

---

# On the Design of an Evolutionary Preprocessor

---

S. Kazadi, D. Choi, A. Chang, T. Kang, H. Li, D. Kim, S. Ho, J. Wu  
Jisan Research Institute  
28 North Oak Avenue  
Pasadena, CA 91107  
USA

## Abstract

In this paper we explore methods of enhancing the evolvability of a particular device. We assume that the device may be specified by a table of inputs and outputs. We investigate a method of extracting the topological structure of the device from *rarified absolute Hessian* matrices (raH matrices) and using this topological information as the basis for construction of solutions to evolutionary problems. We validate the algorithm by demonstrating its ability to extract the structure of devices to be evolved from the input/output table. Moreover, we validate this structure by using a genetic algorithm to train a perceptron, yielding perceptrons which solve the computational problem with error rates of less than 4%.

## 1 Introduction

Perhaps the most important technical hurdle for the development of evolutionary algorithms capable of handling complex evolutionary problems is the development of methods of handling multipart devices in evolutionary systems [Hounsell, Hornby, Munetomo, Kazadi, 3]. While many researchers have explored the use of groupings in the development of evolutionary algorithms, there seems to be no specific algorithm that has yet been devised that is capable of extending the frontier of what is evolvable beyond single-function systems or relatively non-integrated multifunction systems.

This problem is so pervasive, that it has received extensive attention in recent years. It has been the subject of multiple studies starting with genetic algorithm studies, continuing with genetic programming studies,

and now appearing in evolutionary systems studies. Yet despite the great effort on this problem, few studies have demonstrated complex evolutionary capabilities significantly beyond those of the early 90's. The problem may be understood, in part, because of the way in which evolutionary systems develop. Theoretically, it has been shown that increasing complexity of a system increases the computation time factorially. This extraordinary increase in computation time is only partially alleviated by phenomena such as *evolutionary acceleration* [Kazadi, 4] and *partial compartmentalization*, which is the method most researchers have used to deal with this problem.

The difficulty in the design can be alleviated by the creation of *compartmental models*. These are models under which evolution occurs which specify the groups of inputs and outputs of a device, along with interconnection structure between such devices. These can be taken as an initial step in the evolutionary system, and can help direct the evolutionary model in use. As an example, suppose that a particular genetic algorithm is using the fitness function

$$f_1 = g(x, y), f_2 = h(w, z) \quad (1)$$

The compartmental model might consist of two discrete blocks consisting of single outputs fed by two inputs each; the first output would be fed by inputs  $x$  and  $y$ , while the second output is fed by inputs  $w$  and  $z$ .

In this paper, we describe a partial solution to this problem for some evolutionary design problems. In the case that the device to be evolved can be specified by an input-output table, topological information can be extracted in fourth order computational time, which can be used to specify the overall organizational structure of the device to be evolved. This organizational structure, which is the missing link in the evolution of high complexity devices, can then be used as a scaffolding for the overall evolutionary process.

The rest of the paper is organized as follows. In Section 2, we examine the theoretical basis of the preprocessor. Section 3 discusses the preprocessor and its function on various design problems involving GA-trained neural network models, comparing the efficacy and structure of the final solution with that of a fully connected structure. Finally, Section 4 offers some concluding remarks.

## 2 Theoretical Framework of the Preprocessor

The evolutionary preprocessor finds its roots in the work of Kazadi [Kazadi, 1, Kazadi, 2]. In that work, it was demonstrated that evolutionary algorithms could be reencoded using the eigenvector basis of the absolute Hessian matrix, generating a new encoding of the genetic algorithm problem that maximally divided the search space into direct product spaces. The absolute Hessian matrix,

$$H = [|\partial_{ij}f|]_{i,j=1}^N \quad (2)$$

is related to the Hessian matrix; all elements of the matrix are absolute-valued. It has been demonstrated that this basis most efficiently breaks the search space into product spaces made up of functionally independent subspaces.

The previous study was specifically geared to the evolution of solutions in genetic algorithms. However, the result can be extended to include open-ended evolutionary problems, closed-space evolutionary problems, and multi-function evolutionary problems. Suppose, first that the device whose evolution is desired is specified by an input-output table

$$D = \left\{ \left( \vec{I}_i, \vec{O}_i \right) \mid 0 < i \leq k \right\} \quad (3)$$

where

$$\vec{I} = (I_1, \dots, I_n) \quad (4)$$

is an input vector and

$$\vec{O} = (O_1, \dots, O_m) \quad (5)$$

is an output vector.  $n$  need not equal  $m$  in any of what follows. Evaluation of the completeness and correctness of the design is typically a direct evaluation of whether or not the output of the designed structure matches the expected outputs of the specification table. I.e.

$$f(S) = \sum_{i=1}^m \sum_{j=1}^k \left| S_i(\vec{I}_j) - \vec{O} \right|. \quad (6)$$

As this is a sum, we can regard each of the outputs as a separate entity, and analyze the structure of each of these elements. As indicated in [Kazadi, 1], the absolute Hessian matrix will be made up of components which can be grouped in blocks along the diagonal. This may be represented as

$$aH = \begin{pmatrix} B_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & B_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & B_3 & 0 & \dots & 0 \\ 0 & 0 & 0 & B_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & B_n \end{pmatrix} \quad (7)$$

where each block represents an individual set of related components. Previous work indicated that genetic algorithms would be most effective if crossovers were limited, in large part, or in later stages, to these blocks, so as to avoid disruption of data found therein.

In practice, not all inputs are connected to all outputs. That is, it is possible to have a device that has several inputs and outputs that are not connected to one another. In this case, the absolute Hessian matrix has several zeros in the place of the diagonal blocks given in equation (6). Moreover, linear dependencies are obscured by the double derivative on the diagonals. This can be corrected by instead using

$$A_{ij} = \partial_{ij}f + \delta_{ij}\partial_i f \quad (8)$$

in the place of the components of the matrix, where the  $\delta_{ij}$  refers to the Kronecker-Delta function. The new absolute Hessian matrix is a so-called *rarified altered Hessian* (raH) matrix. Such a matrix might, with some rearrangement, look like

$$raH = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & B_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & B_n \end{pmatrix}. \quad (9)$$

In this case, it is generally the case that the first three elements can be neglected from this particular output's considerations. This produces the device design given in Figure 1, in which several inputs are not connected to the given output.

Each output may be examined according to its raH matrix. This matrix contains all of the design specific information required to construct input/output groupings for the specific given inputs.

It is possible for the input/output groupings to over-

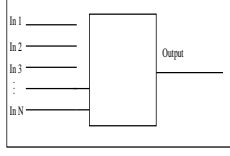


Figure 1: This figure illustrates the device specification which results from the rarified absolute Hessian matrix given in equation (7). This device, as with the raH matrix, does not have any inputs from the first three input lines, and so may be examined in that light.

lap. Consider the four-input device with three outputs.

$$O_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (10)$$

$$O_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (11)$$

$$O_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

Clearly, the first and second outputs contain contributions from the second input, but contain independent contributions from the first and third inputs, respectively. Thus, if one were to create functional groupings, it might be incorrect to group them according to the inputs one and two, and then two and three. Indeed, some of the designs one might imagine building up for output one might be useful for output two.

One useful method of determining the absolute groupings is to calculate the product of any two given raH matrices for different outputs. In our case, this will lead to

$$\frac{1}{2} (O_1 O_2 + O_2 O_1) = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (13)$$

$$\frac{1}{2} (O_1 O_3 + O_3 O_1) = \frac{1}{2} (O_2 O_3 + O_3 O_2) = 0 \quad (14)$$

Thus, it is clearly the case that a reasonable grouping would be to have inputs one, two, and three go to the same group as that containing outputs one and two. Moreover, the third output cannot be grouped with any output, meaning that the fourth input should uniquely determine the third output. The situation is as depicted in Figure 2.

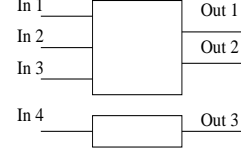


Figure 2: This figure illustrates the combination of the groups of devices that were given in the equations (8) and (9). The two groups containing inputs one and two and inputs two and three, respectively, have been merged into one group which contains all three inputs and two outputs.

The device so specified can then be built without allowing mutations which cause any interactions between the members of the different sub-devices. The new larger groups should be iteratively merged with other groups until no further grouping can be discerned. Thus, the raH matrices for the various inputs may be used with simple matrix operations to create accurate groupings of multipart devices.

We may view the specification table as a vector fitness function. As in all fitness functions, different parts of the vector space can cause different measurements of the raH matrix; that is, the raH matrix is dependant on the search space location. However, the raH measurements may be approximated by

$$|\partial_{ij} O_k| \simeq |O_k(\vec{x}) + O_k(\vec{x}_{ij}^{\rightarrow}) - O_k(\vec{x}_i^{\rightarrow}) - O_k(\vec{x}_j^{\rightarrow})| \quad (15)$$

where  $\vec{x}_{ij}^{\rightarrow}$  represents the vector  $\vec{x}$  with the *i*th and *j*th vector components altered.  $\vec{x}_i^{\rightarrow}$  and  $\vec{x}_j^{\rightarrow}$  are analogously defined. This value can be calculated for any four vectors differing only by these specific components.

As an example, consider the two-bit binary adder. The specification table is as follows:

| $I_1$ | $I_2$ | $I_3$ | $I_4$ | $O_1$ | $O_2$ | $O_3$ |
|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 0     | 0     | 0     | 1     | 0     | 0     | 1     |
| 0     | 0     | 1     | 0     | 0     | 1     | 0     |
| 0     | 0     | 1     | 1     | 0     | 1     | 1     |
| 0     | 1     | 0     | 0     | 0     | 0     | 1     |
| 0     | 1     | 0     | 1     | 0     | 1     | 0     |
| 0     | 1     | 1     | 0     | 0     | 1     | 1     |
| 0     | 1     | 1     | 1     | 1     | 0     | 0     |
| 1     | 0     | 0     | 0     | 0     | 1     | 0     |
| 1     | 0     | 0     | 1     | 0     | 1     | 1     |
| 1     | 0     | 1     | 0     | 1     | 0     | 0     |
| 1     | 0     | 1     | 1     | 1     | 0     | 1     |
| 1     | 1     | 0     | 0     | 0     | 1     | 1     |
| 1     | 1     | 0     | 1     | 1     | 0     | 0     |
| 1     | 1     | 1     | 0     | 1     | 0     | 1     |
| 1     | 1     | 1     | 1     | 1     | 1     | 0     |

Table 2.1: The two-bit adder table.

The output matrices for this table are given by

$$O_1 = \begin{pmatrix} 4 & 2 & 4 & 2 \\ 2 & 2 & 2 & 2 \\ 4 & 2 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix} \quad (16)$$

$$O_2 = \begin{pmatrix} 8 & 4 & 8 & 4 \\ 4 & 4 & 4 & 4 \\ 8 & 4 & 8 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix} \quad (17)$$

$$O_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 8 \end{pmatrix} \quad (18)$$

These matrices have a few interesting properties. First, the components of  $O_1$  are significantly smaller than those of  $O_2$  or  $O_3$ . This indicates that  $O_1$  is very much less strongly affected by the inputs. Secondly, the largest components appear in  $O_2$  and  $O_3$  at components 13 and 24, respectively. Dividing through by the largest number, 8, we obtain

$$O_1 = \begin{pmatrix} 0.5 & 0.25 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0.25 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \quad (19)$$

$$O_2 = \begin{pmatrix} 1 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix} \quad (20)$$

$$O_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad (21)$$

If we were to use a threshold, setting any element to zero if it is smaller than one, the matrices become

$$O_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (22)$$

$$O_2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (23)$$

$$O_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad (24)$$

Using these matrices, we find that

$$O_1 O_2 = O_1 O_3 = O_2 O_3 = 0 \quad (25)$$

That each of the products is zero can be taken as an indication that none of the outputs come from groups with overlapping inputs. Since  $O_2$  has nonzero elements in positions 13 and 31, it can be inferred that inputs 1 and 3 are associated with output 2. Similar reasoning gives a similar outcome for output 3 and inputs 2 and 4. Output 1 seems to have no direct inputs.

Indirect inputs may be inferred from the products of the matrices. Dividing through each matrix by the maximum value in each of the matrices and performing the following operations, we find

$$\frac{O_1 O_2 + O_2 O_1}{2} = \begin{pmatrix} 1.25 & 0.75 & 1.25 & 0.75 \\ 0.75 & 0.50 & 0.75 & 0.50 \\ 1.25 & 0.75 & 1.25 & 0.75 \\ 0.75 & 0.50 & 0.75 & 0.50 \end{pmatrix} \quad (26)$$

$$\frac{O_1 O_3 + O_3 O_1}{2} = \begin{pmatrix} 0 & 0.25 & 0 & 0.25 \\ 0.25 & 0.50 & 0.25 & 0.50 \\ 0 & 0.25 & 0 & 0.25 \\ 0.25 & 0.50 & 0.25 & 0.50 \end{pmatrix} \quad (27)$$

$$\frac{O_3 O_2 + O_2 O_3}{2} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 1.0 & 0.5 & 1.0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 1.0 & 0.5 & 1.0 \end{pmatrix} \quad (28)$$

It is interesting to note that

$$\left| \frac{O_1 O_2 + O_2 O_1}{2} \right| = 3.43 \quad (29)$$

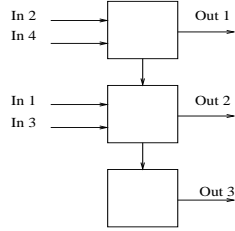


Figure 3: This figure gives the structure of the two-bit adder extracted from the specification chart using conjugate schema-based matrix analysis.

$$\left| \frac{O_3 O_2 + O_2 O_3}{2} \right| = 2.41 \quad (30)$$

$$\left| \frac{O_3 O_1 + O_1 O_3}{2} \right| = 1.21 \quad (31)$$

This can be interpreted as requiring indirect connections from  $O_3$  to  $O_2$ , from  $O_2$  to  $O_1$ , and from  $O_3$  to  $O_1$ . However, applying a threshold of 2.0 to this may allow us to remove the latter indirect connection, yielding the following structure chart.

This structure may be recognized as the standard two-bit adder without look-ahead, as shown in Figure 3.

The methods described above are very dependant on the thresholds that were chosen. No specific method has yet been identified as a reasonable method of choosing these thresholds.

### 3 Evolutionary preprocessor and evolutionary design

In order to explore the efficacy of this preprocessor, we apply it to a complete evolutionary design phase involving neural networks and a genetic algorithm. The neural network employed is a perceptron with firing function

$$f(x) = x. \quad (32)$$

Our training data consists of input/output pairs of data with multiple inputs and outputs. The network is trained using a real-encoded genetic algorithm [Kazadi, 1]. Each element of the population is a vector containing all of the connection weights in the neural network. Each GA has a population size of 100 individuals, a mutation probability of 1.00 using single weight mutation, a crossover probability of 0.8, and elitist round-robin reproduction. Each GA is run for a maximum of 10000 iterations, taking a total computational time of less than 24 hours on a Pentium II class computer with clockspeed 450 MHz. Multiple machines were employed in this study, ranging from

Pentium class computers with clockspeeds of 150 MHz to Pentium II class computers, as previously described.

Our evaluation methodology is twofold. First, we report on the ability of the preprocessor to extract the correct compartmental model from the training data. Second, we report the ability of the evolved network to reproduce the computation given in the problem specifications given below. Once the compartmental model has been extracted, a three-layer perceptron is assembled consisting of individual nodes for each input and output, and intermediate nodes of the same multiplicity as the initial nodes. As the training process completes, the final population's best network is reported and tested against the original data. Moreover, its performance on the training problem is compared to that of the original function.

We choose ten different compartmental models to apply our preprocessor to. These may be summarized using the following table of functions.

| Network: | Formula  |
|----------|--|
| Net1     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} ax_1 \\ bx_2 \\ cx_3 \\ dx_4 \\ ex_5 \end{pmatrix}$   |
| Net2     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} (w_1 w_{10} + w_2 w_{12}) x_1 + \\ (w_3 w_{10} + w_4 w_{12}) x_2 + \\ (w_1 w_{11} + w_2 w_{13}) x_1 + \\ (w_3 w_{11} + w_4 w_{13}) x_2 + \\ (w_5 w_{14} + w_6 w_{16}) x_3 + \\ (w_7 w_{14} + w_8 w_{16}) x_4 + \\ (w_5 w_{15} + w_6 w_{17}) x_3 + \\ (w_7 w_{15} + w_8 w_{17}) x_4 \\ ax_5 \end{pmatrix}$ |
| Net3     | $f \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_3 w_1 x_1 \\ w_4 w_1 x_1 \\ w_5 w_2 x_2 \\ w_6 w_2 x_2 \end{pmatrix}$  |
| Net4     | $f \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} (w_1 w_5 + w_2 w_6) x_1 + \\ (w_3 w_5 + w_4 w_6) x_2 \end{pmatrix}$  |
| Net5     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} (w_1 w_4 + w_2 w_{12} + w_3 w_{13}) x_1 + \\ (w_2 w_{11} + w_5 w_{12} + w_6 w_{13}) x_2 + \\ (w_7 w_{11} + w_8 w_{12} + w_9 w_{13}) x_3 + \\ ax_4 \end{pmatrix}$   |
| Net6     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} ax_1 \\ (w_2 w_7 + w_3 w_{10}) x_2 + \\ (w_4 w_7 + w_5 w_{10}) x_3 + \\ (w_3 w_{11} + w_2 w_8) x_2 + \\ (w_5 w_{11} + w_4 w_8) x_3 + \\ (w_2 w_9 + w_3 w_{12}) x_2 + \\ (w_4 w_9 + w_5 w_{12}) x_3 \end{pmatrix}$   |
| Net7     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} w_3 w_6 x_3 \\ w_1 w_4 x_1 + w_2 w_5 x_2 \end{pmatrix}$   |
| Net8     | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} ax_3 \\ bx_3 \\ (w_1 w_6 + w_2 w_8) x_1 + \\ (w_3 w_6 + w_4 w_8) x_2 + \\ (w_1 w_7 + w_7 w_9) x_1 + \\ (w_3 w_7 + w_4 w_9) x_2 \end{pmatrix}$   |
| Net9     | $f(x_1) = \begin{pmatrix} w_1 w_2 x_1 \\ w_1 w_3 x_1 \\ w_1 w_4 x_1 \end{pmatrix}$   |
| Net10    | $f \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} ax_1 \\ bx_2 \\ cx_2 \\ dx_3 \end{pmatrix}$   |

Table 3.1: The various functions are defined by specifying the weights  $w_i$  in each function, and creating a random set of input vectors

The functions given in Table 3.1 each have different compartmental models. For instance, Net 1 has five different compartments with independent inputs and outputs. On the other hand, Net 2 has only three compartments, with two compartments being fed by two different inputs and determining two different outputs, and one compartment having an independent input and output. In each case, the compartmental model is determined by the inputs leading to specific outputs, and the chain of dependencies between the outputs and inputs.

Our first step in the evolutionary process is the extraction of the compartmental model of the device. This is accomplished by scanning input-output data files for the compartmental structure of the device. In

our experiments, we generate these data files, by generating semi-random inputs to each of the structures given in Table 3.1 and recording the inputs and outputs. The data is generated in groups of four. Initially, a random vector is generated using a uniform random number generator. Then, two random numbers are generated using the same random number generator. Three new vectors are generated by substituting the new random numbers in for elements of the first vector at two points individually, and then together. As an example, we might have the vectors (1.1, 2.2, 3.3, 4.4), (1.7, 2.2, 3.3, 4.4), (1.1, 2.2, 8.5, 4.4), (1.7, 2.2, 8.5, 4.4). Sets of four vectors are necessary for the numerical determination of the raH matrices. One thousand vectors are generated by this method and the raH matrices are approximated from these thousand vectors. In Table 3.2 we summarize the outcome of these experiments.

| Network: | Compartments: | Model   |
|----------|---------------|---|
| Net1     | 5             | $\begin{pmatrix} I_1 \mapsto O_1 \\ I_2 \mapsto O_2 \\ I_3 \mapsto O_3 \\ I_4 \mapsto O_4 \\ I_5 \mapsto O_5 \end{pmatrix}$ |
| Net2     | 3             | $\begin{pmatrix} (I_1, I_2) \mapsto (O_1, O_2) \\ (I_3, I_4) \mapsto (O_3, O_4) \\ I_5 \mapsto O_5 \end{pmatrix}$           |
| Net3     | 2             | $\begin{pmatrix} I_1 \mapsto (O_1, O_2) \\ I_2 \mapsto (O_3, O_4) \end{pmatrix}$  |
| Net4     | 1             | $((I_1, I_2) \mapsto O_1)$  |
| Net5     | 4             | $\begin{pmatrix} I_1 \mapsto O_1 \\ I_2 \mapsto O_2 \\ I_3 \mapsto O_3 \\ I_4 \mapsto O_4 \end{pmatrix}$                    |
| Net6     | 2             | $\begin{pmatrix} I_1 \mapsto O_1 \\ (I_1, I_2) \mapsto (O_2, O_3, O_4) \end{pmatrix}$                                       |
| Net7     | 2             | $\begin{pmatrix} I_3 \mapsto O_1 \\ (I_1, I_2) \mapsto O_2 \end{pmatrix}$   |
| Net8     | 2             | $\begin{pmatrix} I_3 \mapsto (O_1, O_2) \\ (I_1, I_2) \mapsto (O_3, O_4) \end{pmatrix}$                                     |
| Net9     | 1             | $(I_1 \mapsto (O_1, O_2, O_3))$   |
| Net10    | 3             | $\begin{pmatrix} I_1 \mapsto O_1 \\ I_2 \mapsto (O_2, O_3) \\ I_3 \mapsto O_4 \end{pmatrix}$                                |

Table 3.2: The number of compartments extracted from the data tables using the raH method of compartmental model analysis.

As expected the compartmental model is correctly extracted in every case.

We next use the compartmental model extracted from the training data as a template for the generation of neural network models. These models have single inputs for each of the input variables, with single outputs for each of the output variables. Each network has a single hidden layer. Connectivity is decided according

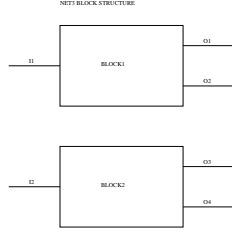


Figure 4: This is the compartmental model for the Net 3 training set. Each compartment has a single incoming input and two outputs.

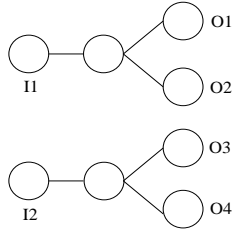


Figure 5: This gives the neural network model of the Net 3 compartmental neural network. The network consists of two compartments with connectivity between input one and outputs one and two, and input two and outputs three and four.

to the compartmental model used for the particular problem.

As an example consider the block structure for Net 3, shown in Figure 4. The compartmental model consists of two compartments, each with a single input and two outputs.

The network structure for this compartmental model is given in Figure 5. This model consists of two input nodes, two hidden nodes, and four output nodes. The connectivity of this model is consistent with the compartmental model. Notably, this means that input one is not connected (directly or indirectly) to outputs three and four, and input two is not connected (directly or indirectly) to outputs one and two.

The network’s connectivity is thus minimal in the sense that all extraneous connections have been eliminated due to the compartmental model extraction.

As indicated above, each perceptron is trained using a real-encoded genetic algorithm. This GA trains the weights of the network and produces a network which closely reproduces the functionality of the original functions. We train using one data set prepared as described above, and validate it using a second, independently generated data set. Because of limitations of our computational speed, we only train three of the

ten network types, with differing training data sets and various network sizes. In Table 3.3, we report the performance of the networks so trained in terms of their percent deviation from that of the functions themselves.

| Network Type | I/O   | % Error | Std. Dev. |
|--------------|-------|---------|-----------|
| Net1         | 5/5   | 1.30784 | 0.0068203 |
| Net1         | 10/10 | 3.2225  | 0.0261049 |
| Net2         | 5/5   | 0.254   | 0.0003    |
| Net3         | 2/4   | 2.823   | 0.01392   |
| Net3         | 4/8   | .694    | 0.00334   |

Table 3.3: This table gives the average and standard deviation of percentage error of the trained neural network outputs as compared to the outputs of the original function. In all cases, the average percentage error is below 10%.

The average error in each case is below 10%. This is reasonably good agreement in light of the simple genetic algorithm employed. As no effort was expended improving the efficacy of the GA - it was applied “out of the box” - we attribute this error rate to the lack of effort expended on the design of the genetic algorithm.

What we have then succeeded in doing is applying the general preprocessor-evolutionary stage method in the design of a blocked device. This device is designed in two stages. In the first, the preprocessor extracts the compartmental model from the specification table. In the second, the block diagram is applied to a particular instantiation of the device, and the device design is completed according to an appropriately chosen evolutionary algorithm. The entire process represents a complete circuit whose final step might be any appropriately chosen algorithm, with the design structure and evolutionary algorithm choice depending on the particular problem to be solved.

## 4 Concluding Remarks

This study has examined a method of performing the first step in an evolutionary design process. The first and perhaps most critical step in the development of a complex device such as complex electronic devices [Thompson] is the design of its high level structure. The necessity of this step has been demonstrated in [Kazadi, 3] in which it was demonstrated that the evolution of multi-part complex systems has an evolution time that increases factorially with increases in complexity. This factorial increase in computation time can be alleviated only by automatic or deliberate solution of the compartmentalization problem.

In this study, we've explored the design of just such a pre-evolutionary processor. We've established a method of extracting the group and group-based connection information from the specification table. Our tests of the system have included both the creation of compartmental models from the specification tables and the genetic algorithm training of neural network models of the compartmental models. In all cases, the compartmental models extracted using our method of design have been accurate in building the expected design, and the genetic algorithm-trained networks reproduced the original device's functionality within a deviation from the performance of the original device.

Despite these successes, this is but a tiny step in the direction of automated structure evolution. The next logical steps are two-fold, and parallel. As we have seen in Section 2, the development of methods of determining the indirectly connected structures and the directly connected structures is problematic at best, as there seems to be no theoretically motivated way to determine the threshold values relevant to this structure determination. This problem is part of a larger study in the development and application of device specifications for complex devices in evolutionary systems.

The other center of further studies is the development of more open-ended evolutionary models. In this study, we examined the efficacy of the method in the design of a genetic algorithm-based neural network training method. In order to be more useful, open-ended evolutionary algorithms must be used. In the absence of such algorithms, it is unlikely that devices of significant design complexity will be created outside of those devices of prearranged design. The use of these algorithms would seem to be necessary for the development of novel designs outside of human-mediated design.

## 5 Acknowledgements

This research was supported, in part, by a grant from the Center for Neuromorphic Engineering, an NSF Engineering Research Center, under grant number EEC-9402726, and by the Integrated Media Systems Center, an NSF Engineering Research Center at the University of Southern California, under grant number EEC-9529151. D. Choi was supported, in part, thanks to a generous grant from these Centers.

## References

[Hounsell] B. Hounsell and T. Arsian. *A Novel Evolvable Framework for the Evolution of High Performance Digi-*

*tal Circuits*. D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, L. Parmee, and H. Beyer, eds. **Proceedings of the Genetic and Evolutionary Computation Conference 2000**, Las Vegas, Nevada, USA, July 10-12, 2000.

[Hornby] G. Hornby, H. Lipson, and J. Pollack. *Evolution of Generative Design Systems for Modular Physical Robots*. **Proceedings of the IEEE International Conference on Robotics and Automation**, 2001.

[Kazadi, 1] S. Kazadi. *Conjugate schema in genetic evolution*. **Proceedings of the Seventh International Conference on Genetic Algorithms**. San Mateo, Ca: Morgan Kaufmann Publishers, pp. 10-17, 1997.

[Kazadi, 2] S. Kazadi. *Conjugate Schema and Basis Representation of Crossover and Mutation*. **Evolutionary Computation**. 6 (2), 129-160, 1998.

[Kazadi, 3] S. Kazadi, D. Lee, R. Modi, J. Sy, W. Lue. *Levels of Compartmentalization in Artificial Life*. **Proceedings of Artificial Life VII**, Bedau, McCaskill, Packard, and Rasmussen, eds.: MIT Press, 81-89, 2000.

[Kazadi, 4] S. Kazadi, S. Cheung, C. Ogletree, S. Kim, A. Min, C. Lee. *Evolutionary Acceleration*. In prep, 2003.

[Munetomo] M. Munetomo and D. Goldberg. *Linkage identification by non-monotonicity detection for overlapping functions*. **Evolutionary Computation**. 7 (4), 377-398, 1999.

[Thompson] A. Thompson. *Evolutionary design for novel technologies*. **IEEE Colloquium on Evolutionary Hardware Systems**, p.4, 1999.