PEPNet: Parallel Evolutionary Programming for Constructing Artificial Neural Networks[1]

Gerrit A. Riessen and Graham J. Williams
CSIRO Mathematical and Information Sciences
GPO Box 664 Canberra ACT 2601 Australia
Gerrit.Riessen@cmis.csiro.au
Graham.Williams@cmis.csiro.au

Xin Yao
Department of Computer Science
Australian Defence Forces Academy
Canberra ACT 2600 Australia
Xin@adfa.cs.adfa.oz.au

Abstract

This paper presents a description of an evolutionary artificial neural network algorithm, EPNet and its extension taking advantage of a High Performance Computing Environment. PEPNet, Parallel EPNet, implements four forms of parallelism and this paper describes two of those parallelisms. Experimental studies have shown promising results with better time and prediction performance.

1 Introduction

Artificial Neural Networks (ANNs) provide an important classification tool for Knowledge Discovery in Databases (KDD). Feed-forward fully-connected neural networks using a back propagation algorithm for weight adjustment are common. Unfortunately such ANNs require considerable time to train, particularly when large datasets are involved. Training time is also adversely affected when the characteristics of the dataset are not consistent with the chosen structure of the ANN. Finally, determining the best ANN network structure for a particular task remains a difficult art, with no hard and fast rules.

Evolutionary Artificial Neural Networks (EANNs) [1] take advantage of evolutionary techniques to address some of the problems associated with developing optimal ANNs. EANNs dynamically modify ANNs on the basis of their classification performance. EPNet [2] is a serial algorithm which adopts these ideas to produce efficient ANNs. Such techniques produce greater accuracy in the networks, but at the expense of further computational and storage requirements.

Parallel Evolutionary Artificial Neural Networks (PEANNs) have the potential to produce accurate networks in significantly less time using larger datasets than serial EANNs. PEPNet is a development of the EPNet algorithm oriented towards parallel architectures. PEPNet is being developed for use as a Data Mining tool, where very large datasets need to be analysed within restricted time frames. We describe two parallel architectures for the implementation of the PEPNet algorithm. These architectures represent initial attempts at parallelising the EPNet algorithm.

[1]Published in "Evolutionary Programming VI", Peter J. Angeline, Robert G. Reynolds, John R. McDonnell, and Russ Eberhart (Eds.), Lecture Notes in Computer Science, Volume 1213, Springer Verlag, 1997.

Data Mining, defined as the "non trivial extraction of implicit, previously unknown, and potentially useful knowledge from large datasets" [3], is part of the larger KDD process. Knowledge Discovery in Databases (KDD) describes the process of retrieving data from large databases, performing exploratory data analysis on that data, and developing models of the data. KDD consists of several major steps: data selection, data preprocessing, data mining, and evaluation [4]. PEPNet has the potential to facilitate the use of ANNs in a data mining context, where previously their use has been limited because of the large amounts of data.

This paper describes the design and implementation of the PEPNet algorithm. This paper is structured as follows. Section 2 describes the basic EPNet algorithm, discussing its main components. Section 3 describes the parallel structure utilised by the PEPNet algorithm. Section 4 presents some preliminary results of PEPNet, identifying features of the alternative parallel architectures which have proven successful. Section 5 presents conclusions and further work.

2 The EPNet Algorithm

The EPNet algorithm has two primary constituents: a Multilayered Perceptron [5] and an Evolutionary Programming [6] subsystem. A Multilayered Perceptron encoding of an ANN facilitates both the use of Evolutionary Programming operators and the representation of multiple layered ANNs. We first present an overview of the EPNet algorithm and then discuss the encoding and evolutionary operators in detail.

2.1 Overview

Figure 1 summarises the phases of the EPNet algorithm. We say that a *run* of the algorithm *evolves* a specific number of *generations*. Each generation modifies a single ANN from the population of ANNs. There are five basic stages in the algorithm:

1. **Population Creation** initialises user parameters and creates an initial population of ANNs. A supplied dataset is split into three subsets: a training set; a validation set; and a test set. Training and validation sets interchange members and both sets are seen by the ANNs during training. The test set remains unseen and is used for evaluation purposes only.

2. **Selection** ranks each ANN based on test set performance, providing a measure of fitness. Each ANN is assigned a probability based on its rank. ANNs performing badly are favoured over those already performing well, thereby improving the entire population rather than a single ANN.

3. **Mutation** is the training phase of the EPNet algorithm and employs Evolutionary Programming techniques. Training of ANNs in EPNet involves both modifying the weights on connections between nodes and removing or adding connections between nodes (modifying the structure of the network). The training is effected by the use of six evolutionary operators which are ordered to favour compact, efficient ANNs (so that deletions are considered before additions).

4. **Replacement** replaces an individual in the population with a (improved) newly trained ANN. In performing a replacement either the parent or the least fit individual is replaced. If connection weight training produces an improved ANN, parental replacement is performed, maintaining a behavioural link between generations. If structural training is successful then the

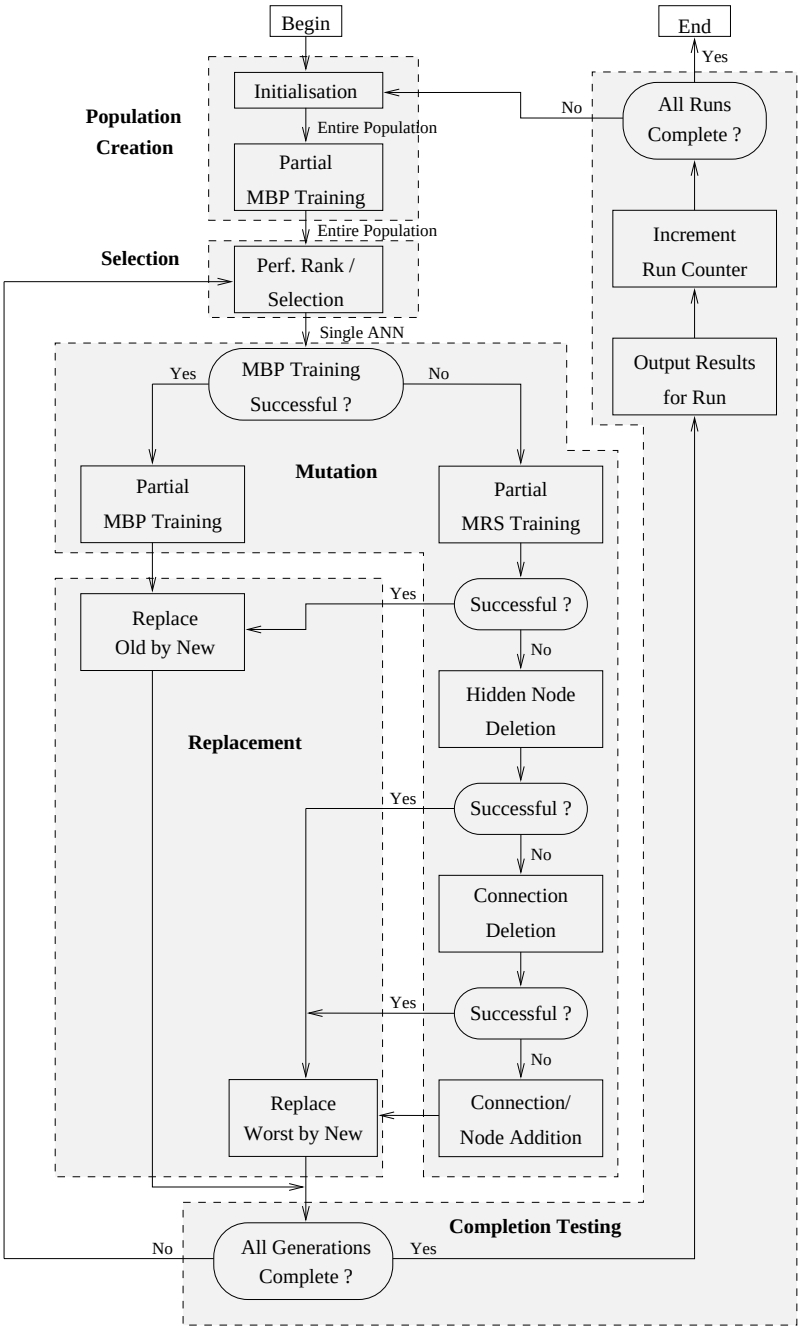

Figure 1: EPNet Algorithm.

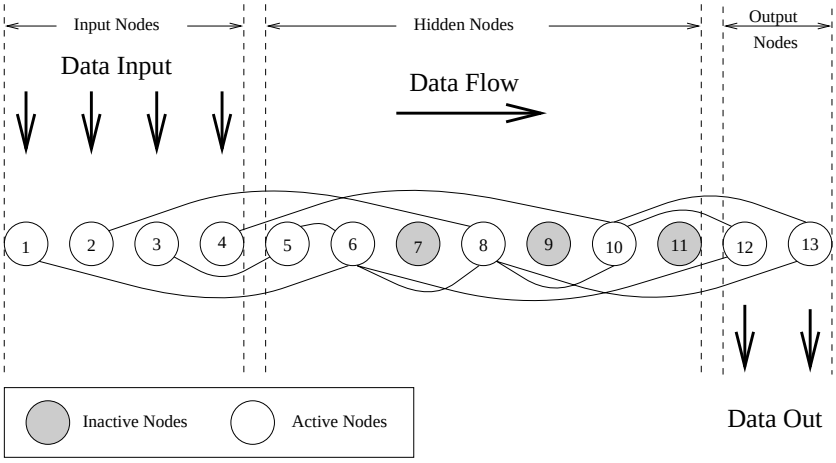

Figure 2: Generalised multilayer perceptron.

least fit individual of the population is replaced as the behavioural link between parent and offspring has been lost.

5. **Completion Testing** tests whether the required number of generations and runs has been completed.

2.2 Multilayer Perceptron

Figure 2 illustrates the features of a multilayer perceptron. It consists of a single row of nodes, each having an activation function. A node takes input via connections from nodes to its left and computes an output value which it transmits to its right. Connections between nodes have weights which are adjusted during weight training.

The concept of active and inactive nodes is used to facilitate structural modifications. Input and hidden nodes can be active or inactive (there is little point in output nodes being inactive). Inactive nodes do not receive input nor produce output, effectively removing the node from the ANN.

2.3 Evolutionary Operators

EPNet performs architectural modifications through evolutionary operators. A total of six operators are defined: two connection weight trainers and four architectural modifiers. An operator is considered to have been successful if performance improvement has been made to the ANN being trained, otherwise the operator has failed.

Modified Back Propagation is a connection weight operator. Back propagation (BP) being a gradient descent search technique is prone to finding local optima. BP trains ANNs by *back propagating* the difference between the actual output and the desired output through the ANN. Connection weights are either enhanced if providing a positive influence on the output or decreased if a negative influence is being supplied. Modified BP uses adaptive learning rates.

Modified Random Search (MRS) is a connection weight modifier used as an alternative to the MBP operator. The MRS operator is based on simulated annealing as described by Solis and Wets [7]. Training involves a number of iterations, where each iteration modifies a connection

weight and tests for an improvement in the overall performance of the ANN. The technique allows for making a bad modification, i.e., one that decreases performance, in the hope of finding further modifications which improve the ANN.

Hidden Node Deletion is an architectural operator compressing the ANN to maintain or improve performance. Hidden node deletion sets an active node as being inactive thereby removing it from the ANN. The node is chosen at random and all connections in and out are removed. This implies that data flowing through the ANN will be redirected around the deleted node.

Connection Removal is an architectural operator randomly removing chosen connections between active nodes. The choice of connections is based on a probability distribution of the importance of individual connections. EPNet removes connections which have an unimportant influence on the output of the ANN, aiming to remove redundant connections.

Connection Addition is an architectural operator adding a number of connections between previously unconnected nodes. A random number of connections are made between randomly chosen unconnected node pairs.

Hidden Node Addition is an architectural operator allowing EPNet to add a single hidden node to the ANN. The operator uses a method of node splitting to add new nodes. Node splitting is similar to biological cell division, and is implemented as follows. Suppose node i is split to create an extra node k. The new node k is the first inactive node to i's right (that is, $k > i$). To activate k, input and output connections are established. Node k is assigned the same input and output connections as i. The output connections are assigned the same weights. The input connection weights are calculated: for $k, w'_{jk} = -\alpha w_{ji}, j < i$; for $i, w'_{ji} = (1 + \alpha)w_{ji}, j < i$. Here α is a mutation parameter ranging from 0 to 1. Thereby EPNet is able to reinforce an existing node with a new node. The influence of a node is not considered so it is possible for this operator to reinforce a node which has an overall detrimental effect. However, the extra genetic material added by the new node might well lead to an improvement in performance later in the training.

3 Parallelising EPNet

Riessen [8] describes a number of ways of parallelising the EPNet algorithm. We will only discuss two of these parallel structures. These parallelisms are based on two features of the EPNet algorithm: population parallelism and individual parallelism.

Population parallelism takes advantage of the independence of each member of the ANN population to train a number of individuals concurrently. Individual parallelism takes advantage of the independence of the nodes within an ANN. The connection weights between several node-pairs can be altered simultaneously.

The intention is for a general-purpose, parallel implementation, not tied to any particular parallel architecture. The general model is of a collection of processors each able to communicate with all other processors. A MIMD (Multiple Instruction Multiple Data) architecture was chosen over alternative architectures as it has the advantage of not requiring processors to be in lock step. The disadvantage though is that careful consideration needs to be given to the computation/communication ratio—if communications outweighs computation there is no benefit.

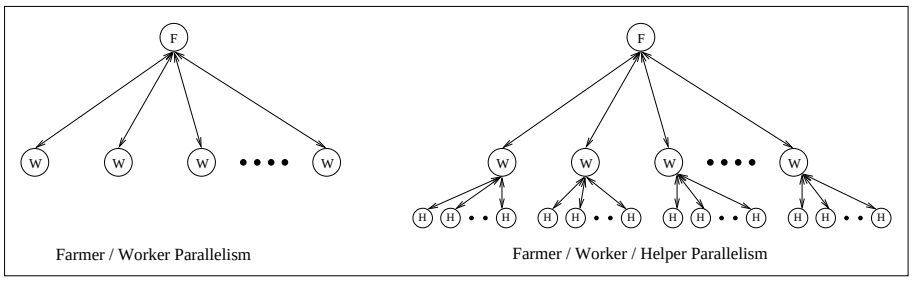

Figure 3: PEPNet Parallel Architectures.

PEPNet allows one of four alternative levels of parallelism to be chosen, ranging from none (effectively a serial implementation), to extreme parallelism (communication swaps computation). We will only discuss the two main parallelisms, and compare these to each other and a serial implementation.

Figure 3 shows the two primary parallel architectures of PEPNet: the so called farmer/worker and farmer/worker/helper architectures. Each node in Figure 3 represents a separate process which is either a farmer F, a worker W, or a helper H.

A *farmer* is responsible both for the generation of a new population of ANNs for each run and for the division of that population amongst the workers. The farmer must also ensure that the global population remain consistent. Each *worker* is assigned a sub-population of ANNs, and performs the EPNet algorithm on that population. *Helpers* add an extra level of individual parallelism by aiding workers in performing specific tasks.

Both parallel architectures of PEPNet use population parallelism. The farmer/worker/helper architecture also takes advantage of individual parallelism with the use of helper processors.

3.1 Farmer/Worker Architecture

Having generated a global population, the farmer transmits to the workers a sub-population. Each worker controls the generations of a population for a run, ensuring that the Selection, Mutation, Replacement, and Completion Testing[2] are performed on its sub-population.

On completion of all the required generations for a run, workers send back their sub-populations. The run completes when all evolved sub-populations have been received by the farmer.

Communication costs are not a concern, as communication is concentrated at the beginning and completion of a run. The cost of communicating populations relative to performing several generations is minimal. Workers execute independently of one another and there is no communication between workers.

Farmer/worker parallelism takes advantage of population parallelism, with workers performing the EPNet algorithm locally, not utilising individual parallelism. On the other hand, the farmer/worker/farmer parallelism takes advantage of both population and individual parallelism.

3.2 Farmer/Worker/Helper Architecture

The farmer/worker/helper architecture, utilising both population and individual parallelism, has a farmer and worker performing similar tasks to those under the farmer/worker architecture. The workers are assigned a group of helpers.

[2]Only the Generation Completion Test.

Helpers aid in two tasks: the initial MBP training of the entire population, and any modified random search training performed on any individual. The latter of these represents the individual parallelism employed by the farmer/worker/helper architecture. In this form of parallelism the helpers work cooperatively in reaching a common goal. MBP training requires the helpers to work independently, without communicating amongst themselves.

In aiding the worker with MBP training, each helper is assigned a single ANN from the workers sub-population. On this individual the helper performs the MBP training algorithm. Upon completion, the helper communicates the ANN to the worker. The worker continues the EPNet algorithm upon receiving all ANNs.

The helpers are also called upon by the MRS operator, working cooperatively and communicating partial results. Each helper is assigned a region of the ANN on which it is the only helper permitted to make modifications. The worker performs load balancing by identifying the number of active nodes within the ANN, allocating a similar number of active nodes to each helper.

Before helpers make permanent modification to the ANN, they check for alterations made by other helpers. If none have been made, a helper is able to make its modification permanent, otherwise the modification is discarded.

Communication among the helpers is via the worker, reducing the amount of communication necessary and also allowing the worker to maintain a consistent copy of the ANN.

The overall communications are greater than in the farmer/worker architecture and therefore it is necessary to offset this with the computational costs. To judge this architecture as being an improvement on the farmer/worker architecture, it is important to show that the extra communication has improved performance of the PEPNet algorithm.

3.3 Implementation

PEPNet is implemented on a Fujitsu MIMD AP1000 with 128 processors. The MPI [9] message passing interface standard for implementing parallel systems, has been used, allowing PEPNet to run on a variety of platforms ranging from networked workstations to super computers.

4 Results

Experimental results have been obtained for running PEPNet over many different populations. The illustrative results presented here were derived from the Australian Credit Dataset. The dataset consists of a total of 690 records of which 307 (44.5%) were positive (class attribute equal to 1) and 383 (55.5%) were negative (class attribute equal to 0). For each experiment, 346 records were used as training records, 172 as test records and 172 as validation records. Three relevant measures are recorded for the results:

- *Rate* measures the accuracy of prediction and is the ratio of incorrectly predicted records to total predicted records. A correctly predicted record is within a specific distance of the desired output for the record.
- *Error* measures the distance of the PEPNet prediction from the actual output value.
- *Accumulated Execution Time* records the time taken by PEPNet to perform a set number of generations.

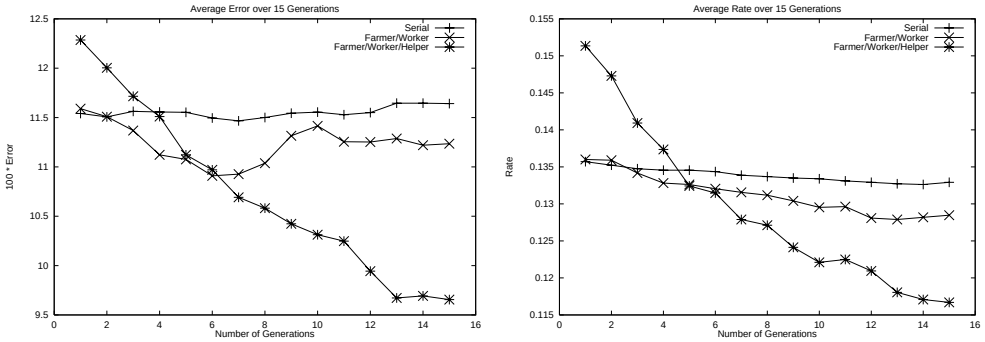

Figure 4: Average Error and Rate Plots.

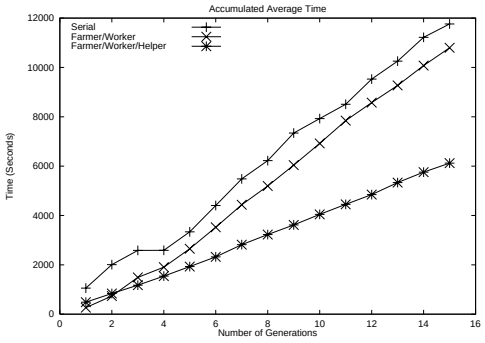

Figure 5: Average Accumulated Time Plot.

Figures 4 and 5 show the results of performing one run of 15 generations. For this experiment four workers are assigned to farmer. For the farmer/worker/helper architecture each worker has five helpers. The global population comprised 20 individuals, with each worker assigned a sub-populations of five ANNs.

For each generation the serial architecture trains a single ANN whilst both parallel architectures train a total of four ANNs per generation. As a result, both parallel architectures train 60 ANNs compared to the serial architectures 15 ANNs.

The figures reflect this difference in the rate and error plots, with both parallel architectures producing fitter populations after the 15 generations. Figure 5 shows that the overall time taken actually decreases compared to the serial implementation.

The experimental results show a trend where the farmer/worker/helper architecture outperforms the farmer/worker architecture. The use of extra helpers provides a definite improvement, outweighing the extra communications cost.

We can justification this improvement by considering the MRS operator. The farmer/worker/helper architecture is able to make modifications on later nodes sooner than the farmer/worker architecture. This results in a more diverse series of modifications, hence more diversity in the genetic material, and therefore improving performance.

The experimental results also show that helpers provide a time performance benefit (as seen in the accumulated time plots and more clearly in the time performance experiments). This suggests that the extra communications involved when using helpers is out-weighed by the time saved through their use. The farmer/worker/helper completes all 15 generations in under 2 hours whilst the alternative architectures failed to do so.

5 Summary and Conclusions

We have presented a description of a Parallel Evolutionary Artificial Neural Network data mining tool. The PEPNet algorithm exploits inherent parallelisms within the EPNet algorithm to achieve a improved prediction and time performance.

The architectures implemented in PEPNet use a single controlling farmer to distribute a globally generated population of ANNs over a group of worker nodes. A refinement of this architecture assigns to each worker node a group of helper nodes, one for each individual in the population.

The experimental studies (Figures 4 and 5, representing just one set of results) indicate that the use of parallel architectures for EANNs is feasible and efficient. The farmer/worker/helper architecture (making more use of available processing power) appears to be a better option in terms of accuracy and time performance. The extra overhead introduced by helpers is offset by their computational benefits. The helpers were particularly useful in supporting modified random search, the most time consuming component of EPNet.

Further development of PEPNet is required to turn it into a practical data mining tool. In particular, PEPNet will need to take advantage of all available processing power regardless of population size. The current implementation can only take advantage of a large number of processors if a large population of ANNs is chosen. The original EPNet algorithm can also be modified so that instead of using the best individual of the final population for classification the entire population is used (Yao and Liu [2] showed that the knowledge of the final population as a whole is often greater than the knowledge of a single ANN).

6 Acknowledgements

The authors acknowledge the support provided by the Cooperative Research Centre for Advanced Computational Systems (ACSys) established under the Australian Government's Cooperative Research Centres Program.

The authors also wish to thank Zhexue Huang and Peter Milne for their comments and insightful ideas.

References

[1] Yao, X.: 1995, Evolutionary artificial neural networks, *in* A. Kent and J. G. Williams (eds), *Encyclopedia of Computer Science and Technology*, Vol. 33, Marcel Dekker Inc., New York, NY 10016, pp. 137–170.

[2] Yao, X. and Liu, Y.: 1996a, Ensemble Structure of Evolutionary Artificial Neural Networks, *Third IEEE International Conference on Evolutionary Computation (ICEC'96)*, Nagoya, Japan, pp. 659–664.

[3] Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P.: 1996, From data mining to knowledge discovery: An overview, *in* U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds), *Advances in Knowledge Discovery and Data Mining*, AAAI Press.

[4] Williams, G. J. and Huang, Z.: 1996, A case study in knowledge acquisition for insurance risk assessment using a kdd methodology, *in* P. Compton, R. Mizoguchi, H. Motoda and T. Menzies (eds), *Proceedings of the Pacific Knowledge Acquisition Workshop*, Sydney, Australia, pp. 117–129.

[5] Liu, Y. and Yao, X.: 1996, A population-based learning algorithm which learns both architectures and weights of neural networks, *Chinese Journal of Advanced Software Research (Allerton Press, Inc., New York, NY 10011)* **3**(1), 54–65.

[6] Yao, X. and Liu, Y.: 1996b, Making use of population information in evolutionary artificial neural networks, *IEEE Trans. on Systems, Man, and Cybernetics.* Accepted.

[7] Solis, F. and Wets, R.-B.: 1981, Minimization by random search techniques, *Mathematics of Operation Research* **6**(1), 19–30.

[8] Riessen, G. A.: 1996, Parallel evolutionary artificial neural networks, *Honours Thesis*, Dept of Computer Science, Australian National University.

[9] Message Passing Interface Forum: 1995, *MPI: A Message-Passing Interface Standard*, version 1.1 edn, ARPA and NSF and Esprit.