PARALLEL EVALUATION OF HOPFIELD NEURAL NETWORKS

Antoine Eiche, Daniel Chillet, Sebastien Pillement and Olivier Sentieys
University of Rennes I, IRISA, INRIA, 6 Rue de Kerampont, Lannion, France

Keywords: Hopfield neural networks, Parallelization, Stability, Optimization problems.

Abstract: Among the large number of possible optimization algorithms, Hopfield Neural Networks (HNN) propose interesting characteristics for an in-line use. Indeed, this particular optimization algorithm can produce solutions in brief delay. These solutions are produced by the HNN convergence which was originally defined for a sequential evaluation of neurons. While this sequential evaluation leads to long convergence time, we assume that this convergence can be accelerated through the parallel evaluation of neurons. However, the original constraints do not any longer ensure the convergence of the HNN evaluated in parallel. This article aims to show how the neurons can be evaluated in parallel in order to accelerate a hardware or multiprocessor implementation and to ensure the convergence. The parallelization method is illustrated on a simple task scheduling problem where we obtain an important acceleration related to the number of tasks. For instance, with a number of tasks equals to 20 the speedup factor is about 25.

1 INTRODUCTION

Hopfield Neural Network (HNN) is a kind of recurrent neural network that has been defined for associative memory or to solve optimization problems (Hopfield and Tank, 1985). They have been used to solve a lot of optimization problems such as travelling saleman (Smith, 1999), N-queens (Mańdziuk, 2002) or task scheduling (Wang et al., 2008). Our context is to solve the task scheduling problem at runtime, and, therefore, the execution time of the algorithm is crucial. HNN allows for an efficient hardware implementation because the control logic to evaluate a HNN is really simple. A HNN provides a solution when the network has converged, and this convergence has been demonstrated under some constraints on the input and connection weights, and with a sequential evaluation of neurons. This article focuses on a parallel evaluation model of HNNs in order to further improve the execution time.

A lot of authors proposed different approaches to improve the quality of generated solutions, but the literature about improvements of execution time is not very large. Moreover, in this kind of works, the HNN convergence constraints are not often respected, then a controller is needed to stop the network when the solution seems to be satisfactory. In this article, we focus on reducing the HNN convergence time by evaluating several neurons simultaneously. Because this evaluation method modifies the initial HNN convergence constraints, we recall the convergence proof in order to exhibit the required properties ensuring the convergence. Then, we show how to build a HNN which can be evaluated in parallel while ensuring convergence.

We present a parallelization method of HNN evaluation which aims to

- decrease the evaluation time of the HNN, and
- ensure the convergence.

Section 2 is a brief presentation of the HNN model. Section 3 presents some related works about the improvement of HNN evaluation. Since several neurons are evaluated simultaneously, we recall the convergence proof at the beginning of Section 4. Then, we show how the evaluation must be achieved to ensure the convergence. In Section 5, we present improvements brought by our parallelization method on a simplified scheduling problem. Finally, Section 6 concludes and gives some perspectives.

2 HOPFIELD NEURAL NETWORK

In this work, HNNs are used to solve optimization problems. This kind of neural networks is modeled as a complete directed graph. Figure 1 presents a HNN with three neurons. Each neuron has a threshold (in-

Eiche A., Chillet D., Pillement S. and Sentieys O..
PARALLEL EVALUATION OF HOPFIELD NEURAL NETWORKS.
DOI: 10.5220/0003682902480253
In *Proceedings of the International Conference on Neural Computation Theory and Applications* (NCTA-2011), pages 248-253
ISBN: 978-989-8425-84-3

put) value (I_i for neuron X_i) and receives connections $W_{i,j}$ from all other neurons (e.g. $W_{1,2}$ and $W_{1,3}$ for X_1). In order to simplify notation and implementation, when the weight of a connection between two neurons is equal to zero, these neurons are not connected by an edge.

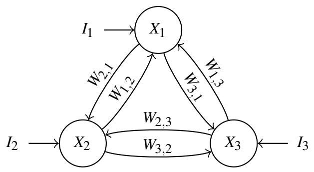

Figure 1: Example of Hopfield neural network with three neurons.

A neuron has a binary state, which is either *active* or *inactive*, respectively represented by values 1 and 0. A HNN can be evaluated in several ways, the most common is called sequential mode: neurons are randomly evaluated one by one. In this case, the evaluation of a neuron is given by

$$X_i = H(\sum_{j=1}^{n} X_j \times W_{ij} + I_i), \quad (1)$$

$$\text{where } H(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}, \quad (2)$$

and n the number of neurons in the network, X_i the state of the neuron i, W_{ij} the connection weight from neuron j to neuron i, I_i the threshold of neuron i. We then define respectively the state and threshold vector of the network $X = [X_i]$ and $I = [I_i]$ of size n and the $n \times n$ connection matrix $\mathbf{W} = [W_{ij}]$.

The main idea behind using HNN for solving optimization problems is to map the optimization problem to a particular function called the *energy function*. From this energy function, the parameters (input and connection weights) of the network can be derived. The dynamics of the network is launched until it reaches a stable state. When the network becomes stable, the state of the neurons represents one possible solution.

3 RELATED WORKS

To parallelize the evaluation of a HNN, two main techniques can be used. The first one is based on the parallelization of the internal neuron computation, while the second one is to update several neurons at the same time.

In (Del Balio et al., 1992), the authors proposed an optimized evaluation of HNN. To make the evaluation of the network faster, they parallelize the state update of a neuron. Eq. (1) exhibits some multiplications and an accumulation during the update process of a neuron. The authors parallelize all multiplications and use a special communication infrastructure to accelerate the accumulation. When the number of neurons is sufficiently large, the theoretical speedup factor is similar to the number of neurons. In (Domeika and Page, 1996), the authors propose some techniques which are specific for HNNs. They have observed that neurons almost share the same evaluation expression. They compute a common expression for several neurons, then a small expression is subtracted to the global expression for each neurons. This technique factorizes evaluations of several neurons. While these methods improve the evaluation time of a HNN, they do not modify the convergence properties of a HNN. Although we propose another way to improve the convergence time, these methods can also be used together with our method.

In (Domeika and Page, 1996), several neurons are evaluated in parallel, on different processing units. Because, in general HNN implementations include more neurons than processing units, several sets of neurons are created. All neurons of a set are sequentially executed on the same processor. Then, the proposed method to create these sets try to group some similar neurons in order to share some parameters, such as the value of weights. But, the network convergence problem is not taken into account in this work.

(Mańdziuk, 2002) presents an important review of HNN used to solve the N-queens problem. For this problem, it has been shown that the initial Hopfield constraint on the self-feedback connection significantly decreases the quality of results. Some negative self-feedback connections are used, the convergence of the HNN is no longer guaranteed.

Finally, we can also note that authors of (Wilson, 2009) proposed a HNN which is able to represent the behavior of several HNNs. Because they do not improve the convergence time of a HNN, this work is out of the scope of this article.

4 PARALLEL NEURAL NETWORK EVALUATION

In this section, we first present a convergence proof of the HNN parallel evaluation. This proof shows that if the connection weight between two neurons is positive (i.e. greater than or equal to 0), these neurons can be evaluated in parallel without affecting the convergence property. The second part of this section presents a way to build a HNN that respects rules to ensure convergence.

4.1 Convergence of the HNN

A HNN is defined to ensure convergence towards one solution of the problem. The convergence property means that the network evolves to a stable state where the energy function has reached a local minima. To ensure this convergence, the neural network must respect some constraints that are defined in the initial Hopfield article (Hopfield and Tank, 1985) for the sequential mode: the connection matrix must be symetric and its diagonal elements must be positive.

Convergence constraints appear from the proof of the network convergence and the proof is strongly bound to the chosen evaluation mode. In this section, we develop from (Kamp and Hasler, 1990) a convergence proof of a HNN using a parallel evaluation.

To be able to evaluate neurons in parallel, the state vector X is partitioned into K blocks of arbitrary size such as

$$X^T = \left[X_1^T, X_2^T, \ldots, X_K^T\right], \quad (3)$$

where X^T is the transposed vector of X representing neuron states. Then, the connection matrix $\mathbf{W}$ and the input vector I are partitioned in the same way.

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} & \ldots & \mathbf{W}_{1K} \\ \mathbf{W}_{21} & \mathbf{W}_{22} & \ldots & \mathbf{W}_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{W}_{K1} & \mathbf{W}_{K2} & \ldots & \mathbf{W}_{KK} \end{bmatrix} I = \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_K \end{bmatrix} \quad (4)$$

Neurons belonging to the same block X_k are evaluated in parallel and all the blocks are evaluated sequentially. The evaluation order of these blocks does not affect the convergence and could be random. To simplify notations, we consider a sequential evaluation order of all blocks X_k from 1 to K. From Eq. (1), the neuron update of the block X_k becomes

$$X_k(t+1) = H\left(\sum_{i=1}^{k-1} X_i(t+1) \times \mathbf{W}_{ki} + \sum_{i=k}^{K} X_i(t) \times \mathbf{W}_{ki} - I_k\right), \quad (5)$$

where $X_k(t)$ denotes the evaluation of X_k at iteration t. To evaluate the block $X_k(t+1)$, the value at iteration $t+1$ of blocks $X_1, \ldots, X_{k-1}$ and the value at iteration t of blocks $X_k, \ldots, X_K$ are used.

To use this parallel evaluation mode, the network convergence must be verified with a parallel evaluation of neurons. In (Kamp and Hasler, 1990), a theorem about the convergence with a parallel evaluation is proposed. By using a parallel dynamics, the HNN converges to a fixed point if the matrix $\mathbf{W}$ is symmetric and if diagonal blocks $\mathbf{W}_{kk}$ are positive or equal to zero.

To prove this convergence, the Lyapunov theorem is used and we show that the energy function is strictly decreasing during the network evolution. Hopfield proposed to use the following energy function to prove the network convergence

$$E(X) = -1/2 \sum_i \sum_j W_{ij} \times X_i \times X_j - \sum_i X_i \times I_i. \quad (6)$$

To verify if the energy function is decreasing, the sign of the difference between two successive iterations is evaluated. Without loss of generality, we consider that the first block of X is evaluated. To simplify notations, we rewrite the state vector X, the connection matrix $\mathbf{W}$ and the input vector I as

$$X = \begin{bmatrix} X_1 \\ X' \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{V}^T \\ \mathbf{V} & \mathbf{W}' \end{bmatrix}, I = \begin{bmatrix} I_1 \\ I' \end{bmatrix}. \quad (7)$$

It is important to note that the matrix $\mathbf{W}$ is supposed symetric to achieve the proof. Moreover, it is not a strong limitation because connection values are naturally symetric when a HNN is built to solve a problem. This constraint is studied and relaxed in (Xu et al., 1996).

From Eqs. (3), (7) and (6), we can express the difference between two successive iterations as:

$$\Delta(E(X)) = E(X(t+1)) - E(X(t)) =$$

$$-\overbrace{\left[X_1^T(t+1) - X_1^T(t)\right]}^{A_1}\overbrace{\left[\mathbf{W}_{11}X_1(t) + \mathbf{V}^T X'(t) + I_1^T\right]}^{A_2}$$

$$-\tfrac{1}{2}\overbrace{\left[X_1^T(t+1) - X_1^T(t)\right]}^{B_1}\mathbf{W}_{11}\overbrace{\left[X_1^T(t+1) - X_1^T(t)\right]}^{B_1}. \quad (8)$$

$\Delta(E(X))$ has to be negative to prove the convergence of the HNN evaluation using K parallel blocks. If $X_1^T(t+1) = X_1^T(t)$, then a fixed point is reached and the HNN did not evolve. In the following, we consider that $X_1^T(t+1) \neq X_1^T(t)$. In this case, because elements of X_i belong to $\{0,1\}$, $X_1^T(t+1) - X_1^T(t)$ can be equal to -1 or 1.

If products $A_1 \times A_2$ and $B_1 \times W_{11} \times B_1$ are both positives, $\Delta(E(X))$ is negative and therefore the function $E(X)$ is proved decreasing. In the following the sign of these two products is studied.

Concerning the product $A_1 \times A_2$ of Eq. (8), from Eq. (5), we have

$$X_1^T(t+1) = H(\mathbf{W}_{11}X_1(t) + V^T X'(t) + I_1^T) = H(A_2). \quad (9)$$

Then, from Eq. (2), if an element i of $X_1^T(t+1)$ is equal to 0, the element i of A_2 is then negative. Because we consider that $X_1^T(t+1) \neq X_1^T(t)$, when an element i of $X_1^T(t+1)$ is equal to 0, the element i of $X_1^T(t)$ is equal to 1, and the element i of A_1 is then negative (equal to -1). In this case, A_1 and A_2 are both negatives and their product is positive.

We have shown that when an element i of $X_1^T(t+1)$ is equal to 0, the product $A_1 \times A_2$ is positive. An analogical reasoning can be applied to an element i of $X_1^T(t+1)$ that is equals to 1. Therefore, we can conclude that elements of $A_1 \times A_2$ are always positive.

Concerning the product $B_1 \times \mathbf{W}_{11} \times B_1$ of Eq. (8), the sign of an element depends on the sign of $\mathbf{W}_{11}$. If all elements of $\mathbf{W}_{11}$ are positive then $B_1 \times \mathbf{W}_{11} \times B_1$ is positive.

Finally, because $A_1 \times A_2$ and $B_1 \times \mathbf{W}_{11} \times B_1$ are always positive, the sign of $\Delta(E(X))$ is negative and hence the energy function Eq. (6) is strictly decreasing until a fixed point is reached. By the Lyapunov theorem, we can conclude that the network reaches a fixed point if

- the matrix $\mathbf{W}$ is symmetric, and
- the diagonal blocks $\mathbf{W}_{kk}$ are positive.

This theorem is a sufficient but not necessary condition to ensure convergence. We are working on a more indepth study of the energy function in order to exhibit less restrictive constraints.

4.2 Application to Optimization Problem

Since the needed constraints are known, it is now possible to explain how a HNN can be evaluated in parallel for an optimization problem. To evaluate the HNN in parallel, we have to build some packets of independent neurons. Neurons belonging to a packet are evaluated simultaneous.

We can note that in our HNN applications, we never use strictly positive connections. Thus, we consider that we can build a packet of parallel neuron if connections between these neurons are equal to zero.

Figure 2 shows a connection matrix example of a HNN containing six neurons. This connection matrix contains three diagonal blocks with elements equal to zero. Then, neurons of this network can be grouped into three parallel packets: $\{1,2\}$, $\{3\}$ and $\{4,5,6\}$.

From Section 4.1, the diagonal block of the matrix must be positive to ensure that the network reaches a stable state. Thus, if the connection values in the diagonal block are equal to zero, this constraint is satisfied.

To construct packets of neurons which could be evaluated in parallel, we have to find neurons that are not connected. The next section presents the construction of packets on a example.

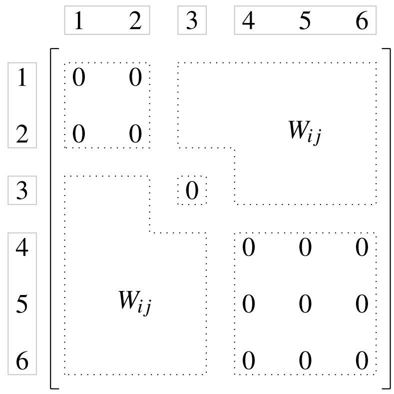

Figure 2: Example of a connection matrix for a HNN evaluated in parallel. Diagonal matrix elements are equal to zero. All other elements W_{ij} are negative or equal to zero.

5 APPLICATION TO A SCHEDULING PROBLEM

To illustrate the parallel evaluation of a HNN, the single machine scheduling problem (Sidney, 1977) is treated. It consists in determining for each scheduler "tick" which task has to be executed. Then, the HNN aims at finding a valid task scheduling scenario on a period which corresponds to the sum of execution time of all tasks.

Figure 3 presents an example of the neural model of the scheduling problem. In this example, the scheduling period p is equal to six scheduler "ticks", and there are four tasks. The neural network contains 6×4 neurons, one neuron for each task and each "tick". An activated neuron means the associated task will be executed at the corresponding "tick".

To define connections and input values of the HNN, some *k-outof-n* rules defined in (Tagliarini et al., 1991) are used. A *k-outof-n* rule ensures that k neurons are active among n when a stable state is reached.

All used *k-outof-n* rules are represented by dotted rectangles in Figure 3. All neurons belonging to a rectangle form a complete digraph. For each task, a *k-outof-n* rule is applied with n equals to the number of cycles needed to execute the task on the processor, and k is set to the required execution ticks for each task. Moreover for each tick, just one task can be executed, then a *1-outof-n* is applied on each column with n set to the number of tasks.

To build packets of neurons, disconnected neurons have to be selected. On Figure 3, we can note that there is no connection between diagonal neurons. For example, neurons $(3,0)$, $(2,1)$, $(1,2)$ and $(0,3)$ are disconnected, so they can form a packet. Thus, it is

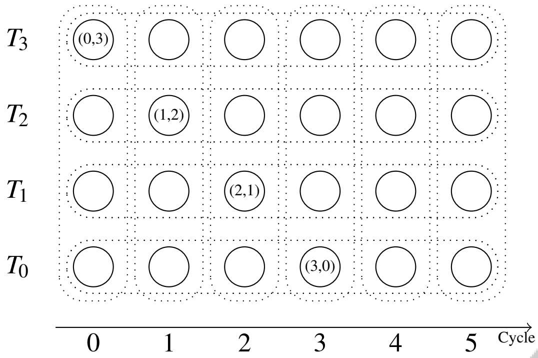

Figure 3: Graphical representation of all applied *k-outof-n* rules.

possible to group these neurons into a parallel packet.

6 EXPERIMENTATIONS

In this section, a comparison is presented between the sequential evaluation and the parallel evaluation of the HNN to exhibit the improvement factor provided by our parallelization method. The considered application is the scheduling problem presented in Section 5.

The first step is to build packets of neurons for the parallel evaluation of the network. Figure 4 presents the size of packets compared with the number of neurons. In our study example, the size of a packet is the number of tasks. To build a task set of size t, t tasks are generated with a random task duration belonging to $[1,5]$. The scheduling period is the sum of task durations in order to have enough time to schedule all tasks. Thus, the number of neurons belongs to $[t^2, 5 \times t^2]$. Figure 4 describes the data set used in the rest of this study.

The metric used for this comparison is the number of packet evaluations. When the sequential evaluation is used, a packet consists of one neuron. Then, to compare these two modes, we consider that we are able to evaluate a packet as fast as a neuron. In this case, the execution time of a HNN is strictly bound to the number of packet evaluations in both modes.

Figure 5 presents the results of several neural network evaluations. These evaluations are achieved with a HNN software simulator developped in our team. For each network, a sequential and a parallel evaluation achieved to show the gain obtained by our parallelization method. In both modes, packets are evaluated in a random order. Simulations stop when a stable state is reached without the need of a external controller. Then, the number of evaluations contains the last iteration which is needed to exhibit the stable state.

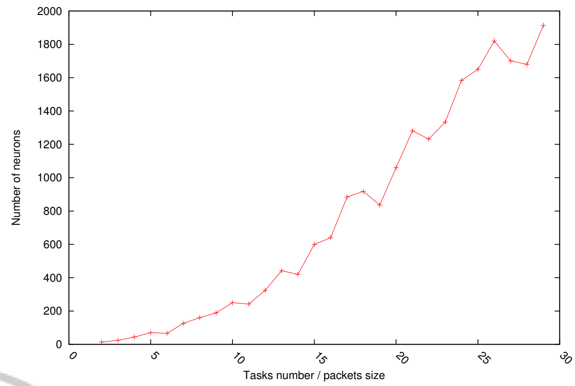

Figure 4: Size of packets compared with number of neurons. Because the size of packet is equal to the number of task, the abscissa represents the task number or the packet size.

Figure 5 shows that the improvement is really high and moreover increases with the number of neurons because the size of packets depends on the number of neurons. As a example, for about 2000 neurons, the parallized version is 38 times faster than the sequential version.

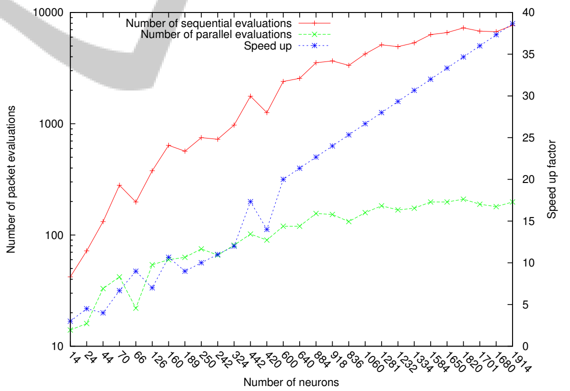

Figure 5: Number of packets evaluations for a sequential and a parallel evaluation.

The speedup factor presented in Figure 5 supposes that architecture executing the HNN has enough resources to evaluate in parallel all neurons present in a packet. Otherwise, it is necessary to split all packets into several sub-packets which increases the execution time.

From Section 5, the number of neurons in a parallel packet is equal to the number of tasks t. Considering n as the number of neurons in a HNN, the number of packets is equal to $\frac{n}{t}$. The number of packets in the sequential mode is t times higher than the number of packets in the parallel mode. Therefore, the speedup should be approximately equal to the task number t.

Figure 6 presents the speedup factor compared to the number of tasks. We can observe that the speedup factor is equal or higher than t. To reach the stable state, all neurons are evaluated several times. One evaluation of all neurons is called an *iteration*. The speedup factor could be higher than the number of tasks because the parallelization can decrease the number of iterations. This is mainly due to an appropriated neurons evaluation order. In the sequential mode, we can agree the diagonal evaluation order is the fastest order to reach a stable state: at two consecutive times, neurons corresponding to different tasks and ticks are evaluated. Concerning the parallel mode, a packet contains a diagonal of neurons, then neurons are implicitly evaluated diagonal by diagonal.

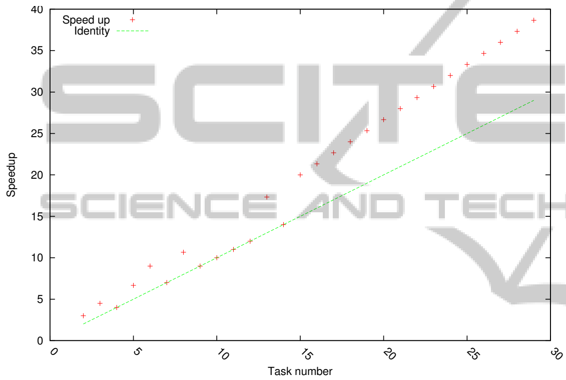

Figure 6: Speed up factor versus number of tasks.

The experiment results show a significant gain obtained by our parallelization method. Moreover, the number of iterations needed to reach a stable state is, in some cases, decreased by this method.

The methods presented in Section II allow for improving the sequential evaluation as well as the parallel evaluation, therefore, our speedup factor should not be impacted. Combined with our approach, they can further improve the evaluation time of the HNN without affecting the convergence property.

7 CONCLUSIONS

We presented a parallelization method to improve the convergence time of HNN to solve optimization problems. This approach has been applied on the scheduling problem which can be easily defined as an optimization problem. The HNN associated to this problem has been built by adding several constraints (such as k-outof-N rules) on some sets of neurons. We demonstrated that the network convergence is maintained when a subset of disconnected neurons is evaluated in parallel. This means that when two neurons do not belong to the same constraint, they can be evaluated in parallel. Because the construction of a HNN based on the addition of several constraint rules is really common, we assume that this method can be used for large number of optimization problems modelled by HNNs.

The parallelization of neural evaluations leads to an important improvement of the convergence time. We have seen that on the task scheduling problem, the speedup depends on the number of tasks. Thus, for a scheduling problem with 20 tasks, the speedup is about 25. Contrary to other works about parallel evaluation of a HNN, our method preserves the convergence property which permits to simplify the implementation of a HNN.

REFERENCES

Del Balio, R., Tarantino, E., and Vaccaro, R. (1992). A parallel algorithm for asynchronous hopfield neural networks. In *IEEE International Workshop on Emerging Technologies and Factory Automation*, pages 666 –669.

Domeika, M. J. and Page, E. W. (1996). Hopfield neural network simulation on a massively parallel machine. *Information Sciences*, 91(1-2):133 – 145.

Hopfield, J. and Tank, D. (1985). "Neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152.

Kamp, Y. and Hasler, M. (1990). *Recursive neural networks for associative memory*. John Wiley & Sons, Inc.

Mańdziuk, J. (2002). Neural networks for the N-Queens problem: a review. *Control and Cybernetics*, 31(2):217–248.

Sidney, J. (1977). Optimal single-machine scheduling with earliness and tardiness penalties. *Operations Research*, 25(1):62–69.

Smith, K. (1999). Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research. *Informs Journal on Computing*, 11:15–34.

Tagliarini, G., Christ, J., and Page, E. (1991). Optimization using neural networks. *IEEE Transactions on Computers*, 40(12):1347–1358.

Wang, C., Wang, H., and Sun, F. (2008). Hopfield neural network approach for task scheduling in a grid environment. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 04*, pages 811–814.

Wilson, R. C. (2009). Parallel hopfield networks. *Neural Computation*, 21:831–850.

Xu, Z., Hu, G., and Kwong, C. (1996). Asymmetric Hopfield-type networks: theory and applications. *Neural Networks*, 9(3):483–501.