be derived for formulation TT3, and this term can also be modified to enforce class and room unavailabilities or preferences.

Capacity constraints can also be handled using the penalty parameter approach as follows: suppose venue k has a capacity of Φ_k and class i has a size of S_i. Then in order to satisfy capacity constraints on each room, the following constraint must hold: $S_i x_{ijkl} \leq \Phi_k$ for all i and k. This constraint can be expressed as a penalty term in the Hopfield energy function as: $\frac{\mu}{2}\sum_{i=1}^{C}\sum_{j=1}^{T}\sum_{k=1}^{V}\sum_{l=1}^{P}\sum_{j'=1}^{T}\sum_{l'=1}^{P}(\frac{S_i}{\Phi_k})x_{ijkl}x_{ij'kl'}$.

These additional constraints can be considered as soft constraints, compared to the harder constraints intended to eliminate clashes. Consequently, the values of the penalty parameters φ and μ would be smaller than the parameters for the harder constraints α, β, χ and γ. Other constraints can also be handled in this manner.

6. Conclusions

In this paper we have presented a modified discrete Hopfield neural network as a competitive approach for solving timetabling problems compared to existing metaheuristic approaches. We have proposed two formulations for mapping the school timetabling problem onto a Hopfield network. The first one is a basic extension of the work of Hopfield and Tank (1985), applicable to all combinatorial optimisation problems. The second and more efficient formulation borrows the idea of mapping triplets to periods (Abramson, 1991) and then maps this problem onto a Hopfield neural network. We have also presented some modifications to the standard dynamics of the discrete Hopfield network which allow better and faster results to be obtained. These modifications enable the neural network to work in short bursts of gradient descent, followed by random but controlled perturbations to escape local minima. Stability measures are used to enable a dynamic number of iterations at each descent, preventing the computation time running longer than is necessary. A further advantage of these modifications is that they enable the hardware implementability of the neural network to be retained.

Several experiments have been conducted to evaluate the performance of the new Hopfield network approach based on benchmarked data sets (Abramson & Dang, 1993). Firstly, we have examined the effect of the two different formulations and mappings of the timetabling problem. The triplet mapping formulation, TT3, has been shown to be more efficient in terms of the number of neurons and weights required to encode the problem. While similar solution quality is obtained for both neural network formulations, the improvement in computational speed for TT3 makes it the preferred neural network approach. The second set of experiments aimed to determine the effect of the modifications we have made to the dynamics of the neural network, namely the effect of the random flips, controlled through the parameter *threshold*. An example was used to demonstrate the improved performance generated by this modification. Finally, we compared the neural network results to other metaheuristic approaches to determine if neural networks are a competitive solution approach for timetabling problems. Two simulated annealing algorithms were used, together with tabu search and greedy search. The neural network approach based on the more efficient formulation proved to be comparable on average to the best metaheuristic, simulated annealing, outperforming it on larger problems.

Thus the neural network approach described in this paper has been shown to be capable of producing good quality solutions to extremely difficult timetabling problems. The more efficient formulation demonstrated good scaling properties as well. When we consider the speed advantages that can be obtained when implementing the neural network in hardware (Abramson et al., 1998), it is foreseeable that neural networks will soon become a rapid solution technique for large and complex timetabling and other combinatorial optimisation problems.

The outcomes of this research have indicated several future research directions. Further improvements to the solution quality obtained by our modified Hopfield neural network could be found by annealing some of the parameters including *threshold*, so that initially many flips are likely, with less flips occuring as the algorithm proceeds. This is clearly similar to the concepts employed by simulated annealing, and may improve the wandering ability of the neural network search.

Further research focusing on efficient formulations for timetabling problems with real-world constraints not considered here may also need to be conducted. We have seen in this paper that the original formulation of a timetabling problem can have a large effect on the performance results. The final direction for future research is the hardware implementation of the neural network, which we are currently investigating, so that the speed advantages can be realised.

Acknowledgements

The authors are grateful to the Australian Research Council and the School of Business Systems, Monash University for their financial support of this project. They also thank Dr. Marcus Randall for providing his metaheuristic results for comparison.

References

Abramson, D. (1991). Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37(1), 98-113.

Abramson, D., & Dang, H. (1993). School timetables: a case study in simulated annealing. In: V. Vidal, *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematics Systems (Chapter 5, pp. 103-124). Berlin: Springer-Verlag.

Abramson, D., & Randall, M. (1999). A simulated annealing code for general integer linear programs. *Annals of Operations Research*, 86, 3-21.

Abramson, D., Smith, K.A., Logethetis, P., & Duke, D. (1998). FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problems. *Proceedings EuroMicro Workshop on Computational Intelligence*, 688-693.

Bianco, L., Bielli, M., Mingozzi, A., & Ricciardelli, S. (1992). A heuristic procedure for the crew rostering problem. *European Journal of Operational Research*, 58(2), 272-283.

Burke, E., & Carter, M. (1998). *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, volume 1408. Berlin: Springer-Verlag.

Burke, E., & Ross, P. (1996). *Practice and Theory of Automated Timetabling*, *Proceedings First International Conference*, *Lecture Notes in Computer Science*, volume 1153. Berlin: Springer-Verlag.

Burke, E., Elliman, D., & Weare, R. (1995). Specialised recombinative operators for timetabling problems. In: T. C. Fogarty, *Evolutionary Computing*, *AISB Workshop* (pp. 75-85). Berlin: Springer-Verlag.

Carter, M., & Laporte, G. (1998). Recent developments in practical course timetabling. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 3-19). Berlin: Springer-Verlag.

Chan, H.W., Lau, C.K., & Sheung, J. (1998). Practical school timetabling: a hybrid approach using solution synthesis and iterative repair. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408*. Berlin: Springer-Verlag.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). Genetic algorithms – a new approach to the timetable problem. In A. Akgul, *Lecture Notes in Computer Science*, *vol. F82*. (pp. 235-239). Berlin: Springer-Verlag.

Connolly, D. (1990). An improved annealing for the QAP. *European Journal of Operational Research*, 46, 93-100.

Connolly, D. (1992). General purpose simulated annealing. *Journal of the Operational Research Society*, 43, 495-505.

Cooper, T.B. & Kingston, J.H. (1993). The solution of real instances of the timetabling problem. *The Computer Journal*, 36(7), 645-653.

Costa, D. (1994). A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76, 98-110.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, 19, 151-162.

Dowsland, K.A. (1998). Off-the-peg or Made-to-measure? Timetabling and scheduling with SA and TS. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 37-52). Berlin: Springer-Verlag.

Elmohamed, M.A.S., Coddington, P., & Fox, G. (1998). A comparison of annealing techniques for academic course scheduling. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 92-112). Berlin: Springer-Verlag.

Erben, W., & Keppler, J. (1996). A genetic algorithm solving a weekly course-timetabling problem. In: E. Burke & P. Ross, *Practice and Theory of Automated Timetabling*, *Proceedings First International Conference*, *Lecture Notes in Computer Science*, *vol. 1153* (pp. 198-211). Berlin: Springer-Verlag.

Even, S., Itai, A. & Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing*, 5(4), 691-703.

Ferland, J.A., & Lavoie, A. (1992). Exchange procedures for timetabling problems. *Discrete Applied Mathematics*, 35(3), 237-253.

Gianoglio, P. (1990). Application of neural networks to timetable construction. *Neuro-Nimes'90*, *Proceedings Third International Workshop Neural Networks and Their Applications*, 53-67.

Gislen, L., Peterson, C., & Soderberg, B. (1989). "Teachers and classes" with neural networks. *International Journal of Neural Systems*, 1(2), 167-176.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic Publishers.

Guéret, C., Jussien, N., Boizumault, P., & Prins, C. (1996). Building university timetables using constraint logic programming. In: E. Burke & P. Ross, *Practice and Theory of Automated Timetabling*, *Proceedings First International Conference*, *Lecture Notes in Computer Science*, *vol. 1153* (pp. 130-145). Berlin: Springer-Verlag.

Hertz, A. (1992). Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54, 39-47.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Sciences*, 79, 2554-2558.

Hopfield, J.J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings National Academy of Sciences*, 81, 3088-3092.

Hopfield, J.J., & Tank, D.W. (1985). 'Neural' computation of decisions in optimization problems. *Biological Cybernetics*, 52, 141-152.

Hsu, C.C., Lin, D.M., & Hu, C.C. (1994). A timetabling model based on constraint checking technique for rostering problems. *Proceedings 3rd Pacific Rim International Conference on Artificial Intelligence*, 1, 84-89.

Kamgar-Parsi, B., & Kamgar-Parsi, B. (1987). An efficient model of neural networks for optimisation. *Proceedings IEEE International Conference on Neural Networks*, 3, 785-790.

Kang, L., & White, G.M. (1992). A logic approach to the resolution of constraints in timetabling. *European Journal of Operational Research*, 61, 306-317.

Kovacic, M. (1993). Timetable construction with Markovian neural network. *European Journal of Operational Research*, 69(1), 92-96.

Krarup, J. (1982). Chromatic optimisation: Limitations, objectives, uses, references. *European Journal of Operational Research*, 11, 1-19.

Lim, J.H., & Loe, K.F. (1991). Timetable scheduling using neural networks with parallel implementation on transputers. *Proceedings IEEE International Joint Conference on Neural Networks*, 1, 122-127.

Mausser, H.E., & Magazine, M.J. (1996). Comparison of neural and heuristic methods for a timetabling problem. *European Journal of Operational Research*, 93(2), 271-287.

Mausser, H.E., Magazine, M.J., & Moore, J.B. (1996). Application of an annealed neural network to a timetabling problem. *INFORMS Journal on Computing*, 8(2), 103-117.

Mehta, N.K. (1981). The application of a graph coloring method to an examination scheduling problem. *Interfaces*, 11, 57-64.

Nepal, T., Melville, S.W., & Ally, M.I. (1998). A brute force and heuristics approach to tertiary timetabling. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 254-265). Berlin: Springer-Verlag.

Osman, I.H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41, 421-451.

Osman, I.H. (1995). Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations Research Spektrum*, 17, 211-225.

Osman, I.H., & Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63, 513-623.

Papageorgiou, G., Likas, A., & Stafylopatis, A. (1998). Improved exploration in Hopfield network state-space through parameter perturbation driven by simulated annealing. *European Journal of Operational Research*, 108(2), 283-292.

Pellerin, D., & Herault, J. (1994). Scheduling with neural networks: application to timetable construction. *Neurocomputing*, 6(4), 419-442.

Randall, M., Abramson, D., & Wild, C. (1999). A general meta-heuristic based solver for combinatorial optimisation problems. *Technical Report, TR99-01*. School of Information Technology, Bold University, Gold Coast, Queensland 4229, Australia.

Ross, P., & Corne, D. (1995). Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems. In: T. C. Fogarty, *Evolutionary Computing*, *AISB Workshop* (pp. 94-102). Berlin: Springer-Verlag.

Ross, P., Hart, E., & Corne, D. (1998). Some observations about GA-based exam timetabling. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 115-129). Berlin: Springer-Verlag.

Schreuder, J.A.M.. & van der Velde, J.A. (1984). Timetables in Dutch High Schools. In: J.P. Brans, *Operational Research '84* (pp. 601-612). Amsterdam: North-Holland.

Smith, K.A. (1999). Neural networks for combinatorial optimisation: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1), 15-34.

Smith, K.A., Palaniswami, M., & Krishnamoorthy, M. (1996). Traditional Heuristic versus Hopfield Neural Network Approaches to a Car Sequencing Problem. *European Journal of Operational Research*, 93(2), 300-316.

Smith, K.A., Palaniswami, M., & Krishnamoorthy, M. (1998). Neural Techniques for Combinatorial Optimisation with Applications. *IEEE Transactions on Neural Networks*, 9(6), 1301-1318.

Taillard, E. (1991). Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17, 443-455.

Tsuchiya, K., & Takefuji, Y. (1997). A neural network parallel algorithm for meeting scheduling problems. *Applied Intelligence*, 7(3), 205-213.

Van Den Bout, D.E., & Miller, T.K. (1989). Improving the Performance of the Hopfield-Tank Neural Network through Normalization and Annealing. *Biological Cybernetics*, 62, 129-139.

White, G.M., & Haddad, M. (1983). An heuristic method for optimizing examination schedules which have day and night courses. *Computers and Education*, 7(4), 235-238.

White, G.M., & Zhang, J. (1998). Generating complete university timetables by combining tabu search with constraint logic. In: E. Burke & M. Carter, *Practice and Theory of Automated Timetabling*, *Proceedings Second International Conference*, *Lecture Notes in Computer Science*, *vol. 1408* (pp. 187-198). Berlin: Springer-Verlag.

Wilson, G.V., & Pawley, G.S. (1988). On the stability of the TSP algorithm of Hopfield and Tank. *Biological Cybernetics*, 58, 63-70.

Wright, M. (1996). School timetabling using heuristic search. *Journal of the Operational Research Society*, 47(3), 347-57.

Figure 1: Comparison of neural network performance on two formulations TT2 and TT3 and different problem sizes, for a) average CPU seconds for 500 descents, b) average cost over 20 runs, and c) percentage of runs resulting in a perfect timetable solution.

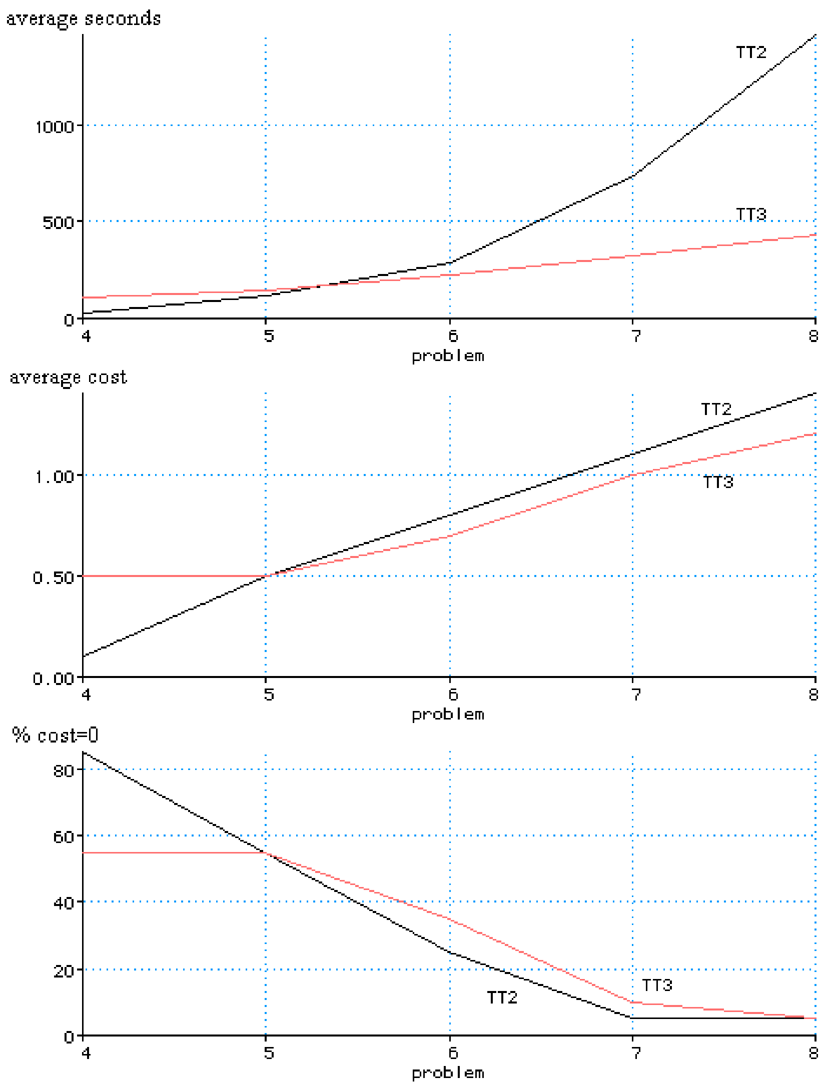

Figure 2: Effect of flips on Hopfield network performance

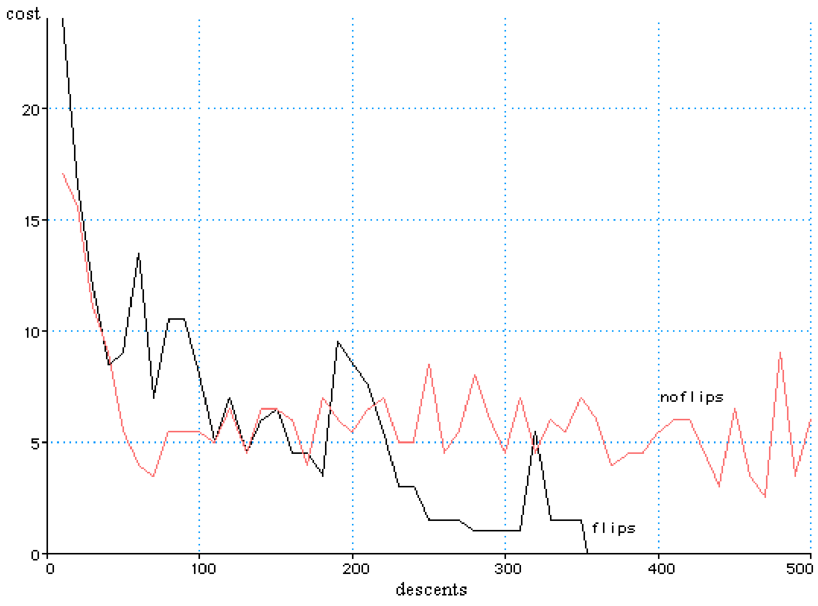

Table 1: Description of data sets

Problem Name	**Number of Classes (C)**	**Number of Teachers (T)**	**Number of Venues (V)**	**Number of Periods (P)**	**Number of Triplets (τ)**
hdtt4	4	4	4	30	120
hdtt5	5	5	5	30	150
hdtt6	6	6	6	30	180
hdtt7	7	7	7	30	210
hdtt8	8	8	8	30	240

Table 2: Results for different techniques on timetabling data sets

method		*hdtt4*	*hdtt5*	*hdtt6*	*hdtt7*	*hdtt8*
NN-TT2	best cost	0	0	0	0	0
	average cost	0.1	0.5	0.8	1.1	1.4
	av CPU (secs.)	28.0	115.9	290.9	735.7	1465.1
NN-TT3	best cost	0	0	0	0	0
	average cost	0.5	0.5	**0.7**	**1.0**	**1.2**
	av CPU (secs.)	109.2	146.3	226.9	323.7	431.3
SA1[†]	best cost	*	0	0	2	2
	average cost	*	0.67	2.5	2.5	3.83
	av CPU (secs.)	*	72.45	80.22	6635.00	7859.00
SA2	best cost	0	0	0	0	0
	average cost	**0**	**0.3**	0.8	1.2	1.9
	av CPU (secs.)	14.57	41.20	123.02	256.59	471.20
TS	best cost	0	0	3	4	13
	average cost	0.2	2.2	5.6	10.9	17.2
	av CPU (secs.)	630.14	686.90	1143.47	1276.17	1221.53
GS	best cost	5	11	19	26	29
	average cost	8.5	16.2	22.2	30.9	35.4
	av CPU (secs.)	15.83	39.47	77.95	140.30	397.02

[†] SA1 is the approach of Abramson and Dang (1993). The results are averaged over 6 runs, rather than 20 runs for all other results. The CPU times are also not based on the same machine as the other techniques. Results for hdtt4 are not available.