Algorithmic Design of a Fuzzy-Neural Method

William Franklin

University of Maryland/General Dynamics Land Systems
franklwm@gdls.com

Abstract: A Fuzzy-Neural method is often presented as a black-box toolkit for MathCad. The algorithmic design is hidden inside this black box approach. This paper is intended to develop the algorithmic design of a Fuzzy-Neural method in which fuzzy logic provides a front end input to a neural network that then provides a threshold weighted output from a directed acyclic graph. Fuzzy logic provides an initialization of data inputs while neural networks learn from this input and adapt to it to provide a weighted decision output. From a first glance it would appear that the algorithms that would be used by fuzzy logic are sorting and combinatorial or numerical in order to do the abstract algebra that fuzzy logic requires. Since neural networks operate off the input from a directed acyclic graph they require graph, combinatorial and numerical algorithms. The objective of this paper is to design and develop generic algorithms as well as understand the required algorithmic approach to this technology. The Appendix will feature in detail all algorithms that are developed and are listed off to the side in parenthesis throughout the text.
Applications to prognostics, predictive trending, of system faults will also be examined.

1 Fuzzy Logic Input

Fuzzy logic is used for data input into a Fuzzy-Neural method and provides initialization. Easy to understand, debug and develop fuzzy logic can explain but cannot learn as neural networks do. The main drawback with fuzzy logic is that it requires a rule based approach that utilizes heuristic adaptations. Consider an application regarding engine diagnostics that is used to characterize failure signatures that will be used for prognostics (fault prediction). Two data sets A and B (normalized to 1.0) have a corresponding distribution of data that is mapped according to the membership level representing a degree of certainty. This membership level requires a function that discriminates the data for membership. The evolution from crisp data to fuzzy data is shown in Figures 1 to 3 [1].

The triangular waveform of the membership function is most common and represents a gradual slope for error overlap in temperature value. This broadening in temperature value is determined by the heuristics **If** (condition1) **And** (condition2) **Then** result. These results are then combined into a logical sum for each membership function to produce the broadened waveform.

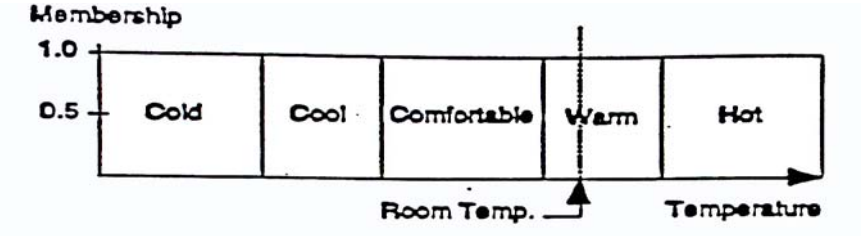


Figure 1
Crisp Sets and Variable

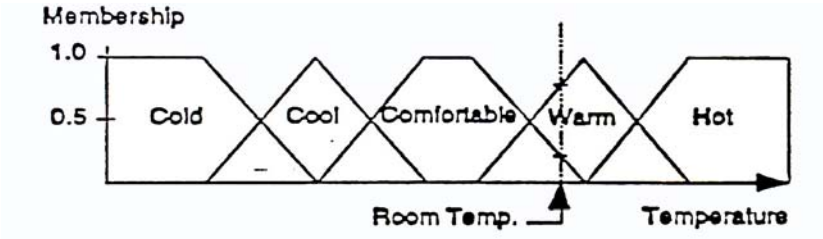


Figure 2
Fuzzy Sets and Crisp Variable

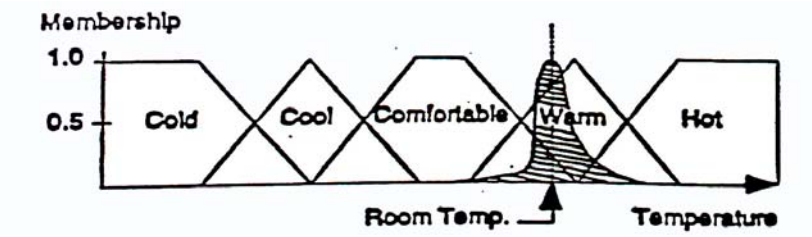


Figure 3
Fuzzy Sets and Fuzzy Variable

A degree of membership is given by the ordinate value from 0 to 1. The heuristics **If** (condition1) **And** (condition2) **Then** result are representative of a transition from initial state vertices to goal state vertices in a state space graph. Such a state space graph could be represented as a fault diagnostics decision tree with nodes describing different subsystem failure checks [2]. Decision trees typically utilize Bayesian theorems for branching left or right from the parent node. A decision tree utilizing fuzzy logic would have to be at least ternary with left, middle and right reaction leafs to allow for the possible states {yes, maybe, no}. For a ternary tree utilizing fuzzy logic the minimum number of comparisons would be O(log n) for n nodes [3 (pp. 339-344)].

Following the methodology of abstract algebra, fuzzy logic merges data sets A and B together in competitive processing to determine the best input values. The process involves an intersection of the two sets of data for A and B:

$\mu A \cap B(x,y) = \min[\mu A(x), \mu B(y)]$ (min mergesort algorithm [3]) (1)

This approach shows how data is correlated initially in fuzzy logic. Defuzzification is used to derive a single output crisp value. This defuzzification represents a weighted union of the intermediary output sets C1(z) and C2(z) that provides a centroid value as a weighted output. This

is represented in the following equations (where C1 is A1∩B1, etc):

$\mu C(x) = \max[\mu C1(x), \mu C2(x)]$ (max mergesort algorithm [3]) (2)

followed by the centroid calculation

$$Z^*_{COA} = \frac{\sum_{i=1}^{q} z_i \mu_C(z_i)}{\sum_{i=1}^{q} \mu_C(z_i)}$$ (weighted sort algorithm [1]) (3)

An example of the fuzzy logic heuristics is given by the following graph [1]

Rule 1: If x is A1 and y is B1 Then z is C1

Rule 2: If x is A2 and y is B2 Then z is C2

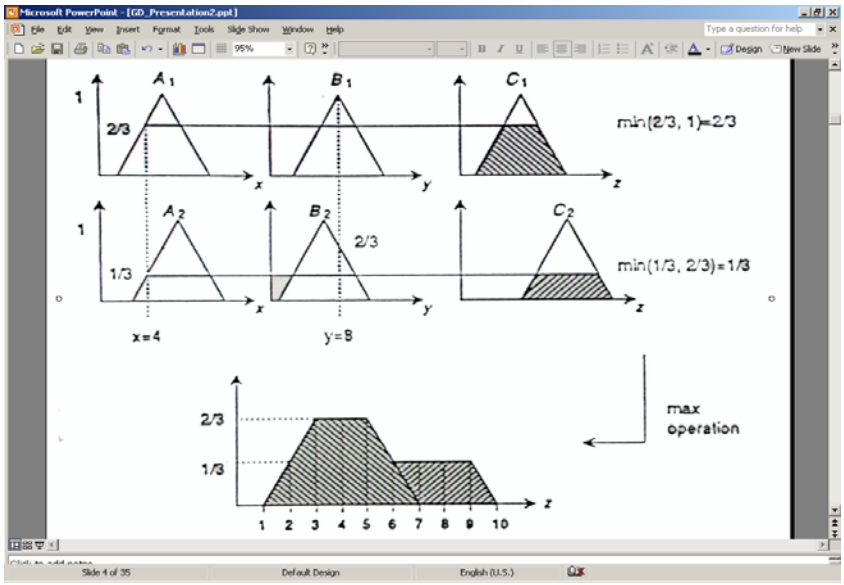


Figure 4
Centroid Calculation

$$Z^*_{COA} = \frac{1\cdot 0 + 2\cdot \frac{1}{3} + 3\cdot \frac{2}{3} + 4\cdot \frac{2}{3} + 5\cdot \frac{2}{3} + 6\cdot \frac{1}{3} + 7\cdot \frac{1}{3} + 8\cdot \frac{1}{3} + 9\cdot \frac{1}{3} + 10\cdot 0}{0 + \frac{1}{3} + \frac{2}{3} + \frac{2}{3} + \frac{2}{3} + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 0} = 5.09$$

Each dataset A and B represents one waveform. The min mergesort algorithm collects the data that intersects the overlapping regions between the waveforms of A and B. In a similar manner the max mergesort algorithm collects the data that is not intersected in the overlapping region between the waveforms of A and B. Instead this data from the max mergesort is a union between the two datasets and is data collected above a threshold maximum setting in the waveforms. Usually this threshold maximum is taken at the 50% crossover between waveforms.

The min and max mergesort algorithms utilize the divide-and-conquer technique [3 (pp. 121-126)]. This provided a straightforward competitive comparison between the two data sets. Although the greedy algorithm Kruskal works with sets and unions it provides path optimization for acyclic graphs. Fuzzy logic deals with data sorting for the intersection and union of data sets. The intersection data would correspond to a cyclic graph and this would invalidate the use of Kruskal's algorithm. Each dataset A, B is first sorted in increasing order using quicksort [3 (pp. 127-132)] which has an average efficiency of O(n log n).

For mergesort the number of worst case comparisons is known to be Θ(nlog n). A worst case estimate for the total running time of these mergesort algorithms is given as [3 (pp. 194-197)]

$$T(n) = Tsort(n) + Tsearch(n) \quad (4)$$

which is O(n log n). In the case of the centroid calculation requiring a weighted sort algorithm there is only a simple calculation each for the numerator and denominator values. The number of comparisons is O(2n) and this also provides a good estimate for the total running time.

2 Neural Networks

Neural networks serve as the processing back end for a Fuzzy-Neural method and provide adaptive learning. Difficult to understand and debug, neural networks rely on fuzzy logic to provide the data initialization. The input from fuzzy logic is provided as a set of nodes that present topological sorting in the form of digraphs. This will allow a feed forward back propagation input for learning [3 (pp. 170-173)]. These digraphs provide breath first search and are dags, directed acyclic graphs that have only forward edges and allow convergence to a solution as indicated by the Output units ajWj,i. In the topological sorting [4] below of Figure 5 for a neural network the final output unit is given as O = A + B + C + D. This combinatorial network becomes a maximum flow network with maximum

bipartite matching [5]. Each hidden unit can be represented as a summation from the input units as the following equations show:

$$\text{Output Unit (at external node i with j internal nodes)} = \sum_j a_j W_{j,i} \bullet \text{(leaf nodes)} \quad (5)$$

$$\text{Hidden Unit (at internal node j with k leaf nodes)} = \sum_k a_k W_{k,j} \quad (6)$$

given k Input Units. This can be expressed comprehensively as

$$\text{Output Unit (at external node i)} = \sum_j a_j W_{j,i} \sum_k a_k W_{k,j} \quad (7)$$

The associated numeric weights Wj,i represent the connection strength from a given nodal sequence to the next level nodal sequence. An activation function g derives the output and each subsequent output nodal sequence can be given as [4 (p. 737)]

$$a_i = g\left[\sum_j a_j W_{j,i} \right] \quad (8)$$

$$a_j = g\left[\sum_k a_k W_{k,j} \right] \quad (9)$$

whereby the node is active (g near 1) when the right inputs are given and (g near 0) when the wrong inputs are given. Adjusting the weights Wk, j changes the activation function and how the network operates. It is better to have a differentiable activation function that allows a gradient response for the weight learning algorithm. The sigmoid function provides optimal behavior since it yields a wide differentiable slope change [6].

A bias weight W0, j could be established such that the node is activated when the weighted sum of (real inputs – bias inputs) is greater than W0, j and this adjusts the slope of the gradient response making the learning process faster or slower. These adjustments also allow the minimization of error on the training set of defuzzified data. The back propagation of error from the output node to the inner nodal sequences can be expressed as a modified error Δi with Errori (error between true output and training set input) in the following equations [4 (pp.741-746)]

$$\Delta i \sim \text{Error}_i * \partial g / \partial W_{j,I} \quad (10)$$

and this modified error is back propagated as

$$\Delta j = (\partial g / \partial W_{k,j}) * \sum_i W_{j,i} * \Delta I \quad (11)$$

The neural network is representative of a maximum bipartite matching graph whose cardinality is determined by the weights Wj,i between different levels of nodal sequences. This cardinality represents those nodal level connections that actually provide the data transmittal to the output [5]. Figure 5 has a cardinality of 1 as indicated by the heavy orange line and this is established by the fuzzy logic heuristics.

The Perceptron Learning and Back Propagation algorithms utilize equations 5-11.

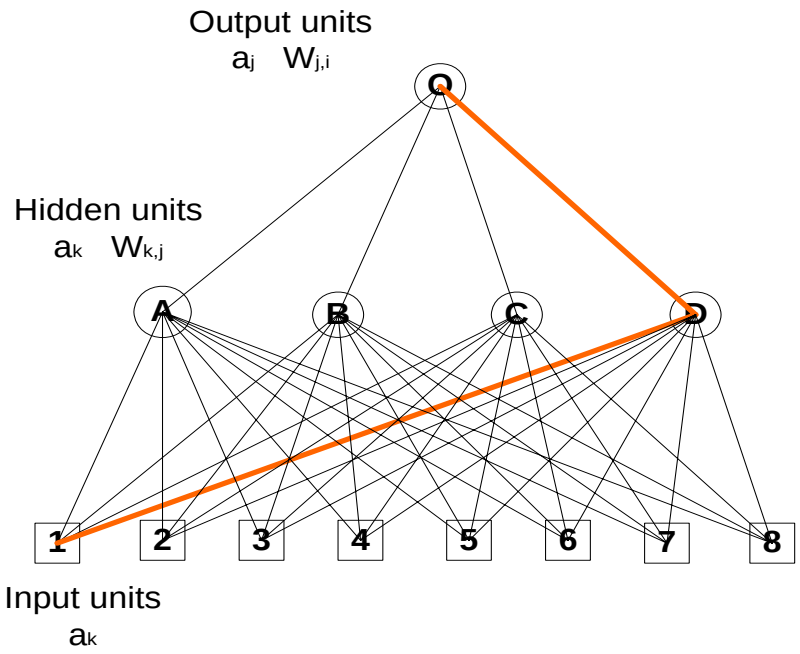
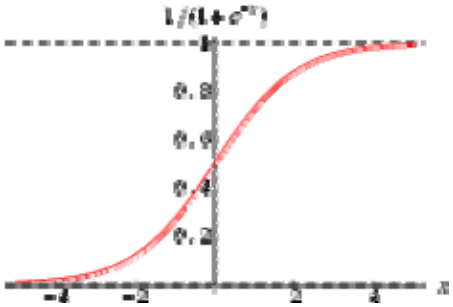


Figure 5
Breadth First Search – Neural Network Nodes



The Perceptron-Learning algorithm applies to a single layer input to output neural network and the number of comparisons and the total time are both estimated by O(k) for k data samples. As shown in Appendix 5 this algorithm consists of straightforward multiplications and summations. The Back Propagation algorithm which applies to a multiplayer network is more complicated as shown in Appendix 6. Applying the fuzzy logic heuristics allows this algorithm to reduce from a high order polynomial given by O(Nk • L • Nj • Ni) to a best case of O(L). The combination of fuzzy logic heuristics and Back Propagation results in a greedy algorithm approach such as a inverted Dijkstra's for path optimization [3 (pp. 319-

323)]. The term modified is used to indicate that the path chosen is not the shortest in distance but the path with the highest weight. The weight would be determined from initial heuristic conditions. Appendix 6 indicates this in red. Both the Perceptron Learning and Back Propagation algorithms also utilize the concepts of backtracking and branch and bound with state spaces for difficult combinatorial problems [3 (pp. 367-385)].

3 Preventive Health Maintenance

It is expected that by combining the attributes of Fuzzy Logic and Neural Networks into a Fuzzy-Neural method that this would provide a prognostic, predictive trending, tool for Preventive Health Maintenance (PHM). A successful prognostic requires accuracy and repeatability in the measurements and data collection as well as precision in the tools used in these measurements. The combination of all three of these – accuracy, repeatability and precision (ARP) must minimize the overall error of measurement in order to provide sufficient predictive trending and lead times to forecast the fault mechanisms in the system under study. If ARP does not satisfy these requirements then either prognostics is not measurable or not reliable in providing any predictive trending. Another serious consequence of unsuccessful ARP is that the lead times provided by predictive trending may not be early enough to avoid imminent failure mechanisms in the system under study. Short lead times before the onset of the failure results in the transition into diagnostics of the fault and causes prognostics to be a useless endeavor. For the Fuzzy-Neural method to be successful it will require datasets from high ARP quality measurements. Such datasets will be needed to train the neural network and initialize it with the proper weights and then the actual datasets of real world results can be evaluated.

An example of the result of providing corrupted ARP quality datasets was a classified SECRET test by the U.S. Army for developing a neural network linked to sensing pixel images that would do pattern recognition of tanks. To initialize the neural network and calibrate the weights two training datasets were provided of tanks camouflaged behind trees, bushes etc. and the same scenes without the tanks. During the training phase the neural network correctly picked out the tanks and the results were considered successful. Upon final testing with random field scenarios it was found that the neural networks were unsuccessful at recognizing the tanks.

Analysis of the training dataset showed that the images taken of camouflaged tanks had all been taken on a cloudy day and the images without the tanks had been taken on sunny days. In calibrating the decision weights the neural network had focused on the brightness of the background terrain and not if a tank was

present or not. By not providing datasets with sufficient diverse scenarios the datasets were inaccurate and did not train the neural network properly [7].

Database archiving leads to profiling fault characteristics. In order to prognosticate the future one must know the past and diagnostics provides past profiles of fault diagnostics which populate database archives. By setting fault limits and trending behavior the onset of a fault could be mapped over time as in Figure 6 [1].

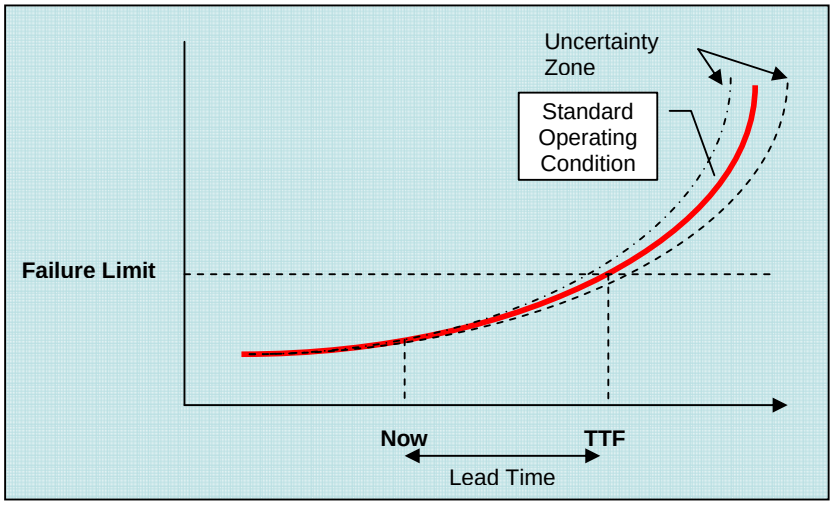


Figure 6
Prognostics to Diagnostics Transition

At the failure limit there is a transition from prognostics to diagnostics. The failure behavior that has been diagnosed yields useful trending behavior for future referencing. One can build up database mining, clustering and warehousing from this. Fuzzy-Neural methods must work in conjunction with database archives to be proficient. The following table could be established to represent database archiving from Figure 6 behavior:

Parametric Data	Late Limit	Standard	Early Limit
Power Management	Fault ?		Fault ?
Bit Commands	Fault ?		Fault ?
Temperature	Fault ?		Fault ?
Voltage	Fault ?		Fault ?

One application for using this Fuzzy-Neural method for PHM is in modeling the behavior of electrical systems such as electromagnetic sensors. Electrical systems

are difficult to model because of electrical jitter, white noise and random spiking behavior that is indicative of an impending failure. One approach for this Fuzzy-Neural method is to determine the theoretical signal response S and compare this to what is actually measured to infer how the system under study is actually behaving. A theoretical response is given by [8]

$$S(t, V, T) = \int \text{Response}(\lambda, T) \bullet \text{VoltageBias}(t, V, T) \bullet BW(\lambda)\, d\lambda \qquad (12)$$

for parameters (t-time, V-voltage, T-temperature, λ-wavelength of radiation, BW – bandwidth). To determine if there is a discrepancy in signal due to the onset of noise behavior we would examine the curve fitting output of the Fuzzy-Neural method as

$$S(t, V, T)|i=O = g\,[\sum j\ \ Wj, i\ (V, T) \bullet g\,[\sum k\ sk\ (t)\ \bullet Wk, j\ (V, T)]\ ] \qquad (13)$$

where sk (t) are the data samples provided as input from the fuzzy logic evaluation.

This equation is a numerical computation algorithm and provides the localization of what is contributing to the failure behavior of the system under study. Appendix 7 features the application of equation 13 to Figure 5.

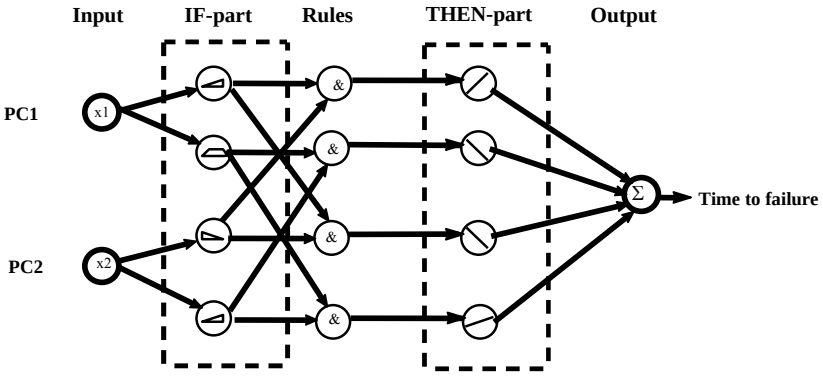


Figure 6
Distributed Fuzzy-Neural Method

As Figure 6 indicates the fuzzy logic heuristics could be distributed throughout the neural network [1]. Specific heuristic conditions and results would determine the connection strengths between nodal sequence levels and their corresponding numeric weights. The activation function as given by equations (8, 9) would exhibit these heuristic conditions in its gradient response. It is these distributed fuzzy logic heuristics that allows the neural network to adapt as it learns through the connection strengths. By appropriate condition and results the efficiency of the network can be increased significantly from initial learning when all the connections have to be established. A feedback loop from the output back to the input would establish faster convergent learning capabilities.

Conclusion

This paper not only presented an algorithmic approach to a Fuzzy-Neural method but also showed how fuzzy logic and neural networks must be coupled together to provide an efficient adaptive transform method. The whole process is adaptive and the neural network responds to the fuzzy logic input and logistics. Artificial intelligence is now established and progresses. Overall, the Fuzzy-Neural method has strong potential applications in prognostics, predictive trending, for Preventive Health Maintenance of high frequency electronic systems or systems of nonlinear behavior. Systems that exhibit nonlinear behavior are hard to analyze because the physical models are not readily established. However, there is miniscule allowance for experimental error as any deviance in ARP can bias the results. Such sensitivity makes the overall approach questionable for practical use outside of nonlinear systems.

Appendix

1) $\mu A \cap B(x,y) = \min[\mu A(x), \mu B(y)]$ (min mergesort algorithm [3])

```
// Algorithm compares data set A at value x to data set B at value y and chooses the
// minimum value between the two sets. This corresponds to an intersection between the
// two sets where membership functions for A, B overlap in degree of membership value.
// The intersection of the membership functions determines the median value for merging
// Inputs sorted data sets A[0,..x,….n-1], B[0,..y,….n-1]; Output intersection set A∩B[0...n-1]
// Mergesort algorithm is applicable for both # 2 and 3
Nmedian = static_cast< int > ( f*n) + 1 // assume f is factor like 0.5 for 50% on waveforms
Merge(A,B,A∩B,Nmedian,True) // for intersection
```

2) $\mu C(x) = \max[\mu C1(x), \mu C2(x)]$ (max mergesort algorithm [3])

```
// Algorithm compares data set C1 at value x to data set C2 at value x and chooses the
// maximum value between the two sets.  This corresponds to a union between the two datasets.
// Inputs data sets C1[0….n-1], C2[0….n-1]; Output set C[0….n-1]
Merge(C1, C2, C, Nmedian, False)  // for union
vector<int> Merge(vector<int> Set1, vector<int> Set2, vector<int> FinalSet,
                  int Nmedian, bool check) //using STL for C++ application, could use arrays
if( check == True)
{ int i=0, j=0, k=0, p=Nmedian, q=Nmedian; }                              // intersection #2
else
{int i=Nmedian, j=Nmedian, k=Nmedian, p=Set1.size(), q=Set2.size() }      // union #3
while( (i < p) && (j < q) )
{
      if( Set1[i] <= Set2[j] )
      {  FinalSet[k] = Set1[i];  i++;  }
      else
      {  FinalSet[k] = Set2[j];  j++;  }
      k++;
}
if(i == p)
  for (int u = j; u <= q-1; u++)
      FinalSet[u] = Set2[u];
```

```
else
  for(int m = i; m <= p-1; m++)
     FinalSet[m] = Set1[m];
return FinalSet;
```

3) $$Z^*_{COA} = \frac{\sum_{i=1}^{q} z_i \mu_C(z_i)}{\sum_{i=1}^{q} \mu_C(z_i)}$$

```
Weighted Sort Algorithm [1]
// Input grid parameter values z[1..q]; data set C[0..n-1] which is described by # 2 and 3
above
// Output centroid calculation Zcoa as numerator / denominator for crisp value.
numerator = 0
for i = 1 to q
    numerator = numerator + ( z[i] * C[i] )
end for
// we have numerator, now normalize to denominator
denominator = 0.0
for i = 1 to q
    denominator = denominator + C[i]
end for
if ( denominator != 0.0 )
  Zcoa =  numerator / denominator
else
  Zcoa = numerator
```

4) Perceptron Learning Algorithm [4 (p. 741)]

```
// inputs: example set with input vector X(x1, …, xN) and output y,
// network - perceptron with weights Wj,i and activation function g
function Perceptron-Learning(samples, network) returns network
        repeat
                for each k in samples do
                   in = ∑j Wj,i * xj[k]
                   Error = y[k]–g(in)
                   Wj,i  =  Wj,i + α * Error * xj[k] * ∂g/∂Wj,i
            end for
        until stop_criterion
return network
```

5) Back Propagation Algorithm [4 (pp. 744-748)]

```
// inputs: examples – input vector x, output vector y, weights Wj,i and g,
// inputs: network – multilayer network L layers.
function Back_Prop(examples, network) returns neural network
   repeat
     for each k in samples do
    if fuzzy logic condition1 then k = 1 // as in Figure 5
    // reduction to single source allows Dijkstra's approach
          for each node j in input layer do
```

```
        aj = xj[k]
      end for
      for layer = 2 to L do
        ai = g [∑j aj Wj,i]
      end for
      for each node i in the output layer do
        Δi = (yi[k]–ai) * ∂g/∂Wj,i
      end for
      for layer = L – 1 to 1 do          // back propagate L-1 node layers
        for each node j in layer do
        if fuzzy logic condition2 then j = D // as in Figure 5
            Δj = (∂g/∂Wk,j) * ∑i Wj,i * Δi
            for each node i in layer + 1 do
          if fuzzy logic condition3 then i = 1 //as in Figure 5
              Wj,i = Wj,i + α * aj *Δi
          if fuzzy logic condition3 then break
          end for
        if fuzzy logic condition2 then break
        end for
      end for
    if fuzzy logic condition1 then break
    end for
  until stop criterion
return Neural-Network
```

Note: Error = ∑j (yj – hw(xj)) where hw(x) is the output of the perceptron on the example, y is the calibrated true output and α is the learning rate. The fuzzy logic heuristic conditional statements have to be worked in such that if the condition is satisfied then the loop method is ignored and only one value is used. This results in a modified Dijkstra algorithmic approach.

6) Application of equation 13 to Figure 5 [4 (pp. 737 – 740)]

Since there is only one output i = 0, j = A to D, k = 1 to 8

$$S(t, V, T) = g\left[\sum_j W_{j,O}(V,T) \bullet g\left[s1(t) \bullet W1,j(V,T) + \ldots\ldots + s8(t) \bullet W8,j(V,T)\right]\right]$$

after summing over j this can be expressed as

$$\begin{aligned} S(t, V, T) = \; & g[W_{A,O}(V,T) \bullet g[s1(t) \bullet W1,A(V,T) + \ldots\ldots + s8(t) \bullet W8,A(V,T)] + \\ & W_{B,O}(V,T) \bullet g[s1(t) \bullet W1,B(V,T) + \ldots + s8(t) \bullet W8,B(V,T)] + \\ & W_{C,O}(V,T) \bullet g[s1(t) \bullet W1,C(V,T) + \ldots + s8(t) \bullet W8,C(V,T)] + \\ & W_{D,O}(V,T) \bullet g[s1(t) \bullet W1,D(V,T) + \ldots + s8(t) \bullet W8,D(V,T)]] \end{aligned}$$

Note: g is an activation function and not a multiplicative factor. So all the arguments inside the functions are summed parameter values that conform to the behavior of the function g.

References

The author thanks fellow coworker Arthur Townshend/General Dynamics Land Systems for constructive criticism of this paper.

[1] Vachtsevanos, G., Fault Diagnostics/Prognostics for Equipment Reliability and Health Maintenance, Seminar by Georgia Institute of Technology for Distance Learning and Professional Education, May 18-21, 2004 Atlanta, Ga., Section III Tools and Toolboxes

[2] Turban E., Aronson J., Liang T-P., Decision Support Systems and Intelligent Systems, Upper Saddle River, New Jersey: Prentice Hall, 7th ed. 2005, 604-623

[3] Levitin A., Introduction to The Design & Analysis of Algorithms, Boston: Addison Wesley, 2003

[4] Russell S., Norvig P., Artificial Intelligence A Modern Approach, Upper Saddle River, New Jersey: Prentice 2nd ed. 2003, 736-748

[5] Cormen T., Leiserson C., Rivest R., Stein C., ch. 26 Maximum Flow, Introduction to Algorithms, Boston, Massachusetts: McGraw-Hill 2nd ed. 2001, 664-669

[6] Weisstein, Eric W. "Sigmoid Function." From MathWorld—A Wolfram Web, Resource. http://mathworld.wolfram.com/SigmoidFunction.html

[7] Null L., Lobur J., The Essentials of Computer Organization and Architecture, Sudbury, Massachusetts: Jones and Bartlett Publishers 3rd ed. 2003, 438-441

[8] Author William Franklin's experience working in laser systems lab for many years