Chapter 9: Introduction to Fuzzy Logic

- Download CD Content

In this chapter, we'll look at the logic system defined by Lotfi Zadeh to overcome the disparity between real-world systems and software representations. In software, the Boolean logic values of true and false represent unique elements in a two-valued system. The real world rarely operates in this fashion—many conditions can be partially true or partially false (or both). Fuzzy logic was introduced to allow software to operate in the domain of degrees of truth. Instead of binary systems that represent truth or falsity, degrees of truth operate over the infinite space between 0.0 and 1.0 inclusive.

Introduction

To fully understand what Dr. Zadeh proposed in 1963, let's look at a number of examples that provide more information about fuzzy logic and its representation.

Fuzzy Quality of Service Example

The power of fuzzy logic appears when we begin to analyze a system using linguistics. For example, let's say that we're building a QoS (Quality of Service) algorithm that manages the output of packets over a particular link. The purpose of the QoS algorithm is to provide a given application a consistent amount of bandwidth of the link. If the application tries to use too much of the link's bandwidth, we must reduce the rate that packets are emitted for the given application. From a control perspective, we have three elements. The first is the packet arrival rate from the application, the second is the measured utilization of the link, and the third is the gate that controls the flow of packets between the application and the link (see Figure 9.1).

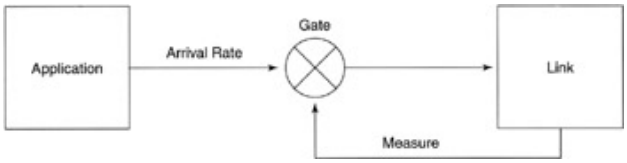

Figure 9.1: Quality of service scenario with rate feedback.

The purpose of the gate is to control when and how many packets are permitted to pass given the bandwidth allocated to the particular application. When we think about this problem, we consider it in terms of linguistics. For example:

> If the application utilization of the link is high, then reduce the rate of flow of packets through the gate for the application.

Conversely,

> If the application utilization of the link is low, then increase the rate of flow of packets through the gate for the application.

These rules imply that there exists a dead-zone between the high and low rates that is "about right" for the application

The question is now what is "high" and "low" for the application rate and what is "about right"? Fuzzy logic determines this using membership functions (see Figure 9.2).

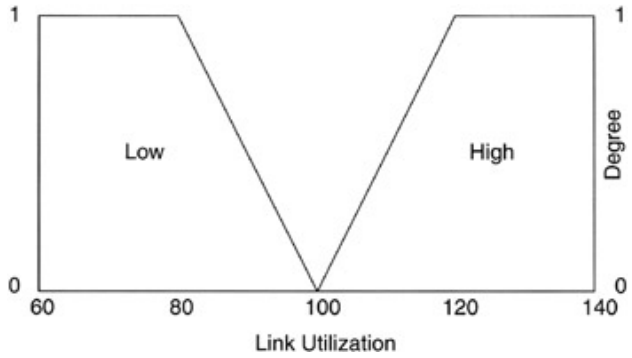

Figure 9.2: Membership function for packet rate.

Membership Functions

A membership function defines the degree of membership (degree of truth) of a given real-world value in the membership function. From our QoS example, we have a membership function over the set of link utilization (see Figure 9.2).

From Figure 9.2, we've defined two segments, with a third centered between what exists as our target. The low range can be defined programmatically as shown in Equation 9.1:

(9.1)
```
m_low(x) = {
      0,                    if rate(x) >= 100
      (100-rate(x))/20      if rate(x) > 80 AND rate(x) < 100
      1,                    if rate(x) <= 80.
}
```

The high range can then be defined as shown in Equation 9.2:

(9.2)
```
m_high(x) = {
      0,                    if rate(x) <= 100
      (rate(x)-100)/20      if rate(x) > 100 AND rate(x) < 120
      1,                    if rate(x) >= 120.
}
```

Fuzzy Control

To illustrate the membership functions, Table 9.1 identifies sample packet rates along with their degree of membership within the two functions.

Table 9.1: Example Degrees of Membership.

Packet Rate	*m_low(x)*	*m_high(x)*
10	1.0	0.0
85	0.75	0.0
90	0.5	0.0
95	0.25	0.0
100	0.0	0.0
105	0.0	0.25
110	0.0	0.5
115	0.0	0.75
180	0.0	1.0

So given these two membership functions, we can determine the degree of membership for a given rate of packet flow. However, how can this be used in terms of a control strategy for adjusting the packet flow to maintain a consistent quality of service? A very simple mechanism involves the degree of membership to be used as a coefficient to the gating algorithm.

Our gating function defines how many packets may pass during a given interval (for example one second). Each second, the number of packets that are permitted to pass is adjusted to maintain consistent utilization for the application. If the utilization is too high, we decrease the number of packets. Otherwise, if the utilization is too low, we increase it accordingly. The question is, how much to increase?

A simple mechanism is to use our degree of membership as the coefficient applied to the packet delta. Consider Equation 9.3.

(9.3) $$\text{rate} = \text{rate} + (\text{m\_low}(\text{rate}) * \text{pdelta}) - (\text{m\_high}(\text{rate}) * \text{pdelta})$$

The variable `rate` is the current number of packets that are permitted to pass through in the given time period (one second). Constant `pdelta` is a tunable parameter that defines the maximum number of packets that can be added to or subtracted from the rate. The membership functions then provide the degree of membership that is used as a coefficient in the rate-tuning algorithm.

Let's look at a few examples using a `pdelta` of 10. While our rate will represent a non-integer value, we'll round up for the tuning algorithm.

Given a `rate` of 110, how will our algorithm adjust the rate for the gating mechanism? Using our Equation 9.3, we get:

rate = 110 + (0 * 10) - (0.5 * 10) = 105

On the next iteration, if the application continues to generate at the given rate of 105, we'll get:

rate = 105 + (0 * 10) - (0.25 * 10) = 102.5

This process continues until we reach our dead-band of 100 at which point no further rate changes are performed.

If our rate had been low, such as 80, Equation 9.3 would have adjusted as follows:

$$rate = 80 + (1 * 10)-(0 * 10) = 90$$

Therefore, using the degree of membership as a coefficient for rate-change results in a very simple control mechanism.

A Visual Fuzzy Control Example

Let's now look at another example of fuzzy control using a tracking scenario. In a simple two-dimensional world, we have an agent that serves as the predator. Another agent in the world serves as the prey. The goal is to build a set of membership functions that provide the ability for the predator to track and ultimately catch the prey. Our membership functions will use the error angle for control (representing the difference between the predator's current direction and the direction of the prey).

A plot of the predator membership function is shown in Figure 9.3. Seven distinct groups are defined that identify how much error is present and thus how much correction should be applied.

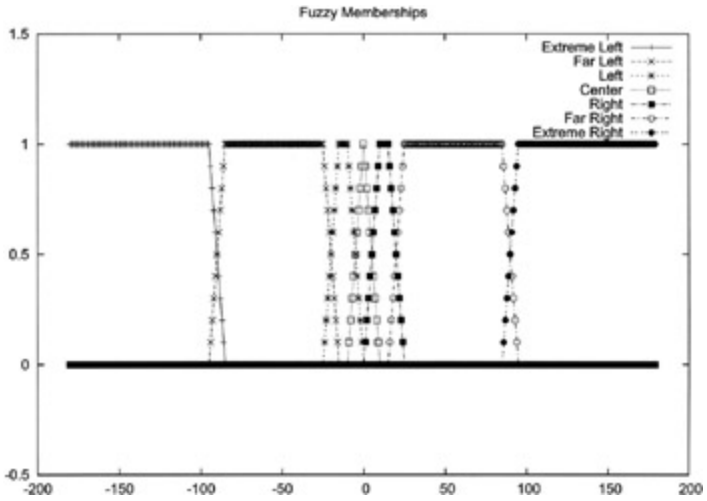

Figure 9.3: Predator membership functions.

The center group is the only group in which no change is made to the current direction of the predator. Left and right are [+1, -1] respectively, far left and far right are [+8, -8], and finally extreme left and extreme right represent [+15, -15]. If the predator's error angle falls into a particular membership function, the associated correction is applied to the predator's direction and the process begins again. At each time step, the predator moves one unit for each correction.

Given the membership functions in Figure 9.3, a plot of the predator in action is shown in Figure 9.4.

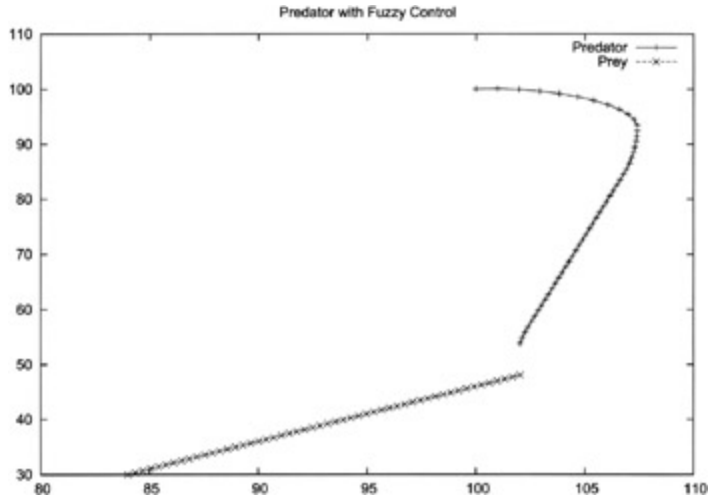

Figure 9.4: Predatory/prey example plot.

The predator begins at coordinates [100,100] where the prey begins at a random location (here, approximately [84,30]). The prey simply moves in a 45° line from its initial location. As shown, the predator correctly alters its course to intercept the prey (come within 5 units of the prey).

The Fuzzy Axioms

Just like conventional Boolean logic, fuzzy logic has a set of basic operators. These follow the conventional Boolean operators, but differ in how they work. The fuzzy axioms are shown in Figure 9.5.

Truth(A OR B)	MIN(truth(A), truth(B))
Truth(A AND B)	MAX(truth(A), truth(B))
Truth(NOT A)	1.0 - truth(A)

Figure 9.5: The fuzzy logic axioms.

These operators provide the basis for logic operations for the approximate fuzzy values. Performing conditions such as:

```
if (m_warm(board_temperature) AND
m_high(fan_speed)) then ...
```

This form is not only very readable, but can also better approximate the state of the system and lead to better control.

Hedge Functions

An important modifier for fuzzy systems is hedges, or modifiers of fuzzy memberships. These modifiers provide an additional linguistic construct to fuzzy logic and maintain mathematical consistency. Consider the hedge functions of `VERY` and `NOT_VERY`. Hedge functions are used in conjunction with membership functions and alter their value based upon the defined linguistic purpose. The hedge functions are defined below as Equations 9.4 and 9.5.

(9.4) $$VERY(m\_x) = m\_x^2$$

(9.5) $$NOT\_VERY(m\_x) = m\_x^{0.5}$$

For example, consider their use with the `m_high` membership function as defined by Figure 9.2. Given a packet rate of 115, `m_high` would result in 0.75. If we applied the VERY hedge function to the membership function (`VERY(m_high(rate))`), then the resulting value would be 0.5625 (in other words, it's high, but not very high). If the rate had been 119, `m_high` would result in 0.95. The VERY hedge function applied to this would then result in 0.903 (or, very high).

Why Use Fuzzy Logic?

Fuzzy logic is an important construct in the design of software as it uses approximations rather than exact values. This is important because human reasoning itself is approximate and therefore it is easier to map fuzzy logic to systems designs.

Sample Application

To demonstrate the fuzzy logic functions, we'll build a very simple battery charger simulation (which will lack some physical details).

Our mythical battery charger operates within an environment where at times a charging voltage exists (for example, from a set of solar cells), and a load exists. The voltage from the solar panels provides the ability to charge the battery while the load discharges the battery into the operating circuit. Our charger provides two modes of operation, trickle charge mode and fast charge mode. In trickle charge mode, only a small amount of current is permitted to pass into the battery to provide a very small amount of charge. In fast charge mode, all available current is provided to the charger for the onboard batteries.

From a control systems perspective, what is necessary is a way to determine when to go into fast charge mode and when to switch to trickle charge mode. When a battery charges, the temperature of the battery will rise. When the battery is fully charged, the additional current arriving at the battery will be realized as heat. Therefore, when the battery gets hot, it's probably a good indication that the battery is fully charged and to switch to trickle charge mode. We can also measure the voltage of our battery to determine if it has reached its limit, and then switch it into trickle charge mode. When the battery is neither hot nor has the battery reached its voltage limit, it's safe to switch to fast charge mode. These are simplified rules, as the derivative of the battery temperature is a better indication of battery charge.

Fuzzy Battery Charge Control

As defined, the battery charger has two modes of operation: trickle and fast charge modes. Two sensors exist to monitor the battery: voltage and temperature. To control the charge of the battery we'll use the following fuzzy rules:

```
if m_voltage_high( voltage )
then mode = trickle_charge
```

```
if m_temperature_hot( temperature )
then mode = trickle_charge

if ( ( not(m_voltage_high( voltage )) AND
     ( not(m_temperature_hot( temperature )) )
then mode = fast_charge
```

Note that these rules are sub-optimal, since the last rule could cover all cases. We'll use three rules to cover more of the available fuzzy operators.

Identifying the fuzzy rules is one of the first steps. This is readily known from system operators or through analysis of the problem. Defining how the linguistics map to the real world is the next problem. This is the process of creating the membership functions.

Fuzzy Charge Membership Functions

Creating the membership functions takes the fuzzy linguistic names and maps them to the real world values in the particular domain. In this example, we'll specify two variables: voltage and temperature. The voltage and temperature membership graphs (showing the actual membership functions) are shown in Figures 9.6 and 9.7 respectively.

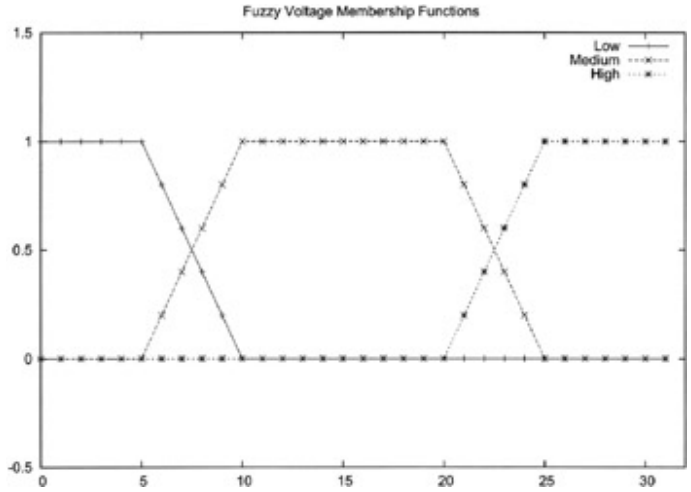

Figure 9.6: Fuzzy voltage membership graph.

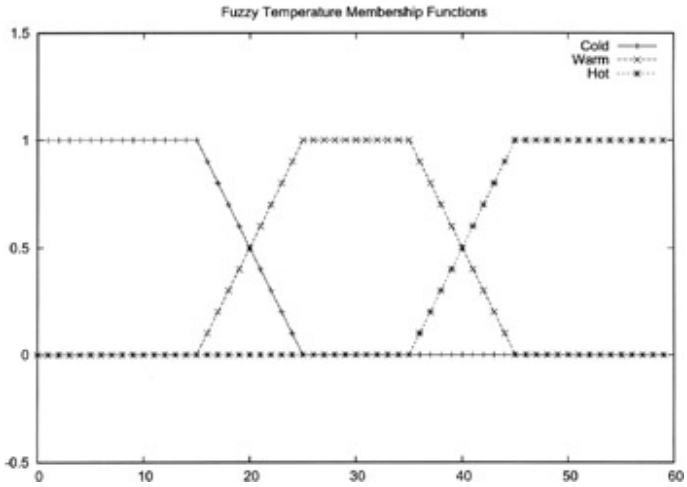

Figure 9.7: Fuzzy temperature membership graph.

The voltage membership graph defines three membership functions within the voltage domain: low, medium, and high. Similarly, the temperature domain also defines three: cold, warm, and hot. These artificial values are for demonstration purposes only and do not represent any type of battery technology.

Note One can easily see patterns in the membership functions. For this reason, a number of helper functions are provided in source to facilitate creating these membership shapes (including the spike pattern, not shown in these graphs).

On the CD The source code for the fuzzy logic battery charge simulator can be found on the CD-ROM at ./software/ch9/fuzzy/.

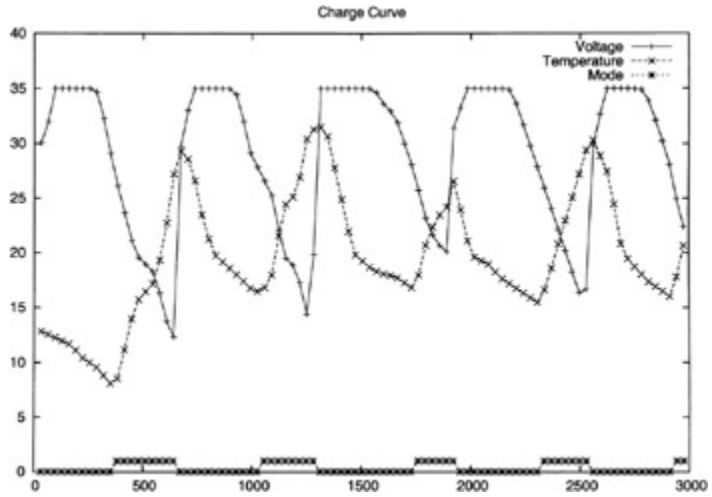

Figure 9.8: Charge curves for the battery charge control simulation.

While the simulation does not represent a true physical battery simulation, it models some of the basic concepts of loading and charging given the voltage and temperature constraints.

Advantages to Fuzzy Logic

There are a number of advantages to utilizing fuzzy logic in systems design. The fuzzy logic operators are simple like traditional Boolean logic operators. Therefore, they can be used by system operators to extend their knowledge of operation into the domain of membership functions and fuzzy logic rules, which linguistically are modeled by our own language.

For traditional systems developers, the complexity of the resulting system using fuzzy logic can be much simplified. Complex applications with a multitude of inputs and outputs can be modeled, and implemented, very simply using fuzzy logic.

Another interesting advantage is that fuzzy logic can reduce the processing requirements of a system and therefore decrease the expense of the resulting embedded control hardware. In many cases, complex mathematical modeling can be replaced with membership functions and a set of fuzzy rules to control a system. Minimizing these mathematical constraints can reduce code size and therefore allow the system to run more quickly, or on less advanced hardware.

Other Applications

Fuzzy logic has been used in a variety of widely varying applications. Its most visible use is in control systems where fuzzy systems have had commercial success. Fuzzy logic has also been applied to self-focusing cameras, cement mixing systems, vehicle controls including anti-lock breaking systems, and even rules-based systems.

Probably the most successful applications are the ones that remain unknown. The name fuzzy logic itself doesn't promote a sense of reliability, though technically we know it is a sound method. As with many other AI methods, there exists a slow migration into the practical realm where someday fuzzy logic will not be associated with AI but instead seen as a visible standard technical solution.

Summary

In this chapter, we've introduced fuzzy logic and fuzzy control. The fundamental fuzzy operators were defined using three very different applications (packet flow, target-tracking, and charge control). Hedge functions were also introduced as a linguistic modifier to existing fuzzy membership functions. A sample application of battery charge control was then detailed to identify how the membership functions and control elements were created using fuzzy semantics. Sample code was used to illustrate this concept, including a set of generic fuzzy logic APIs. Finally, some of the advantages of fuzzy logic were discussed including simplification and cost reduction of the hardware embodiment.

Resources

Aptronix, Inc. (1996). "Why Use Fuzzy Logic," available online at *http://www.aptronix.com/fide/whyfuzzy.htm*(accessed January 17, 2003)

Brule, James F. (1985). Fuzzy Systems—A Tutorial," available online eat *http://www.austinlinks.com/Fuzzy/tutorial.html*(accessed January 17, 2003)

Jantzen, Jan. (1998). "Tutorial on Fuzzy Logic." Technical Report no 98-E 868, University of Denmark, Department of Automation.

Zadeh, L.A. (1965). "Fuzzy sets." *Information and Control* 8:338–53.