

International Journal of Computational Intelligence and Applications  
© World Scientific Publishing Company

## NEURAL LOGIC NETWORK LEARNING USING GENETIC PROGRAMMING

HENRY WAI-KIT CHIA\* and CHEW-LIM TAN†

*School of Computing, National University of Singapore,  
3 Science Drive 2, Singapore 117543*

Received 9 July 2001

Revised 30 September 2001

Neural Logic Networks or *Neulonets* are hybrids of neural networks and expert systems capable of representing complex human logic in decision making. Each *neulonet* is composed of rudimentary *net rules* which themselves depict a wide variety of fundamental human logic rules. An early methodology employed in *neulonet* learning for pattern classification involved weight adjustments during back-propagation training which ultimately rendered the *net rules* incomprehensible. A new technique is now developed that allows the *neulonet* to learn by composing the *net rules* using genetic programming without the need to impose weight modifications, thereby maintaining the inherent logic of the *net rules*. Experimental results are presented to illustrate this new and exciting capability in capturing human decision logic from examples. The extraction and analysis of human logic *net rules* from an evolved *neulonet* will be discussed. These extracted *net rules* will be shown to provide an alternate perspective to the greater extent of knowledge that can be expressed and discovered. Comparisons will also be made to demonstrate the added advantage of using *net rules*, against the use of standard boolean logic of negation, disjunction and conjunction, in the realm of evolutionary computation.

*Keywords:* Neural network, genetic programming, rule-based learning, data mining

### 1. Introduction

Genetically programmed neural networks have generated much interest in machine learning research over the past few years. One approach is to allow the entire neural network to be subjected to genetic operations without the need to resort to chromosome representation of the network architecture. Gaudet's work is representative of this approach where he applied genetic programming on logic-based neural networks which basically represent standard boolean logic namely, negation, conjunction and disjunction.<sup>1</sup> However, neural networks can also be used to represent non-standard logic. A Neural Logic Network or simply *Neulonet* has been proposed that emulates human decision logic which is often too complex to be expressed neatly using standard logic.<sup>2,3</sup>

\*e-mail: chiawaik@comp.nus.edu.sg

†e-mail: tancl@comp.nus.edu.sg

2 *H. W. K. Chia & C. L. Tan*

In the following sections, we shall begin by focusing on the integration of *neulonets* into a genetically programmed environment. Section 2 introduces the concept of *neulonets* with emphasis on how they can be composed to form complex decisions. Section 3 describes a GP system for evolving *neulonets* and discusses pertinent issues relating *neulonet* evolution and the genetic programming paradigm. Moreover, we demonstrate how the system can be used to effectively solve a classification problem. Section 4 focuses on the extraction of human logic rules from the evolved *neulonet*. The depth and extent of knowledge that can be represented using these extracted rules will also be illustrated. In section 5, we provide experimental results to substantiate the use of *neulonets* as an improvement over the use of standard boolean logic networks in genetic programming. The concluding section will give a summary and discuss future works.

## 2. Neural Logic Network

A *Neulonet* differs from other neural networks in that it has an ordered pair of numbers associated with each node and connection as shown in Fig. 1(a). Let  $Q$  be the output node and  $P_1, P_2, \dots, P_N$  be input nodes. Let values associated with the node  $P_i$  be denoted by  $(a_i, b_i)$ , and the weight for the connection from  $P_i$  to  $Q$  be  $(\alpha_i, \beta_i)$ . The ordered pair for each node takes one of three values, namely, (1,0) for true, (0,1) for false and (0,0) for “don’t know”. (1,1) is undefined.

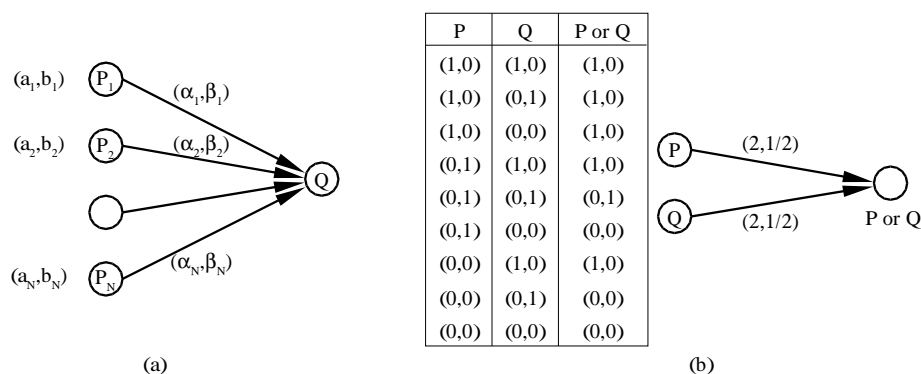


Fig. 1. (a) A schematic representation of a Neural Logic Network – *Neulonet*. (b) A *neulonet* behaving like a disjunction rule.

The following activation function determines the output at  $Q$ :

$$Act(Q) = \begin{cases} (1, 0) & \text{if } \sum_{i=1}^N (a_i \alpha_i - b_i \beta_i) \geq \lambda \\ (0, 1) & \text{if } \sum_{i=1}^N (a_i \alpha_i - b_i \beta_i) \leq -\lambda \\ (0, 0) & \text{otherwise.} \end{cases} \quad (1)$$

where  $\lambda$  is the threshold, usually set to 1.

By applying Eq. (1) on a network in Fig. 1(b), the network behaves like an expert system rule which represents the “OR” operation in Kleene’s three-valued Logical Model as shown by the truth table. Such a network is termed a *net rule* — a rudimentary network that can be chained with other similar networks just like in a conventional rule-based expert system. The power of a *net rule*, however, is not limited to standard boolean logic operations. A wide range of different human logic in decision making can be represented with *net rules* using different sets of weights. Figure 2 shows some examples of *net rules* that emulate a human decision maker’s behavior. These human decision processes are often too complex to be expressed neatly using standard logic. One such complexity is that human reasoning can be biased by giving different degrees of importance to different factors.

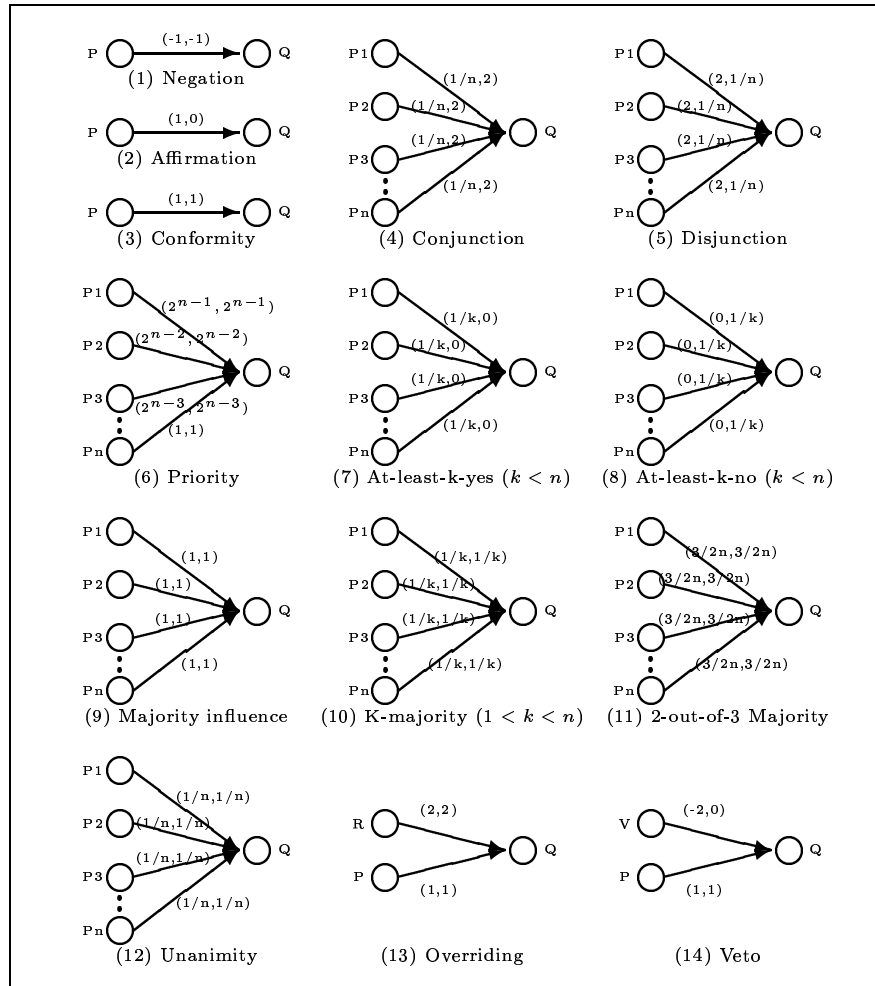


Fig. 2. Examples of component *net rules*.

4 *H. W. K. Chia & C. L. Tan*

For instance, rule (6) “Priority” in Fig. 2 exhibits different degrees of influence on the outcome of  $Q$  with  $P_1$  being the most important factor, while  $P_n$  will only be considered when all other factors are unknown, i.e. (0,0). Another example can be seen in rule (14) “Veto” where  $V$  represents the veto decision. Only when this veto factor constitutes a true decision, will it provide the “Veto Power” to coerce the outcome of  $Q$  to be false regardless of the default decision factor  $P$ . Human reasoning can also be based on the principle of vote-counting. This can be seen in rule (12) “Unanimity” in which the outcome of  $Q$  will only be true (false) when *every* decision factor contributes a true (false) vote. Otherwise,  $Q$  is left unknown. *Net rules* can also be combined to form composite *net rules* to achieve more complicated decisions. In Fig. 3, rule (12) “Unanimity” and rule (14) “Veto” are composed to realize an “XOR” operation in Kleene’s Model.

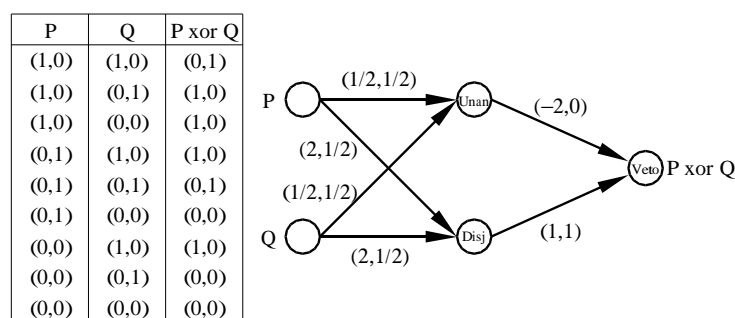


Fig. 3. An XOR composite *neulonet*.

A *neulonet* combines the strengths of neural networks and expert systems.<sup>3</sup> It was proposed that a knowledge engineer could first encode his knowledge into component *net rules* and later use real examples to do refinement training to adjust the weights in the *neulonet*. Training in this case is by means of back-propagation which will pervasively modify all weights in the network, thus rendering the *net rules* indecipherable.<sup>2</sup>

What if the knowledge engineer has only a set of examples without any other knowledge? One approach is to construct a *neulonet* with random weights and train it with the examples. The resultant *neulonet* will be no different from an ordinary neural network, and it is difficult to interpret the logic semantics from the seemingly meaningless set of weights. Another approach is to solve a set of inequalities arising from the activation function in Eq. (1).<sup>2</sup> The solution is not unique. Moreover, the process becomes more difficult with increasing number of inputs and it may not be always possible to find a set of interpretable weights to satisfy all the inequalities.

In view of the above, a novel way of *neulonet* training by means of a genetic programming paradigm is now introduced.

### 3. Integrating Neural Logic Networks With Genetic Programming

Genetic programming is an extension of the conventional genetic algorithm where instead of subjecting bit patterns to genetic evolution, the individuals in the gene population are computer programs.<sup>4</sup> In the context of *neulonet* evolution, these computer programs are represented in the form of *neulonets*.

#### 3.1. Neulonet Structure Undergoing Adaptation

The structure that undergoes adaptation are the population of *neulonets*, each being a recursive composition of *net rules* and input terminals. In the context of data mining, the input terminals are the attributes of the data set. Moreover, bias decision nodes depicting default true, false and unknown decisions also constitute part of the *neulonet* structure. The initial population of *neulonets* is generated in a similar fashion as Koza's "ramped-half-and-half" generative method.<sup>4</sup>

#### 3.2. Genetic Operations

We describe three fitness-proportionate genetic operations for modifying the *neulonet* structure undergoing evolution. The reproduction operation first selects a single *neulonet* from the population. The selected individual is then copied to the new population.

The crossover operation involves swapping the chosen parts of two *neulonets* with constraints imposed on the swapping process to preserve the syntactic integrity. For instance, swapping should not leave any dangling intermediate output nodes as such nodes will not be able to receive input values during firing. Figure 4 illustrates a crossover operation on two *neulonets*.

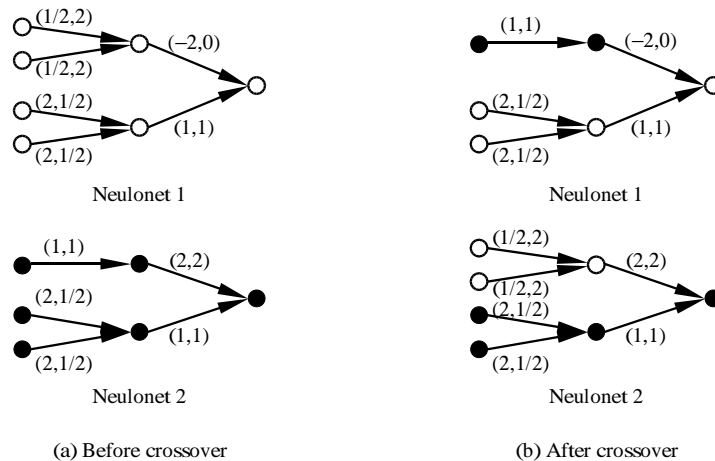


Fig. 4. Two *neulonets* before and after the crossover operation.

6 *H. W. K. Chia & C. L. Tan*

The mutation operation involves changes to a chosen *neulonet*. A random mutation point is picked such that the *neulonet* whose root is the mutation point is replaced by another randomly generated *neulonet*. Figure 5 shows the effect of the mutation operation on a *neulonet* with the “Conformity” *net rule* replaced by an “Overriding” *net rule*.

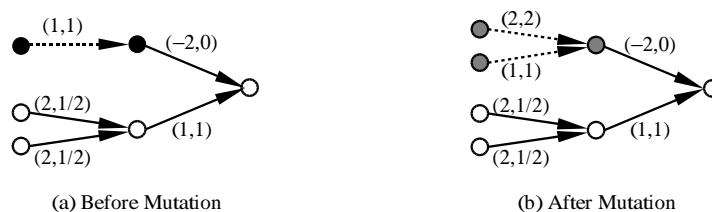


Fig. 5. *Neulonet* with a mutated *net rule*.

It is interesting to note that the “Conformity” *net rule* appears redundant in the activation of a *neulonet* and might be deemed not to be doing anything significant in the evolutionary process. This kind of *net rule* is similar to an *intron* in biology, which is a chromosome that is never expressed and provides spacing between the genes.<sup>5</sup> However, Levenick notes that *introns* are useful in genetic algorithms.<sup>6</sup>

### 3.3. *Fitness Measure and Termination Criterion*

Each *neulonet* in the population is assigned a normalized fitness measure  $f(\epsilon, \sigma; \kappa)$  based on the errors produced by the *neulonet*,  $\epsilon$ , as well as the size of the *neulonet*,  $\sigma$ . The factor,  $\kappa \in [0, 1]$ , is used to weigh the effects of accuracy over size in the fitness measure. A higher value for  $\kappa$  places more emphasis on finding an accurate solution at the expense of the size of the *neulonet*. Moreover, setting  $\kappa$  to an appropriate value allows the extraction of a simpler set of *net rules* that avoids unnecessary complications, so as to provide a satisfactory overall generalization capability.

Termination of evolution is controlled using a parameter that specifies a period in which the termination criterion is examined upon its elapse. The test for termination involves recording the current generation’s fittest *neulonet*, and comparing it against the preceding record. Termination transpires when there is no improvement in the classification accuracy and size of the current recorded *neulonet*, in which case, it is designated as the solution to the problem domain.

### 3.4. *An Illustrative Example*

We shall illustrate the evolutionary process using a simple example from the Space Shuttle Landing Domain<sup>7</sup>. This data set comprises 15 instances and 6 attributes. Table 1 shows each instance being classified as either to auto-land or not. To conform to the input requirements of the *neulonet* structure, every distinct attribute–value

pair has a corresponding boolean attribute in a transformed data set. The valid values for these new attributes can either be yes, no or unknown. In our example, the six-attribute data set transforms to a 16-attribute data set of boolean values.

Table 1. Space Shuttle Landing Domain data set.

Stable	Error	Sign	Wind	Magnitude	Vis	Class
?	?	?	?	?	no	auto
xstab	?	?	?	?	yes	noauto
stab	LX	?	?	?	yes	noauto
stab	XL	?	?	?	yes	noauto
stab	MM	nn	tail	?	yes	noauto
?	?	?	?	OutOfRange	yes	noauto
stab	SS	?	?	Low	yes	auto
stab	SS	?	?	Medium	yes	auto
stab	SS	?	?	Strong	yes	auto
stab	MM	pp	head	Low	yes	auto
stab	MM	pp	head	Medium	yes	auto
stab	MM	pp	tail	Low	yes	auto
stab	MM	pp	tail	Medium	yes	auto
stab	MM	pp	head	Strong	yes	noauto
stab	MM	pp	tail	Strong	yes	auto

Table 2 summarizes the steps in generating a solution using genetic programming. The probabilities assigned to the genetic operations are: (i) reproduction: 15%, (ii) crossover: 80%, and (iii) mutation: 5%. For efficiency considerations, the *neulonet* structure that undergoes evolution is a labeled tree implemented using a prefix-ordered, jump-table approach.<sup>8</sup>

Table 2. An algorithm for evolving *neulonets* using genetic programming.

1. Generate an initial population of *neulonets*.
2. Iteratively perform the following sub-steps until the termination criterion has been satisfied:
  - 2.1. Fire each *neulonet* in the population and assign it a fitness value using a fitness measure.
  - 2.2. Create a new population of *neulonets* by applying the operations of reproduction, crossover or mutation on *neulonets* chosen with a probability based on fitness.
3. The *neulonet* that is identified to be the best individual is designated as the solution to the problem.

For a particular *neulonet* evolution run, a 100% accurate solution was produced in generation 5 which consisted of 8 basic *net rules*. Further evolution produced another 100% accurate solution in generation 11. As a means to further simplify the evolved *neulonet*, a set of simplification rules were applied recursively to identify component *net rules* for elimination or recombination. The final simplified solution consisted of a “Priority” rule rooted at *Q3*, a “At-Least-2-Yes” rule rooted at *Q2*, and a “Negation” rule rooted at *Q1* as shown in Fig. 6.

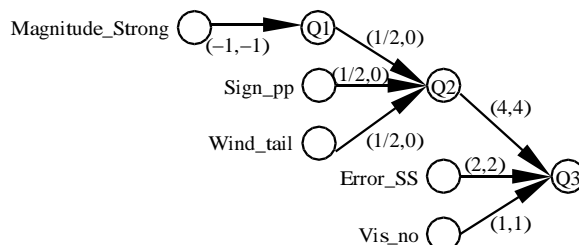
8 *H. W. K. Chia & C. L. Tan*

Fig. 6. A solution to the Shuttle-Landing classification problem.

#### 4. Extracting Rules from Neural Logic Networks

Extracting rules from neural networks has been studied by many researchers.<sup>1,9-12</sup> Existing work, however, basically utilizes standard logic, such as negation, conjunction and disjunction. The rules extracted are generally in the form of a decision tree equivalent to an AND/OR tree based on some classification of the example data. In our present work, the goal is to extract human decision logic from the *neulonets*. The extraction is in fact a straightforward process because the *neulonets* constructed are just a composition of *net rules* which by themselves fully express the human logic in use. These *net rules* can be easily identified from any *neulonet*. As for the case of the Space Shuttle Landing Domain classification problem, the following *net rules* are extracted from the solution in Fig. 6.

- Q1**  $\Leftarrow$  Negation(Magnitude.Strong)
- Q2**  $\Leftarrow$  At-Least-Two-Yes(**Q1**, Sign\_pp, Wind\_tail)
- Q3**  $\Leftarrow$  Priority(**Q2**, Error\_SS, Visibility\_no)

Observe that the set of *net rules* allows us to capitalize on the association between rules and attributes, so as to derive more elaborate decisions that still remain humanly comprehensible. In contrast, production rules of most other rule-based learning systems are independent of one another, and there is no relationship between the attributes across the production rules. For example, the “Priority” *net rule* **Q3** is biased towards rule **Q2** which is of relatively higher importance as compared to the other two decision factors. The contributing factors that constitute *net rule* **Q2** can also be viewed as belonging to the same class of decision makers where each member in the group plays an equally important role in the outcome. Furthermore, *net rule* **Q2** provides us with a concise way of expressing the appropriate rule activations, rather than enumerating every possible combination of decision factors for each rule activation as in the case of production rules.

In layman terms, the decision to auto-land the space shuttle is biased towards any two or more positive factors for a “pp” sign, tail wind and a magnitude that is not strong. Otherwise, the presence (absence) of an “SS” error will result in a decision to auto-land (manual-land) the shuttle. If this error is unknown, then the decision to auto-land (manual-land) depends on a negative (positive) visibility.

We conclude that *net rules* can indeed provide us with an alternative perspective in rule-based learning. Our motivation in considering *net rules* that depict human logic decision making over other forms of rules is due to the greater extent and richness of knowledge that can be expressed and discovered. These rules not only allow us to learn general rules, but also the internal relationship among contributing factors involved in the decision, particularly in their relative level of importance among the factors. This “deeper” knowledge representation is absent from many other forms of rules, including production rules, predicate rules, DNF and MofN rules.

## 5. Empirical Study and Discussion

Our analysis above shows that the proposed approach of using *net rules* in genetic programming should perform well for data sets which encompass some form of ordered logic reasoning. This analysis will be further confirmed via experiments. In particular, we wish to verify whether using an extended set of *net rules* as logical units is comparable to, if not better, than merely using a limited set of *net rules* comprising the standard boolean logic of negation, conjunction and disjunction (rules (1), (4) and (5) of Fig. 2). We selected data sets publicly available from the UC Irvine data repository.<sup>13</sup> Data sets containing discrete attribute data types were transformed to an equivalent binary data set with one attribute for each attribute-value pair in the original set. Moreover, data sets containing continuous-valued attributes were pre-processed using a Chi2 Discretizer prior to the transformation.<sup>14</sup> For the case of data sets having three or more class values, a separate data set that performs a boolean classification for each class value was created.

The entire library of *net rules* given in Fig. 2 was used for our experiments. Each of the *net rules* (4) to (12), were represented by their two- and three-arity equivalents. As a large initial population was required to cater for a wide variety of *neulonets* that could be evolved, we implemented a distributed parallel GP-system on an AP-3000 Fujitsu distributed memory parallel processing system consisting of 32 nodes using a ring-type connection.<sup>15</sup> In our experiments, we used an initial population of 10,000, each having a depth of not more than four component *net rules*, and allowed to evolve to a depth of not more than 17. The evolutionary process would proceed until the classification accuracy and size of the fittest individual were unchanged for the last 100 generations. The weighting factor  $\kappa$  used in the fitness measure was set to 0.99. Results for evolving the best *neulonet* solution from an average of ten runs using the set of *net rules* versus using standard boolean logic are presented in table 3.

### 5.1. Classification Accuracy

In terms of the classification accuracy, *neulonet* evolution provided a comparable, if not better, result than standard logic evolution in all cases. This was especially true

Table 3. Experimental results depicting classification accuracy for the best individual. The numbers below the accuracy value denotes the number of decision nodes and (number of generations).

Data Set	#Instances	#Attributes	<i>Neulonet</i> Evolution	Standard Evolution
Shuttle-landing	15	16	100% 3.0 (34.7)	100% 5.5 (48.3)
Iris-setosa	150	123	100% 1.0 (18.0)	100% 1.0 (12.0)
Iris-versicolour	150	123	99.8% 5.6 (230.0)	99.3% 8.0 (286.7)
Iris-virginica	150	123	99.6% 5.3 (154.0)	98.8% 3.0 (133.3)
Monk-1	432	17	100% 5.6 (31.6)	100% 4.7 (24.0)
Monk-2	432	17	99.8% 19.2 (456.4)	93.2% 20.8 (544.5)
Monk-3	432	17	100% 3.0 (26.6)	99.1% 4.0 (30.3)
Voting-records	435	32	99.6% 14.3 (356.3)	97.7% 13.0 (538.7)
Breast-cancer	699	90	99.5% 20.0 (552.0)	99.4% 20.2 (666.0)
Mushroom	8124	125	100% 6.5 (46.2)	100% 6.25 (71.3)

for data sets having an ordered logic reasoning as in the case of the Shuttle Landing Domain problem, or when the classification rules encompassed a “quantification” of standard boolean logic. For example, in the Monk-2 data set, the given classification rule is as follows:

$$\text{Exactly two of } a_i = 1 \quad \forall i \in \{1, 2, 3, 4, 5, 6\}$$

Clearly, it would be difficult to achieve a high classification accuracy using only a composition of standard boolean logic units. However, in the case of *neulonet* evolution, the expressive power inherent in the set of *net rules* allowed for a more accurate solution tree to be evolved.

## 5.2. Solution Size

Due to the *neulonet's* ability to handle more complex decisions, *net rule* evolution provided more compact solutions than their standard boolean counterparts for the same classification accuracy, particularly for data sets having non-trivial classification rules. Using the best evolution runs for both empirical approaches in the Monk-2 data set, the size profiles for the number of decision nodes is shown in Fig. 7 for accuracies between 70% to 100%. Observe that the profiles for both approaches are comparable in the case of classification accuracies of less than 90%, indicating that the classification rule learned thus far was still relatively simple. However, as the required accuracy increased, the apparent power of *neulonets* in deriving complex decision rules resulted in a significantly smaller solution size.

### 5.3. Number of Generations Required

A general drawback in *net rule* evolution was the longer time needed to converge to an ideal solution due to the larger size of the component *net rule* library. It has to be noted that the experiment conducted actually gave the standard logic an advantage in that the rule set is small (only negation and two- or three-arity conjunctions and disjunctions), while the population sizes in both *neulonet* and standard logic evolution approaches were kept the same at 10,000 individuals. As a result, there were more opportunities for the small set of standard logic rules to quickly evolve to ideal individuals within the population. Thus for problems involving simpler decisions, standard logic evolution attained the required accuracy faster. However, there were cases in which *neulonet* evolution was faster. This was observed from the accuracy profiles for the Monk-2 data set in Fig. 7.

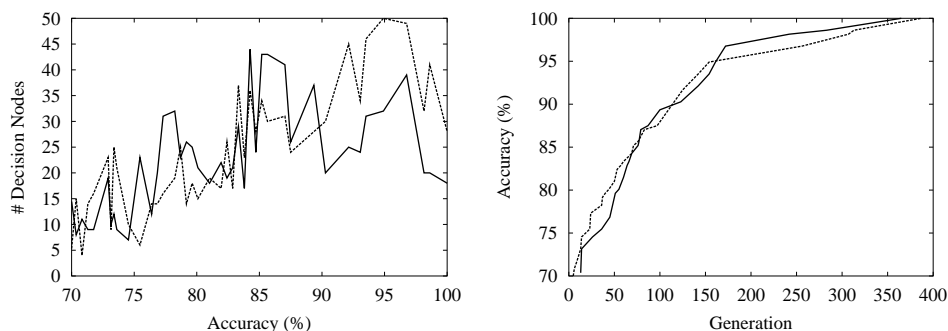


Fig. 7. Size and accuracy profiles for *net rule* (solid-line) versus standard boolean logic (dotted-line) evolution in the Monks-2 data set.

For accuracies of less than 95%, standard logic evolution required slightly less generations. However, as the demand in accuracy increased, it became increasingly difficult to evolve a solution using only standard logic rules. The added advantage in expressing complex rules during *neulonet* evolution, on the other hand, produced a faster rate of convergence.

## 6. Conclusion and Future Work

Genetic programming proves to be an interesting paradigm in constructing *neulonets* with the prescribed human logic *net rules*. The paradigm is also amenable to refinement training. A knowledge engineer could start by constructing *neulonets* based on the human expert's prior knowledge. The constructed *neulonets* may then be subjected to the genetic programming evolutionary process without the need to generate an initial random population. This new mode of neural logic network learning, whether learning from scratch or by refining the existing network, preserves

12 H. W. K. Chia & C. L. Tan

the logic semantics of the extracted rules and facilitates general human decision making.

The *net rule* library will be expanded and more elaborate forms of decision logic will be tested. Variants of crossover and mutation processes, and fitness measures will also be studied. A long term plan in future is to apply genetic programming on fuzzy neural logic networks.<sup>2</sup> For such networks, the values for input and output ordered pairs are real-valued between 0 and 1. Genetic programming will thus be an attempt to evolve the best fuzzy decision rules. We envision an even more exciting horizon of fuzzy *neulonet* learning with genetic programming.

## References

1. V. C. Gaudet, Genetic programming of logic-based neural networks, in *Genetic Algorithms for Pattern Recognition*, eds. S. K. Pal and P. P. Wang (CRC Press, Boca Raton, 1996) 213–226.
2. H. H. Teh, *Neural Logic Network, A New Class of Neural Networks*. (World Scientific, Singapore, 1995).
3. C. L. Tan, T. S. Quah and H. H. Teh, An artificial neural network that modifies human decision making, *Computer* **29** (1996) 64–70.
4. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. (MIT Press, Cambridge Mass., 1992).
5. P. J. Angeline, Genetic programming and emergent intelligence, in *Advances in Genetic Programming*, ed. K. E. Kinnear (MIT Press, Cambridge Mass., 1994) 75–97.
6. J. R. Levenick, Inserting introns improves genetic algorithm success rate: taking a cue from biology, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, eds. R. K. Belew and L. B. Booker, San Mateo, CA, 1991, 123–127.
7. D. Michie, The fifth generation's unbridled gap, in *The Universal Turing Machine: A Half Century Survey*, ed. R. Herken (Oxford University Press, 1988) 466–489.
8. M. J. Keith and C. M. Martin, Genetic programming in C++: Implementation issues, in *Advances in Genetic Programming*, ed. K. E. Kinnear (MIT Press, Cambridge Mass., 1994) 285–310.
9. S. I. Gallant, Extracting rules from neural networks, in *Neural Network Learning and Expert Systems*, chapter 17 (MIT Press, Cambridge Mass., 1993) 315–330.
10. R. Setiono and H. Liu, Symbolic representation of neural networks, *Computer* **29** (1996) 71–77.
11. A. H. Tan, Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing, *IEEE Trans. on Neural Networks* **8** (1997) 237–250.
12. G. G. Towell and J. W. Shavlik, Extracting refined rules from knowledge-based neural networks, *Machine Learning* **13** (1993) 71–101.
13. C. L. Blake and C. J. Merz, UCI Repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html> Irvine, CA: University of California, Department of Information and Computer Science, 1998.
14. H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. The Kluwer International Series in Engineering and Computer Science, Vol. 454 (Kluwer Academic Publishers, Boston, 1998) 161–168.
15. T. Niwa and H. Iba, Distributed genetic programming: empirical study and analysis, in *Genetic Programming: Proceedings of the First Annual Conference 1996*, eds. J. R. Koza, et al., Stanford, CA, July 1996, 339–344.