Scalability of a Neural Network for the Knight's Tour Problem

Ian Parberry*
Department of Computer Sciences
University of North Texas

October 15, 1997

Abstract

The effectiveness and efficiency of a Hopfield-style neural network recently proposed by Takefuji and Lee for the knight's tour problem on an $n \times n$ board are compared and contrasted with standard algorithmic techniques using a combination of experimental and theoretical analysis. Experiments indicate that the neural network has poor performance when implemented on a conventional computer, and it is further argued that it is unlikely to improve significantly when implemented in parallel.

Keywords: Knight's tour problem, neural network, parallel algorithm, Hamiltonian cycle problem.

1 Introduction

A *knight's tour* is a cyclic sequence of moves made by a knight visiting every square of an $n \times n$ chessboard exactly once. The *knight's tour problem* is the problem of constructing a single such tour, given n. Takefuji and Lee [19] (see also Takefuji [18, Chapter 7]) recently proposed a neural network for the knight's tour problem. Their paper contains no analysis of the running time of their neural network (aside from a few tangential and cryptic comments), nor do they discuss its effectiveness beyond providing some examples of knight's tours for boards of size up to 20×20. We rectify their omission by comparing their neural network with conventional algorithms and come to the conclusion that the neural network is neither effective nor efficient.

Takefuji and Lee's [19] paper concentrates on finding single knight's tours on rectangular boards, but the examples are all square boards. We will make the following restrictions in order to simplify the presentation of this paper. Firstly, we limit ourselves to square boards,

*Research supported by the National Science Foundation under grant number CCR–9302917, and by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number F49620–93–1–0100. Author's address: Department of Computer Sciences, University of North Texas, P.O. Box 13886, Denton, TX 76203–3886, U.S.A. Electronic mail: ian@ponder.csci.unt.edu.

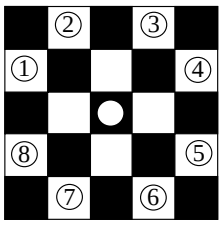

Figure 1: The 8 possible moves that a knight on the centre square can make.

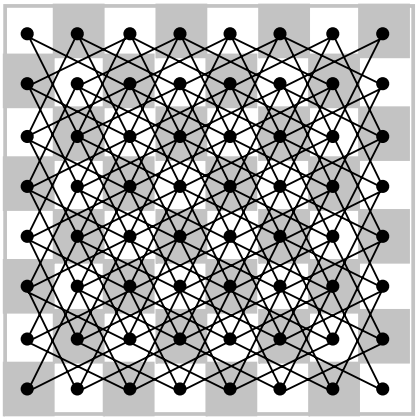

Figure 2: The knight's graph for an 8×8 board.

although it should be noted that the results extend equally well to rectangular boards. Secondly, for the most part we will consider the problem of generating a single knight's tour, however the problem of generating multiple distinct tours will be addressed later in the paper.

The knight's tour problem is a special case of the classic Hamiltonian cycle problem. Recall that a knight can make any of the eight moves described in Figure 1. Define the *knight's graph* for an $n \times n$ chessboard to be the graph $G = (V, E)$ where

$$\begin{aligned} V &= \{(i,j) \mid 1 \leq i,j \leq n\} \\ E &= \{((i,j),(k,\ell)) \mid \{|i-k|, |j-\ell|\} = \{1,2\}\}. \end{aligned}$$

That is, there is a vertex for every square of the board and an edge between two vertices exactly when there is a knight's move from one to the other. For example, Figure 2 shows the knight's graph for the 8×8 chessboard. Then, more formally, a knight's tour is defined to be a Hamiltonian cycle on a knight's graph.

Takefuji and Lee [19] state that they are unsure whether the problem of finding a knight's tour is $\mathcal{NP}$-complete. (Let $\mathcal{P}$ denote the set of problems computable in time at most some polynomial of the length of their input. The $\mathcal{NP}$-complete problems are in a certain technical sense the hardest members of a large and rich class $\mathcal{NP}$ of combinatorial optimization

problems. If any $\mathcal{NP}$-complete problem were proved to be computable in polynomial time, then $\mathcal{P} = \mathcal{NP}$. It is widely conjectured that $\mathcal{P} \neq \mathcal{NP}$; see [9].) However, it is well-known that the knight's tour problem cannot be $\mathcal{NP}$-complete (unless $\mathcal{P} = \mathcal{NP}$). Dudeney [7, 8] contains a description of exactly which rectangular chessboards have knight's tours; in particular, an $n \times n$ chessboard has a knight's tour iff $n \geq 6$ is even. (It is easy to see that there can be no knight's tour when n is odd since such a board has one more white square than black, or vice-versa, and since the colours of the squares visited on a knight's tour must alternate.) This fact appears to be common knowledge in the research community (for example, the result is stated without references in Cole [3]). Furthermore, there exist several linear time (i.e. $O(n^2)$) algorithms for constructing knight's tours (see, for example, Cull [6] and Schwenk [17]).

The remainder of this paper is divided into four sections. Section 2 describes the Hopfield-style neural network of Takefuji and Lee and contains an experimental analysis of its effectiveness and efficiency. Section 3 describes a random walk algorithm obtained by combining two classical techniques due to Euler in 1759 and Warnsdorff in 1823 and contains an experimental analysis of its effectiveness and efficiency. Section 4 describes a divide-and-conquer algorithm for constructing knight's tours and contains experimental and theoretical analyses of its effectiveness and efficiency. Section 5 addresses the significance of the experimental data, and considers the questions of parallel implementation, and of counting the number of knight's tours.

Throughout this paper, $\mathbb{N}$ denotes the set of natural numbers (including zero), $\mathbb{Z}$ denotes the set of integers, and $\mathbb{B}$ denotes the Boolean set $\{0, 1\}$. The programs used in the experiments were written in Pascal, compiled using the Berkeley Unix Pascal compiler, and executed on a SUN SPARCstation 2. The randomized algorithms were implemented using the standard linear congruential pseudorandom number generator provided in Berkeley Unix Pascal, seeded with the time of day. The drawings of knight's tours in the remainder of this paper were created by processing the tour matrix output of programs that implement the algorithms with a simple Pascal program that produces LaTeX `picture` environments.

2 The Neural Network

The neural network devised by Takefuji and Lee [18, 19] for the knight's tour problem uses $O(n^2)$ neurons, one for each edge in the knight's graph for an $n \times n$ board. There is a bidirectional connection between two neurons whenever the two corresponding edges share a vertex. Although the architecture is that of a Hopfield network (Hopfield [10]) with all weights equal to unity, the operation of the neural network differs in the use of a so-called "hysteresis" term.

More specifically, suppose the squares of the board are numbered 1 through n^2 in some manner (for example, row-major order), and let

$$K = \{(i, j) \mid 1 \leq i < j \leq n^2, \text{ and there is a knight's move between squares } i \text{ and } j\}.$$

Takefuji and Lee use a neuron $N_{i,j}$ to represent each knight's move $(i, j) \in K$. Let $G(N_{i,j})$ denote the neighbours of neuron $N_{i,j}$ in the interconnection graph of the neural network.

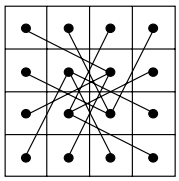

Figure 3: A configuration that selects these edges will be stable.

That is,

$$G(N_{i,j}) = \{N_{i,k} \mid (i,k) \in K\} \cup \{N_{k,j} \mid (k,j) \in K\}.$$

Time is divided into discrete intervals. Each neuron has an internal *state* and an external *output*. If $t \in \mathbb{N}$ and N is a neuron, let $U_t(N) \in \mathbb{Z}$ denote the state of neuron N at time t, and $V_t(N) \in \mathbb{B}$ denote its output at time t. At each time interval one neuron updates its state and output. Suppose that it is neuron $N_{i,j}$. The update rule is as follows:

$$\begin{aligned} U_{t+1}(N_{i,j}) &= U_t(N_{i,j}) + 2 - \sum_{N \in G(N_{i,j})} V_t(N), \\ V_{t+1}(N_{i,j}) &= \begin{cases} 1 & \text{if } U_{t+1}(N_{i,j}) > 3 \\ 0 & \text{if } U_{t+1}(N_{i,j}) < 0 \\ V_t(N_{i,j}) & \text{otherwise.} \end{cases} \end{aligned}$$

A standard Hopfield network would have $V_{t+1}(N_{i,j}) = 1$ if $U_{t+1}(N_{i,j}) \geq 0$; the use of the more restrictive condition $U_{t+1}(N_{i,j}) > 3$ is called a *hysteresis term*.

Takefuji and Lee's neural network is carefully constructed to find a subgraph in the knight's graph of an $n \times n$ chessboard. We say that an edge $(i,j) \in K$ is *selected* for this subgraph at time t if $V_t(i,j) = 1$. The intuition behind the neural network is as follows. A *configuration* of the neural network is defined to be a list of the states of all of the neurons. A configuration is said to be *stable* if it does not change over time. A close examination of the above update rules will show that any configuration in which a subgraph of degree 2 is selected is stable. Hence, a knight's tour will be stable. Unfortunately, other unusual subgraphs will also be stable, such as the one shown in Figure 3 for a 4×4 board.

Takefuji and Lee's paper describes their neural network by physical analogy using a set of "equations of motion" (a concept that appears to have originated with Takefuji), instead of using standard algorithmic concepts and notation as described above. Unfortunately, they describe neither the initial nor the final conditions. It can be inferred from Takefuji and Lee's discussion that their neural network is a randomized algorithm (that is, it gives different outputs on different executions), but they neglect to describe how this randomness is achieved. We found that performance similar to that claimed by Takefuji and Lee is obtained when $V_0[i,j]$ is set using a pseudorandom number generator, and $U_0[i,j]$ is set to zero for all $1 \leq i < j \leq n^2$. We defined convergence to have occurred when there is no further change to U. This is a naive definition since it may be the case that U continues to change, while V remains fixed (for example, two isolated neurons connected by an edge will have their U values increase unboundedly while their V values remain at 1). However,

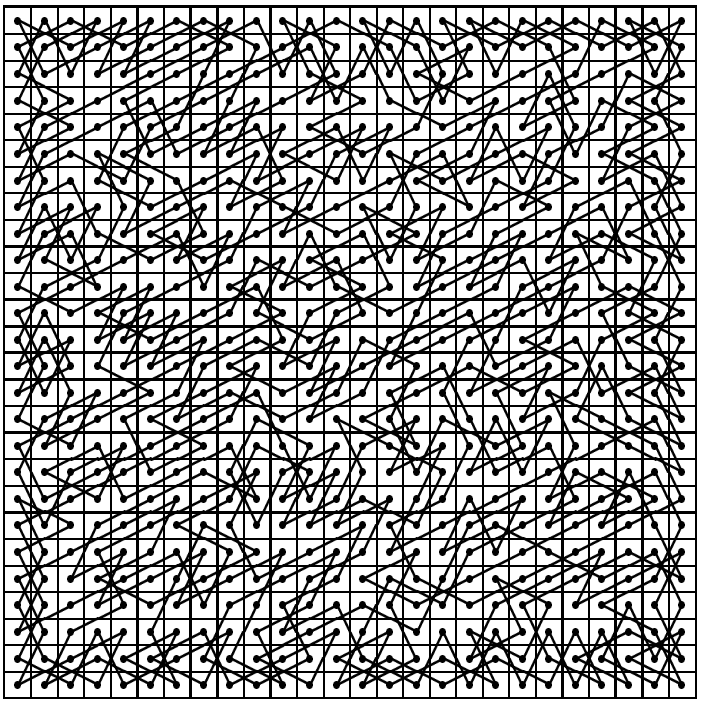

Figure 4: The largest knight's tour (26×26) found by Takefuji and Lee's neural network.

experiments showed that the following more sophisticated definition of convergence did not appear to improve the network's performance. A neuron (i, j) is *stable* at time t if either $V_t[i, j] = 1$ and $U_{t+1}[i, j] \geq U_t[i, j]$, or $V_t[i, j] = 0$ and $U_{t+1}[i, j] \leq U_t[i, j]$. The network is said to have *weakly converged* at time t if all neurons are stable.

Another detail not described by Takefuji and Lee is the update schedule used in their simulations. Standard Hopfield networks are guaranteed to converge if they are updated in sequential mode (one neuron at a time) but may not do so when updated in parallel mode (all neurons updated simultaneously in lock-step). Unfortunately, the hysteresis term destroys this guarantee of convergence (either regular or weak) in sequential mode. Experiments in parallel mode failed to converge almost all of the time, but experiments in sequential mode showed that convergence occurs often enough to generate knight's tours for small n. We therefore chose to update the neurons sequentially in the following order: for each square s of the board in row-major order, the neurons representing moves out of s are updated in some convenient order. The period taken to update every neuron once in turn is usually termed an *epoch* in neural network terminology. Since Takefuji and Lee reported that the average number of epochs was around 100 in their simulations, we abandoned any computation that proceeded beyond 1000 epochs, thus avoiding computations that became locked in a cycle. Some quite long cycles were observed in practice, unlike many other variants of Hopfield networks (see, for example, Bruck and Goodman [2]. Poljak and Sura [16], Poljak [15] and Odlyzko and Randall [12]).

We were able to use Takefuji and Lee's neural network to find knight's tours on an $n \times n$

n	trials	success	failure	diverge	time
6	20000	6103	13561	336	7.6 min
8	20000	3881	15744	375	26.6 min
10	20000	2676	16827	497	1.2 hr
12	20000	1394	17922	684	2.6 hr
14	20000	703	18390	907	5.0 hr
16	20000	305	18618	1077	9.3 hr
18	20000	147	18644	1209	15.7 hr
20	20000	65	18516	1419	22.7 hr
22	40000	44	36817	3139	3.0 day
24	40000	10	36339	3651	4.4 day
26	40000	1	35570	4429	5.5 day

Table 1: Results of experiments with Takefuji and Lee's neural network on an $n \times n$ board. Columns list from left to right n, the number of experiments, the number of experiments that converged to knight's tours, the number of experiments that converged to non-knight's tours, the number of experiments that appeared to diverge, and the total amount of CPU time used.

n	success	failure	diverge	time
6	30.5%	68.0%	1.7%	0.08 sec
8	19.4%	78.7%	1.9%	0.4 sec
10	13.4%	84.1%	2.5%	1.6 sec
12	7.0%	89.7%	3.4%	6.6 sec
14	3.5%	92.0%	4.5%	25.5 sec
16	1.5%	93.1%	5.4%	1.8 min
18	0.7%	93.2%	6.0%	6.4 min
20	0.3%	92.6%	7.1%	20.9 min
22	0.1%	92.0%	7.9%	1.7 hr
24	0.03%	90.9%	9.1%	10.4 hr
26	0.003%	88.9%	11.1%	5.5 day

Table 2: Results of experiments with Takefuji and Lee's neural network on an $n \times n$ board. Columns list from left to right n, the percentage of experiments that converged to knight's tours, the percentage of experiments that converged to non-knight's tours, the percentage of experiments that appeared to diverge (note that these may not sum to 100% due to rounding errors), and the average amount of CPU time needed to find a tour.

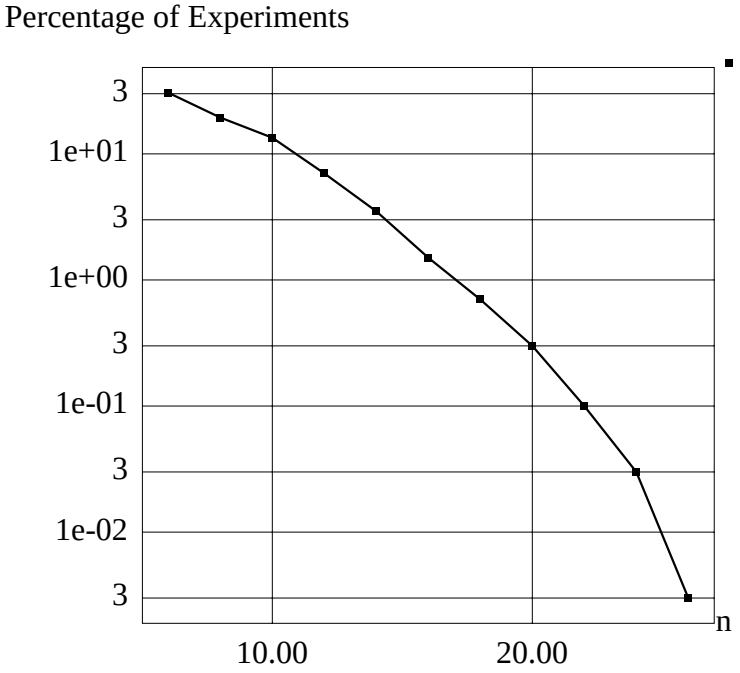

Figure 5: Success rate for Takefuji and Lee's neural network as a percentage of the number of experiments, shown with a log scale on the Y-axis.

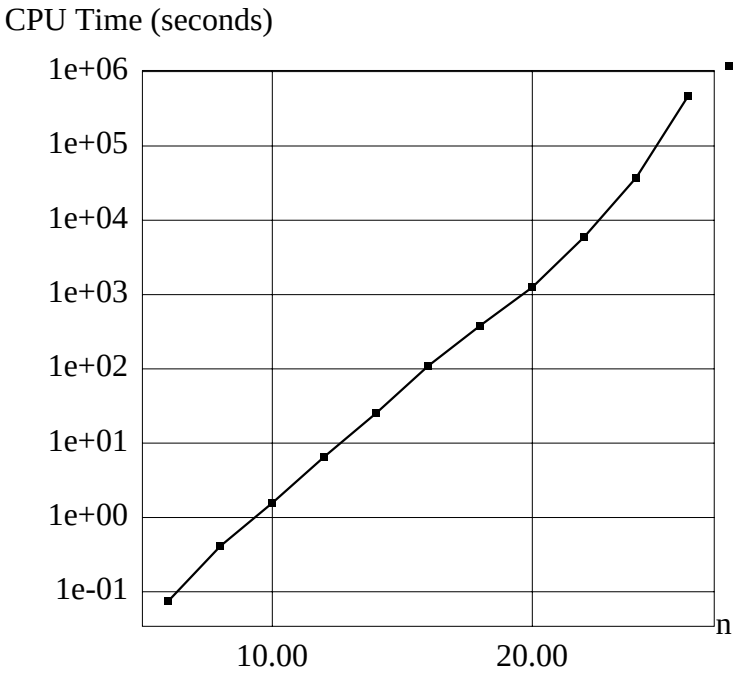

Figure 6: Average running time required for Takefuji and Lee's neural network to find a single knight's tour, shown with a log scale on the Y-axis.

board for all even $6 \leq n \leq 26$ (for example, Figure 4 shows a knight's tour on a 26×26 board). Takefuji and Lee report being able to generate knight's tours for n only as large as 20, and do not give any data on the success rate or running time of their neural network. The raw data from our experiments is summarized in Tables 1 and 2. The networks appeared to converge 90% of the time. However, as can be seen from Figures 5 and 6, the neural network technique scales very poorly. The experimental evidence indicates that number of successes decreases exponentially with n, and that the expected running time (defined to be the average amount of time required to generate a single knight's tour) increases exponentially with n. Note that the figures for the larger values of n are progressively less significant due to the paucity of solutions. Based on this empirical evidence, we consider it very unlikely that Takefuji and Lee's neural network can be used to find a knight's tour for $n = 30$ even with a one-thousandfold increase in computing power.

3 A Random Walk Algorithm

Rouse Ball and Coxeter [1] describe an interesting random walk algorithm which they attribute to Euler. This algorithm starts at any square and repeatedly makes a knight's move to a random unvisited square until no further progress can be made. This leaves a possibly large number of unvisited squares (which we will call *holes*). The holes are then patched by stringing them together using knight's moves whenever possible, and then attempting to insert the strings of holes into the tour by deleting a move and replacing it with one of the strings. The tour is then closed, if possible.

There is a similar random walk algorithm that proceeds by making random moves chosen using an interesting heuristic. Instead of choosing a completely random move, the algorithm chooses a move at random from the set of moves that lead to a square that has remaining the smallest number of legal moves to an unvisited square. This heuristic is attributed to Warnsdorff in 1823 by Conrad, Hindrichs, Morsy, and Wegener [4, 5]. The latter paper adds the heuristic to the standard backtracking algorithm for knight's tours.

We experimented with a random walk algorithm that uses both the technique of Euler and that of Warnsdorff. The result of the experiments appear in Table 3. Although the success rate appears to decreases exponentially with n (see Figure 7), and the average running time required to find each tour appears to increase exponentially with n (see Table 8), the random walk algorithm clearly outperforms Takefuji and Lee's neural network by several orders of magnitude. The largest knight's tour that we were able to find using the random walk algorithm was 78×78, which is considerably larger than the 26×26 tour, which was the best that the neural network could do even with the investment of considerably more running time. Figure 9 shows a 60×60 knight's tour found by the random walk algorithm.

4 A Divide-and-Conquer Algorithm

There exist many conventional algorithms for constructing knight's tours (see, for example, Cull [6] and Schwenk [17]). A particularly elegant divide-and-conquer algorithm proceeds as follows. To construct a knight's tour on an $n \times n$ or $n \times (n + 2)$ board, divide it into four

n	trials	success	percent	time	average
6	10000	9206	92.1	41 sec	0.005 sec
8	10000	9284	92.8	1.3 min	0.008 sec
10	10000	9527	95.3	1.9 min	0.012 sec
12	10000	9604	96.0	2.8 min	0.018 sec
14	10000	9496	95.0	4.0 min	0.025 sec
16	10000	9211	92.1	5.3 min	0.034 sec
18	10000	8325	83.3	6.8 min	0.049 sec
20	10000	6840	68.4	8.4 min	0.074 sec
22	20000	10125	50.6	20.4 min	0.12 sec
24	20000	7322	36.6	23.1 min	0.19 sec
26	20000	5079	25.4	26.0 min	0.31 sec
28	20000	3438	17.2	29.4 min	0.51 sec
30	20000	2360	11.8	33.2 min	0.81 sec
32	20000	1608	8.0	36.6 min	1.4 sec
34	20000	1024	5.1	40.2 min	2.4 sec
36	20000	692	3.5	44.6 min	3.9 sec
38	20000	482	2.4	49.0 min	6.1 sec
40	20000	336	1.7	53.4 min	9.5 sec
42	20000	213	1.1	58.0 min	16.3 sec
44	20000	141	0.7	1.1 hr	26.9 sec
46	20000	99	0.5	1.2 hr	41.8 sec
48	20000	62	0.31	1.3 hr	1.2 min
50	20000	36	0.18	1.4 hr	2.3 min
52	20000	24	0.12	1.5 hr	3.7 min
54	20000	22	0.11	1.6 hr	4.3 min
56	20000	12	0.06	1.7 hr	8.3 min
58	20000	16	0.08	1.9 hr	7.3 min
60	20000	7	0.04	1.9 hr	8.1 min
62	20000	4	0.02	2.0 hr	30 min
64	20000	3	0.015	2.1 hr	42.3 min
66	20000	3	0.015	2.3 hr	45.1 min

Table 3: Results of experiments with random walk algorithm on an $n \times n$ board. Columns list from left to right n, the number of experiments, the number of experiments that successfully generated knight's tours, the percentage of experiments that generated knight's tours, the total amount of CPU time used, the average amount of CPU time needed to find a tour.

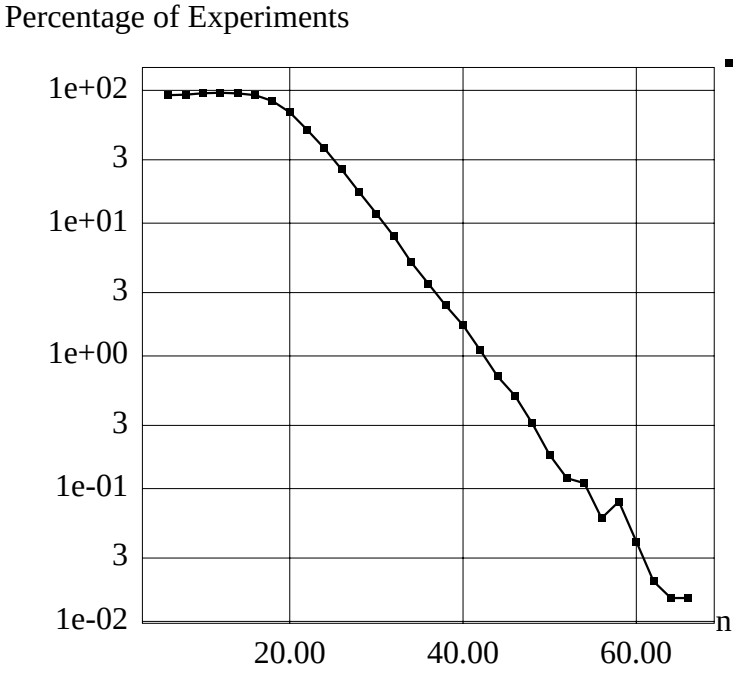

Figure 7: Success rate for the random walk algorithm as a percentage of the number of experiments, shown with a log scale on the Y-axis.

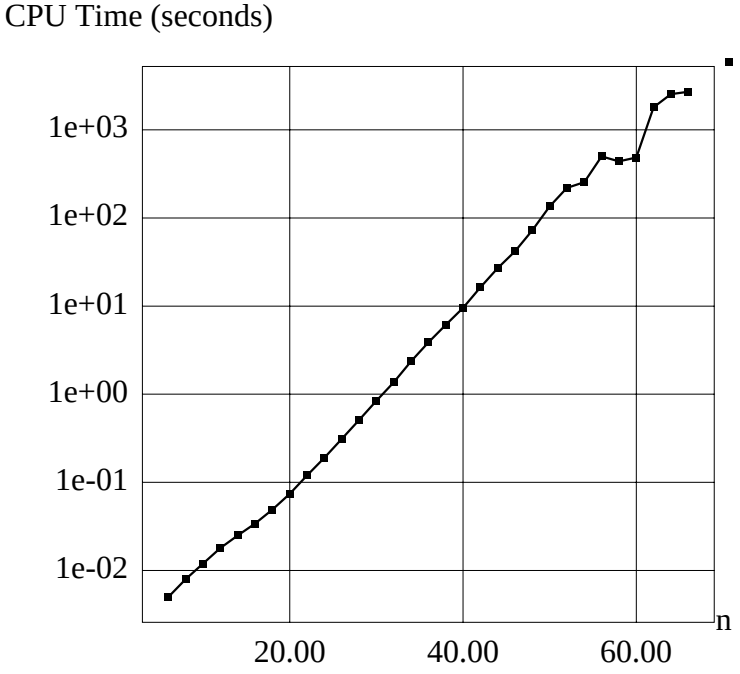

Figure 8: Average running time required for the random walk algorithm to find a single knight's tour, shown with a log scale on the Y-axis.

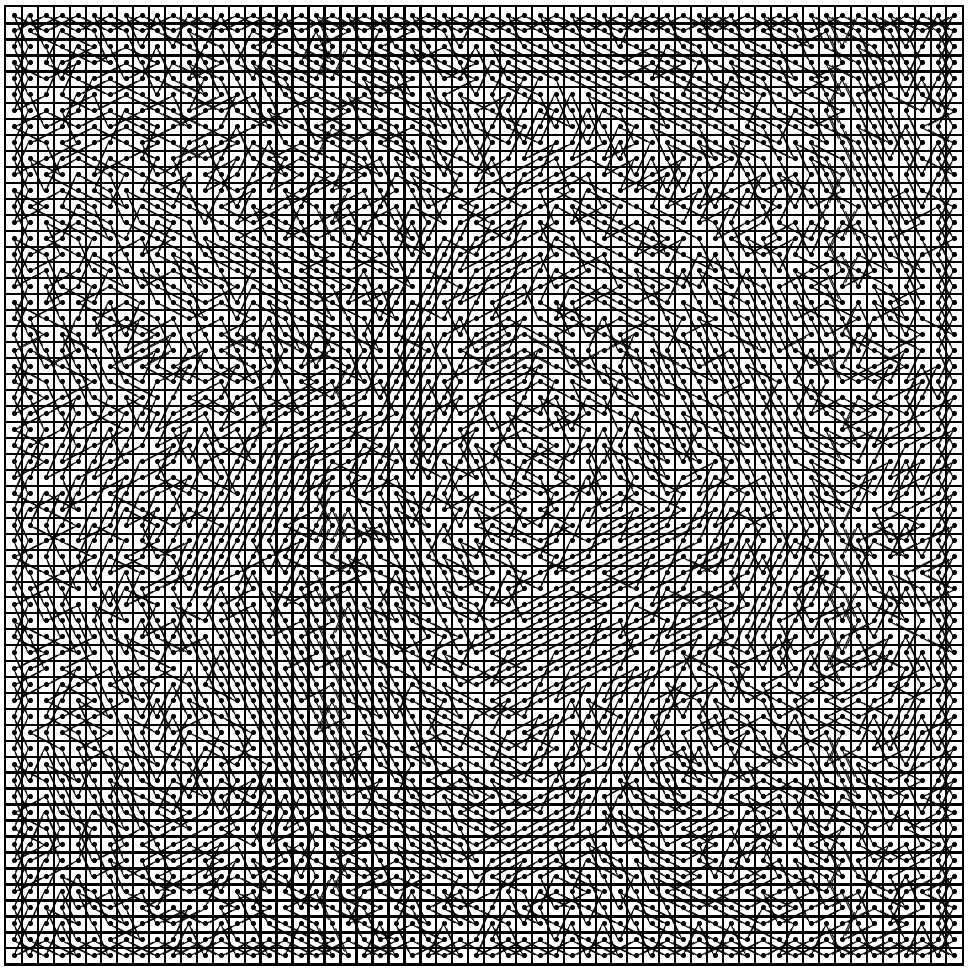

Figure 9: A 60×60 knight's tour found by the random walk algorithm.

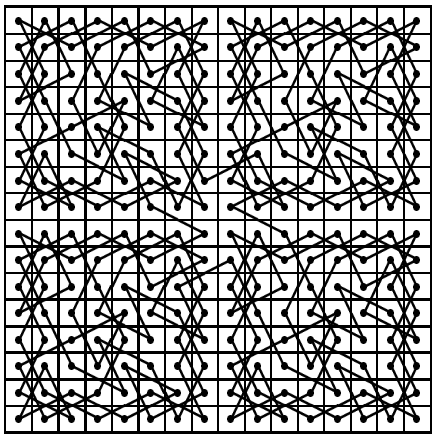

Figure 10: A 16×16 knight's tour constructed from an 8×8 knight's tour using a divide-and-conquer algorithm.

even-sided quadrants, construct a tour in each of the four quadrants, and then patch the four independent tours together to form a single tour. The base of the recursion consists of 6×6, 6×8, 8×8, 8×10 and 10×12 tours with a particular pattern of moves near the corners needed to facilitate the patching together of tours. The technical details are described more formally in Parberry [14, 13]. Figure 10 illustrates the technique on a 16×16 board, constructed from four copies of the knight's tour on an 8×8 board. Figure 11 illustrates the technique on a 60×60 board.

The running time $T(n)$ required for the divide-and-conquer construction of a knight's tour on an $n \times n$ board is given by the following recurrence: $T(8) = O(1)$, and for $n \geq 16$ a power of 2, $T(n) = 4T(n/2) + O(1)$. This recurrence has solution $T(n) = O(n^2)$. Therefore (using the standard argument), the running time for all even $n \geq 6$ is $O(n^2)$, that is, linear in the number of squares on the board. The divide-and-conquer algorithm is particularly easy to implement, and can be used to construct knight's tours of size up to 1000×1000 in under 11 seconds. The running time for the construction of a knight's tour on an $n \times n$ board for $10 \leq n \leq 1000$ is shown in Figure 12.

5 Analysis of Results

The experimental data presented thus far in this paper clearly indicate that the neural network is inferior to standard algorithmic techniques when implemented on a conventional computer. However, Takefuji and Lee [19] state that the major advantage of their neural network algorithm for the knight's tour problem is that it is a parallel algorithm. They do not state explicitly why it is important to have a parallel algorithm, but one of the more obvious effects of parallelism is an increase in computation speed. Nor do they indicate exactly how the neural network is to be run in parallel. The operation of the neural network is inherently sequential with one neuron update per cycle. However, proper behaviour can still be obtained when allowing multiple neuron updates in parallel provided no two edges that have a vertex

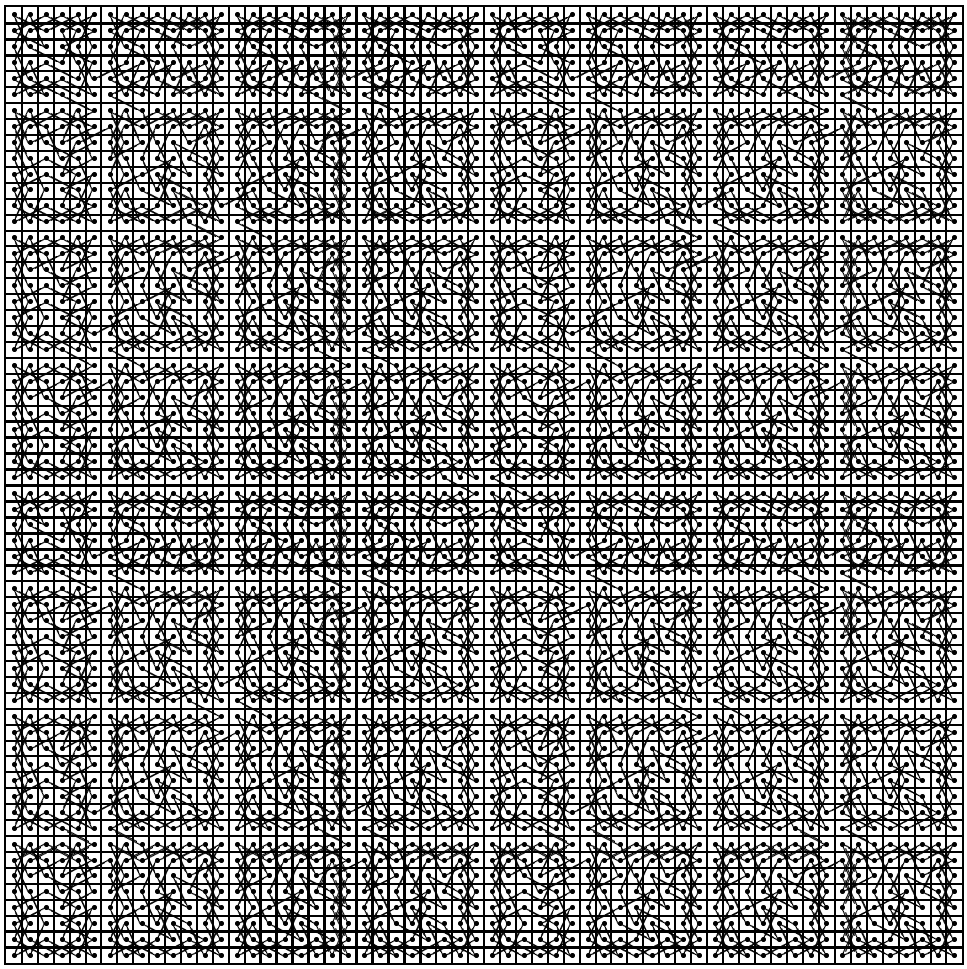

Figure 11: A 60×60 knight's tour constructed from the smaller knight's tours using a divide-and-conquer algorithm.

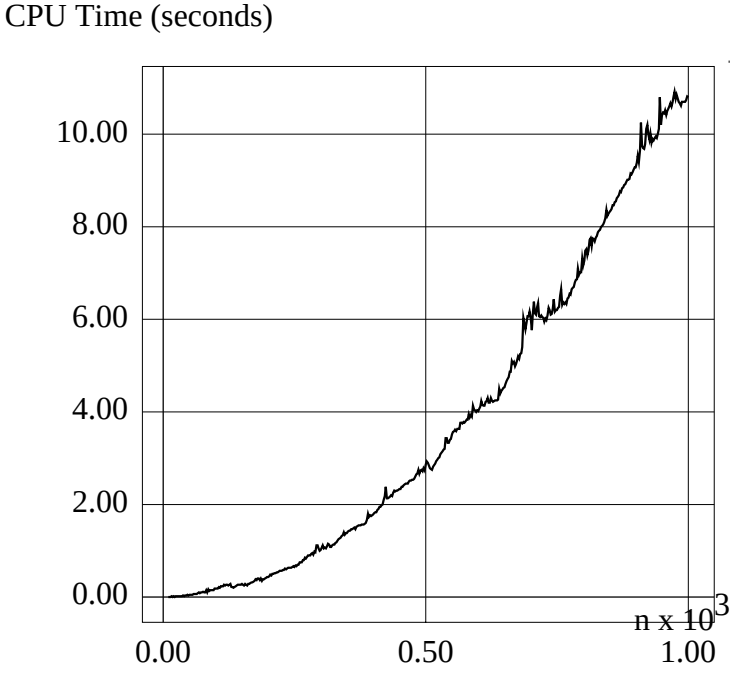

Figure 12: Running time required for the divide-and-conquer algorithm to find a single knight's tour.

Architecture	Time	Processors
Hypercubic network	$O(n^2/p)$	$p = O(n^2/\log n)$
CREW PRAM	$O(1)$	n^2
Mesh	$O(n^2/p^2)$	$p \leq n^{2/3}$
Mesh with CREW buses	$O(1)$	n^2

Table 4: Summary of major parallel algorithms.

in common are updated simultaneously. Each epoch can therefore be realized in parallel in 16 rounds (two for each of the 8 types of knight's move). This reduces the time required for each epoch from $O(n^2)$ on a sequential computer to $O(1)$ on a parallel computer, and hence reduces overall running time by a factor of $O(n^2)$. However, the experimental evidence in Section 2 indicates that the expected time to generate an $n \times n$ knight's tour is exponential in n^2, that is, as large as $2^{O(n^2)}$. This implies that the parallel implementation of the neural network will also take exponential time.

In sharp contrast, the divide-and-conquer algorithm is particularly amenable to parallel implementation in running time proportional to the *diameter* of the parallel machine, which can loosely be defined as the maximum amount of time needed to pass a message from one processor to another. A theoretical analysis of the parallel version of the divide-and-conquer algorithm is provided in Parberry [14, 13]. In that paper, four different types of parallel architecture are considered, the hypercubic network, the PRAM, the mesh-connected computer, and the mesh with row and column buses. The results are summarized in Table 4.

On first thought, it appears that the neural network and random walk algorithms still have one advantage over the divide-and-conquer algorithm: they can be used to generate a large number of knight's tours, and hence enable the determination of a lower bound on the number of knight's tours of a given size. Let $\mathcal{T}_n$ denote the number of distinct (though possibly isomorphic) knight's tours on an $n \times n$ board. A standard backtracking algorithm can be used to verify that $\mathcal{T}_6 = 9,862$, but determination of lower bounds on $\mathcal{T}_n$ for $n > 6$ require more sophisticated techniques. The neural network or random walk algorithm can be used to derive a lower bound on $\mathcal{T}_n$ by generating a large set of tours and removing duplicates. The random walk algorithm is the obvious choice since our earlier experiments have shown that it is clearly the more efficient of the two. We performed initial experiments as follows. The random walk algorithm was used to generate $100,000$ knight's tours of each size. Duplicates were then removed using the Unix utility `sort -u`, and number of lines were counted using `wc -l`. Preliminary experiments found, for example, that $\mathcal{T}_{10} \geq 95,006$, and $\mathcal{T}_{12} \geq 95,902$.

However, much better bounds can be found for general n using the recursive algorithm. For example, it can be shown that the the divide-and-conquer algorithm can generate at least 5.5×10^{17} tours on 12×12 boards, which implies that $\mathcal{T}_{12} \geq 5.5 \times 10^{17}$. This is much larger than the naive lower bound $\mathcal{T}_{12} \geq 95,902$ obtained in the previous paragraph. It is true that the neural network could be used to prove a similar lower bound, but this would require more disk space than is technologically feasible (even at one byte per tour, it would require more

than 10^8 gigabytes). A more sophisticated argument using the divide-and-conquer algorithm can also be used to show an asymptotic lower bound of $\mathcal{T}_n = \Omega(1.35^{n^2})$ (Kyek, Parberry, and Wegener [11]).

6 Conclusion

Neural networks research has progressed to the point at which the mere fact of existence of a neural network solution for a problem is no longer interesting. Long term interest in neural network research can only be sustained if neural networks can compete with conventional algorithm design techniques. A new algorithm is significant if it can be demonstrated beyond reasonable doubt to be more practical, more effective, or more efficient than existing techniques. We have found that this is not the case for a neural network for the knight's tour problem, neither for finding single tours sequentially or in parallel, nor for finding bounds on the number of tours. To quote Samuel Johnson in another context, a neural network for the knight's tour problem

> ...is like a dog's walking on its hinder legs. It is not done well; but you are surprised to find it done at all.

Acknowledgements

The author is indebted to Edwin Clark, Kevin Ford, and Rolf Wanka for helping with references, to Mike Sluyter for implementing the random walk algorithm.

References

[1] W. W. Rouse Ball and H. S. M. Coxeter. *Mathematical Recreations and Essays.* University of Toronto Press, 12th edition, 1974.

[2] J. Bruck and J. W. Goodman. A generalized convergence theorem for neural networks and its applications in combinatorial optimization. In *Proc. IEEE First International Conference on Neural Networks*, volume III, pages 649–656, San Diego, CA, June 1987.

[3] C. Cole. The `rec.puzzles` Frequently Asked Questions List. Version 3, available by anonymous `ftp` from `rtfm.mit.edu` in `/pub/usenet/news.answers/puzzles-faq`, 1992.

[4] A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener. Wie es dem springer gelang, schachbretter beliebiger groesse und zwischen beliebig vorgegebenen anfangs- und endfeldern vollstaendig abzuschreiten. *Spektrum der Wissenschaft*, pages 10–14, February 1992.

[5] A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener. Solution of the knight's Hamiltonian path problem on chessboards. *Discrete Applied Mathematics*, 50:125–134, 1994.

[6] P. Cull and J. DeCurtins. Knight's tour revisited. *Fibonacci Quarterly*, 16:276–285, 1978.

[7] H. E. Dudeney. *Amusements in Mathematics*. Thomas Nelson & Sons, 1917.

[8] H. E. Dudeney. *Amusements in Mathematics*. Dover Publications, 1970.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[10] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Sciences*, 79:2554–2558, April 1982.

[11] O. Kyek, I. Parberry, and I. Wegener. Bounds on the number of knight's tours. Unpublished Manuscript, 1994.

[12] A. M. Odlyzko and D. J. Randall. On the periods of some graph transformations. *Complex Systems*, 1:203–210, 1987.

[13] I. Parberry. Algorithms for touring knights. Technical Report CRPDC-94-7, Center for Research in Parallel and Distributed Computing, Dept. of Computer Sciences, University of North Texas, May 1994.

[14] I. Parberry. Efficient sequential and parallel algorithms for the knight's tour problem. Unpublished Manuscript, 1994.

[15] S. Poljak. Transformations on graphs and convexity. *Complex Systems*, 1:1021–1033, 1987.

[16] S. Poljak and M. Sura. On periodical behaviour in societies with symmetric influences. *Combinatorica*, 3(1):119–121, 1983.

[17] A. J. Schwenk. Which rectangular chessboards have a knight's tour? *Mathematics Magazine*, 64(5):325–332, 1991.

[18] Y. Takefuji. *Neural Network Parallel Computing*. Kluwer Academic Publishers, 1992.

[19] Y. Takefuji and K. C. Lee. Neural network computing for knight's tour problems. *Neurocomputing*, 4(5):249–254, 1992.

Biography

Dr. Ian Parberry is an Associate Professor on the faculty of the Computer Sciences Department at the University of North Texas, where he is also Director of the Center for Research in Parallel and Distributed Computing and a member of the Center for Network Neuroscience. His research interests include theoretical computer science, parallel computing, and neural networks. He is a member of the Association for Computing Machinery, the ACM Special Interest Group on Algorithms and Computation Theory, the IEEE Computer Society, the IEEE Technical Committee on Mathematical Foundations of Computing, the European Association for Theoretical Computer Science, and the International Neural Network Society. He is the author of three books, *Parallel Complexity Theory* (Pitman, 1987), *Circuit Complexity and Neural Networks* (MIT Press, 1994), and *Problems on Algorithms* (Prentice Hall, 1995). He is an Editor of *Information and Computation*, an Associate Editor of *Journal of Computer and System Sciences*, and the Editor of *ACM SIGACT News*.