

# Evolutionary Neuroestimation of Fitness Functions<sup>\*</sup>

Jesus S. Aguilar-Ruiz, Daniel Mateos, and Domingo S. Rodriguez

Department of Computer Science  
University of Seville, Spain  
{aguilar,mateos,dsavio}@lsi.us.es  
<http://www.lsi.us.es/~aguilar>

**Abstract.** One of the most influential factors in the quality of the solutions found by an evolutionary algorithm is the appropriateness of the fitness function. Specifically in data mining, in where the extraction of useful information is a main task, when databases have a great amount of examples, fitness functions are very time consuming. In this sense, an approximation to fitness values can be beneficial for reducing its associated computational cost. In this paper, we present the Neural–Evolutionary Model (NEM), which uses a neural network as a fitness function estimator. The neural network is trained through the evolutionary process and used progressively to estimate the fitness values, what enhances the search efficiency while alleviating the computational overload of the fitness function. We demonstrate that the NEM is faster than the traditional evolutionary algorithm, under some assumptions over the total amount of estimations carried out by the neural network. The Neural–Evolutionary Model proves then useful when datasets contain vast amount of examples.

## 1 Introduction

Evolutionary Computation can be basically used to tackle with two sorts of problems: optimization and machine learning. In machine learning, a dataset plays the role of knowledge base, so each individual of the population is evaluated taking into account every example of that dataset, or in some cases, a representative subset of it [4]. To produce a knowledge model -in any form of knowledge representation- from a dataset is a task -among several steps of the knowledge discovery from databases (KDD) process- that consumes a lot of time when evolutionary algorithms are used, so that it is advisable to incorporate a technique able to reduce the complexity.

One of the most influential factors in the quality of the solutions found by an evolutionary algorithms (EA) is the appropriateness of the fitness function. This function assigns a value of merit (goodness) to each individual of the population

---

<sup>\*</sup> The research was supported by the Spanish Research Agency CICYT under grant TIC2001–1143–C03–02.

for every generation, so that the number of calls to this function is exactly  $PG$ , where  $P$  is the size of the population and  $G$  is the number of generations. However, for machine learning problems, the fitness function needs to analyze every example from the dataset, which can have many attributes. Therefore, the overall cost of the fitness function during the evolutionary process is  $NMPG$ , where  $N$  is the number of examples of the dataset and  $M$  is the number of attributes of every example. As it is a very high cost, we then usually try to optimize the efficiency of the evolutionary algorithm, i.e. to decrease the computational cost by reducing some of these parameters.

When  $N$  is the aim of the reduction, then we are dealing with *editing or condensed algorithms* [5]. On the other hand, if we try to reduce  $M$ , the techniques are so-called *feature selection* or *attribute extraction*. As  $N$  and  $M$  are not factors of the evolutionary algorithm per se, we will call them *external factors*. In addition,  $P$  and  $G$ , which will be called *internal factors* of the evolutionary algorithm, could be also reduced. A small value of  $P$  might limit the exploration through the search space and a small value of  $G$  might reduce the exploitation within the search space. These two parameters,  $P$  and  $G$ , are by far easier to modify than  $N$  or  $M$ , because they don't need a specific preprocessing algorithm but some experience.

There exist many algorithms to reduce  $N$ , most of them based on the Nearest Neighbour technique [5,6,7] or on attribute projections [2]. In the same way, other algorithms can reduce the number of attributes, such as RELIEF [8]. Editing or feature selection algorithms would not lead to a reduction in quality of results.

In this work, we deal with the reduction of the computational cost of evaluating the fitness function. None of the internal ( $P$  or  $G$ ) or external ( $N$  or  $M$ ) parameters are reduced, but our original approach consists in reducing directly the computational cost of the fitness function (FF) by using another "approximate" function (FF\*). This function FF\* will estimate the values of FF using less computational resources.

Neural networks (NN) are robust and exhibit good learning and generalization capabilities in data-rich environments and are particularly effective in modeling relationships with nonlinearities. The estimator FF\* is designed as a NN, which will be trained through the evolutionary process until it can estimate by itself the fitness function, and next generations will begin to use the neural network as a FF estimator FF\*, at the same time that it is trained. From a time, no more evaluations are calculated by the fitness function, but all individuals will be estimated by the neural network. NN have been chosen because they are very easy to implement, adapt well to many different search spaces, provides high quality results, and fundamentally because the training cost is very low and the test cost is virtually nil. Combinations between artificial neural networks and evolutionary algorithms have been analysed in many works, mainly for optimization tasks [9,10]. Neural networks have rarely been used as a function evaluator to facilitate EA search. In fact, little research has focused on neural networks being used for this purpose [11,12].

The main advantage of our approach resides in the significant reduction of the computational cost, since most of the evolutionary process the algorithm has not to calculate the fitness function values, as these are provided by the NN, very much faster than the fitness function. This is so because from the moment in which the NN calculates all the values, the parameters  $N$  and  $M$  disappear from the expression  $NMPG$ , having no influence on the overall cost.

The paper is organized as follows: in Section 2 the motivation of our approach is described; the NEM is presented in Section 3 and its computational cost is calculated; we demonstrate the efficiency of our approach with respect to the traditional EA in Section 4 and some properties about its use are given; finally, the most interesting conclusions and future research directions on this issue are presented in Sections 5 and 6, respectively.

## 2 Motivation

The computational cost of an EA can be separated into several factors: evaluation, selection, recombination, genetic operators, etc. Specifically in the supervised learning context, our ten-fold cross-validated experiments [3] carried out with datasets from the UCI repository [13] revealed that the evaluation function took approximately 85% of the total computational cost. This fact motivates us to analyze how to reduce this percentage by means of a fast evaluation of individuals from the population.

In Section 1 the internal and external factors of the EA were mentioned. Logically, internal factors as  $P$  and  $G$  have great influence on the quality of results. However, we know that a greater value for  $P$  is not necessarily going to provide better results, and an increasing of  $G$  might have the same behaviour over the EA if, for instance, the process is falling into a local minimum. We could fit these parameters, hypothetically knowing that the greater value is, better results might be found. Therefore, it is not technically appropriate to try to reduce the computational cost by decreasing the value of some of these parameters, since the quality of the results would be affected to a large extent. In addition, the range for  $P$  used to be small, for instance in machine learning problems, it is ranging from 20 to 200.

The external factors are not easily manipulated by the EA. The number of examples  $N$  and the number of attributes  $M$  should be constant values along the evolutionary process. As mentioned above, a reduction method can be tried before applying the EA, as a preprocessing technique, although the same result quality is not guaranteed. The largest value out of  $N$ ,  $M$ ,  $P$  and  $G$  is usually  $N$  so that dataset reduction techniques are mostly used. However, the cost of these techniques is mostly quadratic.

As calculating the fitness of an individual will have complexity  $NM$ , in some cases, a sampling method is embedded in the FF in order to reduce  $N$  to a small value  $N'$ . Nevertheless, the correct selection of the  $N'$  examples from the dataset is another difficult problem if we wish to provide exactly the same fitness value that when using the  $N$  original examples, i.e. the subset must be reliable.

Other ideas consist in conserving fitness values of individuals from older populations (previous generations) in order to save these values. We think that this technique is interesting when almost all the individuals are similar, i.e. when the population is very “old”, as the probability of two individuals are equal from consecutive populations is higher (many efficient data structures exist to store the historic information from individuals and fitness value calculations.) Nevertheless, there is relatively little work in the literature that address how to make full use of the search history to guide the future search. It is important to note that, at this point, the similarity among individuals is directly depending on the individual length and thus on the cardinality of the alphabet, what requires an appropriate encoding of the search space [1].

To deal with the problem from the point of view of  $M$  is only advisable when the dataset has hundreds or thousands of attributes. In fact, nowadays there are appearing many medical datasets with those amounts of attributes, such as the leukemia dataset used in [14] to discover the distinction between acute myeloid and acute lymphoblastic sorts of leukemia, which has 6817 attributes (genes) and only 38 examples (patients).

In short, a decreasing of  $N$ ,  $M$ ,  $P$  and/or  $G$  can lead to a significant reduction of the computational cost. However, this reduction can have some side effects on the quality of the results, as we have seen before.

In this work, we tackle the fitness function evaluation as an incremental learning problem, i.e. we will try to learn to calculate fitness values by using a neural network, which is progressively trained through the evolutionary process, like an embedded subprocess in the fitness function.

The aim is to achieve a neural network able to provide similar values as the fitness function, i.e. to construct a fitness function estimator. Thus, the Neuro–Evolutionary Model consists of an evolutionary algorithm and a neural network that is learning from the fitness function at the same time as estimating new values.

### 3 Neural–Evolutionary Model

Before presenting theoretically the Neural–Evolutionary Model (NEM), we will give some definitions. Henceforth,  $FF$  denotes the fitness function and  $NN$  is the neural network. In addition,  $C_{FF}$  is the cost of the evaluation of one individual;  $C_{NNT}$  is the cost of training the  $NN$ ; and  $C_{NNE}$  is the cost of the estimation of the fitness value of an individual with the  $NN$ . A function  $\varphi : \mathbb{N} \rightarrow \mathbb{N}$  indicates how many individuals will be evaluated by  $FF$  and  $NN$  at each generation. Thus, at generation  $i$ , a number  $\varphi_i$  of individuals will be directly evaluated by the  $FF$  and the remainder  $P-\varphi_i$  will be estimated by the  $NN$ .

The idea behind the NEM is very simple: at the generation  $i$  the fitness value of  $P-\varphi_i$  individuals will be obtained by the  $FF$  and the fitness value of  $\varphi_i$  individuals will be estimated by the  $NN$ , which will receive  $P-\varphi_i$  individuals together with  $P-\varphi_i$  fitness values to be trained.

---

```

Function NEM( $E$ )
  var  $\bar{x}$ : Individual
   $i := 1$ 
   $P_0 := \text{Initialize}()$ 
  Evaluation( $P_i, i$ )
  while  $i < \text{num\_generations}$ 
     $i := i + 1$ 
    for  $j \in \{1, \dots, |P_{i-1}|\}$ 
       $\bar{x} := \text{Selection}(P_{i-1}, i, j)$ 
       $P_i := P_i + \text{Recombination}(\bar{x}, P_{i-1}, i, j)$ 
    end for
    Evaluation( $P_i, i$ )
  end while
  return best_of( $P_i$ )
end NEM

Procedure Evaluation( $Q, i$ )
  var  $\bar{x}$ : Individual; Fitness: vector  $[[Q]]$  of real
  FFS:= $\varphi_i$  individuals randomly selected from  $Q$ 
  for  $\bar{x} \in \text{FFS}$ 
    Fitness( $\bar{x}$ ):=FF( $\bar{x}$ )
    Train NN using  $\bar{x}$  and Fitness( $\bar{x}$ )
  end for
  for  $\bar{x} \in Q - \text{FFS}$ 
    Fitness( $\bar{x}$ ):=NN( $\bar{x}$ )
  end for
end Evaluation

```

---

**Fig. 1.** Pseudocode of NEM.

Basically, the NEM is an EA with a more complex evaluation function, which is separated from the EA pseudocode to be better explained. In Figure 1 the EA is described, following a schema inspired by the works of Angeline and Pollack [15]. The FF is called *Evaluation*, and it is depicted at the bottom within the same Figure.

Evaluation uses a vector called Fitness, with size  $P$ , to save the fitness values of every individual. The procedure has four parts: selection of  $\varphi_i$  individuals; fitness calculations of  $\varphi_i$  individuals; training of the NN using  $\bar{x}$  (an individual) and the fitness value provided by Fitness( $\bar{x}$ ) before; and fitness value estimations by the NN. The first part involves the function  $\varphi_i$ , and it is depending on the number of the generation  $i$ . The selected individuals are included in a set, named FFS (Fitness Function Subset), and they will be evaluated by using the FF. Each individual  $\bar{x}$  together with its fitness value Fitness( $\bar{x}$ ) is given to the NN as a training example. After the first loop,  $\varphi_i$  individuals have been evaluated and the NN has been trained with them. Later, the individuals not selected before, exactly  $P - \varphi_i$ , are given as input to the NN to estimate the fitness values.

Table 1 shows the cost at each generation  $i$ . The cost of evaluating one individual with FF is NM; the cost of training the NN with one individual is T; the cost of estimating the fitness value of one individual with NN is E. For an EA, the cost of evaluating all the  $P$  individuals during the  $G$  generations will be  $C_{EA} = NMGP$ , as it is shown in Table 1.

**Table 1.** Cost of evaluating P individuals through G generations.  $C_{FF}$  is the cost of the fitness function;  $C_{NNT}$  is the cost of training the neural network;  $C_{NNE}$  is the cost of estimating the fitness values with the neural network.

Generation	$C_{FF}$	$C_{NNT}$	$C_{NNE}$	$C_{EA}$
1	$\varphi_1$ NM	$\varphi_1$ T	$(P-\varphi_1)$ E	PNM
2	$\varphi_2$ NM	$\varphi_2$ T	$(P-\varphi_2)$ E	PNM
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	$\varphi_i$ NM	$\varphi_i$ T	$(P-\varphi_i)$ E	PNM
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
G	$\varphi_G$ NM	$\varphi_G$ T	$(P-\varphi_G)$ E	PNM
Total	$\Phi$ NM	$\Phi$ T	$(GP-\Phi)$ E	GPNM

The total cost of the Neural–Evolutionary Model,  $C_{N\circ E}$  would be as follows:

$$C_{N\circ E} = C_{FF} + C_{NNT} + C_{NNE}$$

$$C_{FF} = (\varphi_1 + \dots + \varphi_G)NM = NM \sum_{i=1}^G \varphi_i$$

$$C_{NNT} = (\varphi_1 + \dots + \varphi_G)T = T \sum_{i=1}^G \varphi_i$$

$$C_{NNE} = (P - \varphi_1 + \dots + P - \varphi_G)E = EGP - E \sum_{i=1}^G \varphi_i$$

Therefore, the cost of  $C_{N\circ E}$  will be:

$$C_{N\circ E} = (NM+T-E)\Phi + EGP \quad (1)$$

where  $\Phi = \sum_{i=1}^G \varphi_i$ .

## 4 Efficiency

In this section, the aim is to demonstrate that  $C_{N\circ E}$  is smaller than  $C_{EA}$  under some assumptions over  $\varphi$ . In other words, there exist functions  $\varphi$  which make the NEM faster than the EA.

**Theorem 1.** *If  $NM \geq GP$  and  $\Phi < GP-1$  then the cost of the NEM,  $C_{N\circ E}$ , is smaller than the cost of the EA,  $C_{EA}$ .*

*Proof.* Let suppose that

$$NM\Phi + GP < GPNM$$

then

$$\Phi < \frac{\text{GPNM-GP}}{\text{NM}} = \text{GP} \left( 1 - \frac{1}{\text{NM}} \right)$$

if  $\text{NM} \geq \text{GP}$  then

$$\Phi < \text{GP} - 1 \leq \text{GP} \left( 1 - \frac{1}{\text{NM}} \right)$$

We have to note that NM has a large value in comparison to GP. The range from GP normally varies in machine learning problems from 5000 (G=100 and P=50) to 100000 (G=500 and P=200), although greater values can be used. However, since the values for N and M are previously known, we can assure that  $\text{NM} \geq \text{GP}$  by searching values for G and P that satisfy the condition.

Therefore,

$$\left. \begin{array}{l} \Phi < \text{GP} - 1 \\ \text{NM} \geq \text{GP} \end{array} \right\} \Rightarrow C_{N \diamond E} < C_{EA}$$

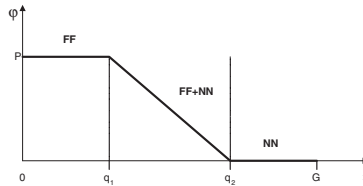
In the next subsection, we are going to analyze the case of  $\varphi$  piece-wise linear, i.e. the number of estimations done by the NN is linear with respect to the number of generations.

### 4.1 Piece-Wise Linear $\varphi$ .

The general case for  $\varphi$  is:

$$\varphi = \begin{cases} P & 0 < i < q_1 \\ -\frac{P}{q_2 - q_1}(i - q_2) & q_1 < i \leq q_2 \\ 0 & q_2 < i \leq G \end{cases} \quad (2)$$

where  $q_1$  and  $q_2$  are constant values within  $[1, G]$ . Figure 2 shows graphically this function.



**Fig. 2.** Linear function  $\varphi$ .

During the first  $q_1$  generations the fitness values are generated by using the FF. At generation  $i = q_1 + \frac{q_2 - q_1}{P}$  the function  $\varphi_i$  takes the value P-1, so that the other individual is estimated by using the NN. As  $i$  increases, more individuals will be estimated with NN and lesser number of individuals will be calculated with FF. From the generation  $i = q_2 + 1$ , the NN estimates all of the individuals and FF is not used anymore, since NN is supposed to have learned from FF

outputs. It is worth to note that  $q_1$  determines when is NN to be used it, and  $q_2$  when is not FF being used anymore.

The cost of  $\Phi$  is as follows:

$$\Phi = \sum_{i=1}^{q_1} \varphi_i + \sum_{i=q_1+1}^{q_2} \varphi_i + \sum_{i=q_2+1}^G \varphi = \frac{P}{2} (q_1 + q_2 - 1) \quad (3)$$

This result is very important because the cost of the NEM is not depending on  $q_1$  or  $q_2$  separately, but the sum  $q_1 + q_2$ . This means that the cost of NEMs will be the same if  $q_1 + q_2$  are equal for both cases. Figure 3 illustrates this fact. (An open problem is to determine which NEM will have better quality, depending on the values of  $q_1$  and  $q_2$ , for those NEMs which have the same  $q_1 + q_2$  and  $G$ .)

Now, we are going to demonstrate that the cost of the function  $\varphi$  is actually small and satisfies the Theorem 1.

Replacing  $\Phi$  from Equation 3 into Equation 1:

$$C_{N\odot E} = NM \frac{P}{2} (q_1 + q_2 - 1) + PG$$

The cost of the NEM will be smaller than that of the EA when:

$$NM \frac{P}{2} (q_1 + q_2 - 1) + PG < GPNM$$

Simplifying,

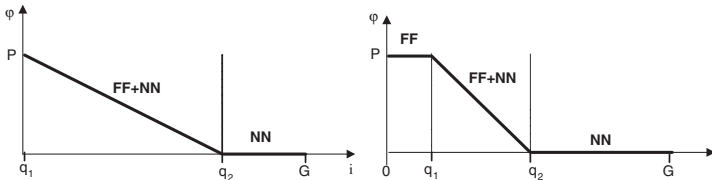
$$G > \left( \frac{q_1 + q_2 - 1}{2} \right) \left( 1 + \frac{1}{NM-1} \right)$$

Assuming that  $NM > 1$  then  $\frac{1}{2} \left( 1 + \frac{1}{NM-1} \right) \leq 1$ . Replacing,  $G > q_1 + q_2 - 1$  and therefore,

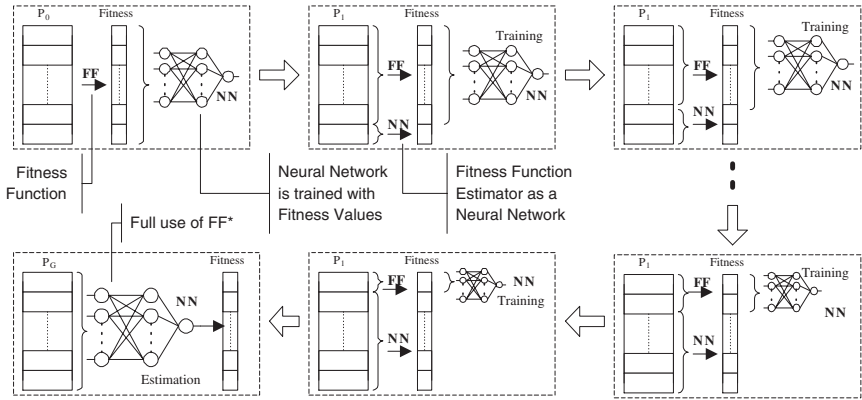
$$G \geq q_1 + q_2 \Rightarrow C_{N\odot E} < C_{EA}$$

when  $\varphi$  is as shown in Equation 2. By definition,  $q_1 + q_2 < 2G$  and  $q_1 < q_2$  (as  $q_1 = q_2$  means no NN is being used together with the FF), so if  $q_2 = G - k$ , then  $q_1 \leq k$ ,  $k$  being any value in  $[0, G[$ .

To understand the operation of the NEM together with the linear function  $\varphi$ , an example of specific linear function is given, where  $q_1 = 0$  and  $q_2 = P$ . In



**Fig. 3.** Two functions  $\varphi$ , having the same computational cost, could provide different results with respect to the quality.



**Fig. 4.** The evolutionary process, within which the neural network is trained with the previously calculated fitness values.

this case, at the first generation all the individuals are evaluated with FF. At the second generation, every individual is evaluated with FF except for one, which is estimated with NN. With the  $P-1$  evaluations, the NN is trained. An so on, until all of the individuals are estimated by NN, when  $q_2 = P+1$ . The process is graphically depicted in Figure 4.

## 5 Conclusions

Specifically, when datasets contain great amount of examples, the use of evolutionary algorithms to extract useful information in form of decision rules is a task that takes a lot of time. Fitness functions are very time consuming so that an approximation to fitness values can be beneficial for reducing its associated computational cost.

We present a Neural–Evolutionary Model (NEM), which uses a neural network as a fitness function estimator. The neural network is trained through the evolutionary process and used progressively to estimate the fitness values. We demonstrate that the NEM is faster than the traditional evolutionary algorithm, under some assumptions over the gradualness of the neural network training and its use as an estimator. In general, for greater values of  $N$  and  $M$ , better results the NEM will give with respect to efficiency.

Despite some approaches include the use of NNs together with EAs, to our knowledge, the design of the NEM is original, since the algorithm learns through two different levels: one is the evolutionary learning and the other is the neural learning, which helps to the first one.

The resulting hybrid approach find solutions with fewer computational resources devoted to the efficient evaluation of the fitness function, so the NEM is specially useful for the evolutionary extraction of interesting knowledge from datasets which have a huge size.

## 6 Future Works

After demonstrating the efficiency of the NEM, our research is focused on the *effectiveness* of the approached model, i.e. on experimentally proving that the quality of the results is also conserved. Another interesting future research direction consists in studying different forms for the function  $\varphi$ , in order to compare the quality of different approached NEMs.

## References

1. Jesus S. Aguilar–Ruiz, J. C. Riquelme, and C. Del Valle, “Improving the evolutionary coding for machine learning tasks,” in *Proceedings of the 15<sup>th</sup> European Conference on Artificial Intelligence (ECAI’02)*, Lyon, France, August 2002, pp. 173–177.
2. Jesus S. Aguilar–Ruiz, J.C. Riquelme, and M. Toro, “Data set editing by ordered projection,” *Intelligent Data Analysis*, vol. 5:5, pp. 1–13, 2001.
3. Jesus S. Aguilar–Ruiz, J.C. Riquelme, and C. Del Valle, “Evolutionary learning of hierarchical decision rules,” *IEEE Transactions on Systems, Man and Cybernetics, Part B*, Vol. 33, Issue 2, pp. 324–331, April 2003.
4. George H. John and Pat Langley, “Static versus dynamic sampling for data mining,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.
5. B. V. Dasarathy, *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, 1991.
6. E. Fix and J. L. Hodges, “Discriminatory analysis, nonparametric discrimination consistency properties,” Technical Report 4, US Air Force, School of Aviation Medicine, Randolph Field, TX, 1951.
7. D. R. Wilson and T. R. Martinez, “Reduction techniques for instance–based learning algorithms,” *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.
8. I. Kononenko, “Estimating attributes: analysis and extensions of relief,” in *Proceedings of European Conference on Machine Learning*. 1994, Springer-Verlag.
9. Xin Yao and Young Liu, “A new evolutionary system for evolving artificial neural networks,” *IEEE Transactions on Neural Networks*, vol. 8, pp. 694–7130, 1997.
10. Xin Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
11. A.G.Pipe, T.C. Fogarty, and A. Winfield, “Balancing exploration with exploitation - solving mazes with real numbered search spaces,” in *Proceedings of the First IEEE Conference on Evolutionary Computation*, 1994, pp. 458–489.
12. D.W. Coit and A.E. Smith, “Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach,” *Computers & Operations Research*, 1995.
13. C. Blake and E. K. Merz, “UCI repository of machine learning databases,” 1998.
14. T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander, “Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring,” *Science*, , no. 285, pp. 531–537, 1999.
15. P. Angeline and J. Pollack, “Evolutionary module acquisition,” in *Second Annual Conference on Evolutionary Programming*, 1993.