Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

FUZZY MEASURES IN INDUCTIVE REASONING

DongHui Li
François E. Cellier

Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721

ABSTRACT

Inductive Reasoning is a technique which allows us to reason about a finite state representation of a system on the basis of available data. If the data stem from a continuous system, they are first discretized (recoded) into a finite set of discrete values. Recently, optimal recoding techniques have been devised which are presented in this paper. The forecasting power of the Inductive Reasoning approach has been shown to be dramatic in a number of examples. Yet, the forecast was always expressed in terms of the recoded, i.e. the discrete, variables, and not in terms of the original continuous variables. Recently, we have been working on a modification of the technique which allows us to reconstruct the continuous signals from the forecast discrete signals with very good accuracy. For this purpose, we exchanged the previously used probabilistic quality measures for fuzzy quality measures, and we predict, together with the discrete states also new fuzzy membership functions of the forecast signals. From these membership functions, we can then regenerate the continuous signals. The technique has been tested by means of a third order continuous–time linear system and has given promising results. These results are presented here.

1. INTRODUCTION

How do humans reasons about the behavior of physical systems? It is known that humans are frequently able to make correct assessments of the qualitative behavior of a given physical system without having a detailed knowledge or understanding as to the true physical mechanisms that govern the behavior of that system.

To this day, we have not been able to successfully code this ability into a computer algorithm such that we could e.g. equip robots aboard unmanned Space missions with it. Yet, since human presence in Space will, for a long time to come, be an expensive and risky business which is necessitated and justified only by the inadequacy of our automation technology, it is a noble goal to tackle this problem. This will, on the short run, enable us to reduce the need for manned Space missions, thereby reducing both the risks and the cost of Space exploration, and it may, in longer terms, help us develop a mature automation technology which can support human life in Space safely and at a reduced cost. At such time, human presence in Space will no longer be necessitated by an inadequate automation technology, but on the contrary, will be enabled by an adequate automation technology.

How can this problem be solved? Two schools of thought have emerged over the past couple of years which both claim to slowly get a handle at the problem.

The first school is known under a variety of names ranging from *Naïve Physics* to *Common Sense Reasoning*. It was triggered by Hayes [1979] in his "Naïve Physics Manifesto." An exposé of this methodology will be presented in the same session as this paper [Morgan 1990]. The idea behind the Naïve Physics approach is to decompose systems into their physical components, and then to reason about the behavior of these subsystems in rough and approximate terms. The major strength of this technique is that it allows us to incorporate whatever knowledge we possess about the system into the design process, i.e., it allows us to iteratively improve and refine our understanding of the system. Its major weaknesses are that it forces us to employ *a priori* knowledge from the beginning, and therefore, it cannot deal with completely unknown and unforeseen situations, and that so far the technique has only been demonstrated to work for very simple (almost trivial) systems.

The second school evolved from earlier *pattern recognition* techniques. Its proponents claim that our intuitive understanding of system's behavior does not base on physical knowledge at all, but rather derives from our ability to recognize similarities with earlier observed systems. When someone holds a glass full of water in his/her hands and opens his/her fingers, we *know* that the glass will fall down, break into many pieces, and spill the water all over the kitchen floor. How come that we know this fact? Is it because we know that there exists a gravitational force which will accelerate the glass downward once the constraint of the holding hand is removed, and then reason about the effect of impact which generates a reaction force too strong to be annihilated by the internal forces that hold the glass together? Or is it because we may remember that, when we were 10 years old, one of the adults in the family let a bottle of Chianti wine slip through his fingers, and our dog came, licked all the wine up from the floor, and was totally drunk for the rest of the day? If the latter assumption is true, then the next question must be how we can capture knowledge such that similarities are recognized quickly and reliably and can be exploited in our reasoning process.

Contrary to the Naïve Physics approach which is basically a *deductive* technique, the pattern recognition approaches are basically *inductive*. These techniques have produced convincing results also for fairly complex systems. However, one major weakness of these techniques is that they do not lend themselves easily to an improved understanding of what happens, i.e., while we may be able to make correct predictions, we often don't know exactly how we came to these conclusions and why they are correct, and it is therefore a difficult task to assess the correctness of a proposed prediction before the predicted event has actually occurred. A second weakness of these techniques is that they are purely inductive, i.e., they do not support very well the incorporation of *a priori* knowledge which may be at our disposal.

Among the pattern recognition techniques, two different classes have led to promising results. One is the *Neural Network* approach. Neural networks have been around for quite some time, but research in this area had been discredited in the late sixties as a consequence of the devastating report by Minsky and Papert [1969]. Only the last couple of years have seen a renaissance of this technique due to the pioneering research of a few individuals such as Grossberg [1982], Hopfield [1982], and Kohonen [1984]. Impressive results have meanwhile been obtained e.g. in the context of robot grasping systems [Ritter et al. 1990].

Neural Networks still suffer somewhat from their pattern recognition heritage. Classical pattern recognition techniques were always concerned primarily with the recognition of *static images*. For this reason, most research efforts in Neural Networks were also concentrated around the analysis of static information. For quite some time, it was not recognized that humans more often than not base their reasoning on *temporal patterns*, i.e., series of sketchy images (cartoons) making up an entire episode. The child who grew up in an abusive family will assume a defensive position whenever his/her vis–à–vis raises his/her hand since s/he identifies that *motion* with a seemingly similar earlier experienced motion that led to pain. The cartoon

of the raising hand within reach of the child's face is a temporal pattern which is identified with a stored temporal pattern of his/her past. The prediction is achieved through a playback of another temporal pattern that used to follow the identified pattern in time. Neural Networks are well suited to identify temporal patterns. All that needs to be done is to store the individual images below each other in a large array, and treat the entire cartoon as one pattern. Due to the inherent parallelism in Neural Network algorithms, the size of a network layer (the length of the pattern array) does not have to enlarge the time needed for its processing.

Only recently, Neural Network researchers have begun to investigate temporal patterns, e.g. in the context of speech recognition systems [Yuhas et al. 1989]. Other important papers in which Neural Networks are applied to the analysis of dynamical systems are e.g. a recent paper by Narendra and Parthasarathy [1990] and a still unpublished paper by Sorsa et al. [1990]. However, a lot remains to be done in this context.

The other pattern recognition technique that has shown promising results is the *Inductive Reasoning* approach which is largely due to Klir [1985]. Inductive Reasoners capture the knowledge about a system in the form of an *optimal mask* which describes the relationship between various variables of the system over time. "Similar" systems are characterized by the same optimal mask. Inductive Reasoners are particularly well suited to learn the dynamical behavior of a system from past observations of that system. They are able to determine an optimal forecast of the system's behavior in terms of discretized variables, i.e., they provide an optimal forecast of the system's *class behavior*. It is this technique which is being emphasized in our paper.

Comparing Inductive Reasoners with Neural Networks, it can be seen that the Inductive Reasoning approach is much more systematic than the Neural Network approach, and it is a little more straightforward to use Inductive Reasoners for behavior forecasting. However, Neural Networks can operate on a much larger number of input signals simultaneously, and they are therefore more general than Inductive Reasoners. Both techniques have been successfully applied to reasonably complex realistic system descriptions.

We have described the application of Inductive Reasoners to qualitative behavior forecasting in two previous publications. In [Cellier 1987], we described the application of the SAPS–II [Cellier and Yandell 1987] Inductive Reasoner to forecasting the behavior of a linear continuous–time system, and in [Vesanterä and Cellier 1989], we described the application of the same technique and software to the problem of fault diagnosis during a high altitude horizontal flight of a Boeing 747 airliner.

In the current paper, we shall tackle two formerly unsolved problems: (i) the problem of *optimal recoding*, i.e., the problem of an optimal selection of *landmarks* (the borderlines between neighboring domains in the discretization of continuous–time variables, and (ii) the problem of reconstructing the continuous variables following the class behavior forecast. We shall also propose a new complexity measure.

2. OPTIMAL RECODING

All qualitative modeling and simulation techniques rely on some sort of discretization of continuous input variables. The Naïve Physics approach usually relies on variables being discretized into the two domains positive (+) and negative (−) with a single landmark 0 separating the two domains. I.e., in Naïve Physics, continuous variables are recoded in an extremely crude manner. Neural Networks are able to process continuous input signals, but the Neural Network will not be able to migrate such signals very well through multiple network layers due to the sigmoid shape of the non–linear output function in each network layer. More practical, continuous signals are digitized and decomposed into series of binary signals. E.g., if an input signal is a voltage between $-1V$ and $+1V$, this signal could be decomposed into 256 levels (eight bits), and fed into the Neural Network in the form of eight parallel binary signals. Inductive Reasoners use a discretization scheme which is halfway between these two extremes. They can tolerate considerably more levels than the crude Naïve Physics algorithms, but they cannot process as many levels as a Neural Network since the Inductive Reasoning algorithms do not lend themselves so easily to parallelization, and since the optimization algorithms employed are usually exhaustive search algorithms.

In Inductive Reasoning, we shall allow more different values than just −, 0, and +, and we shall concentrate on the regions themselves rather than on the landmarks that separate these regions from each other. The values that represent such regions can be symbolic (e.g. *tiny*, *small*, *average*, and *big*, denoting four distinct regions), or they can be integer numbers (e.g. '1', '2', '3', and '4', denoting the same four regions as above). In an inductive reasoning system which is coded in LISP, symbolic names are probably preferred, whereas in an Inductive Reasoner coded in a predominantly numeric software, integers will be the representation of choice. From a practical point of view, it really doesn't matter which of the two representations is being used since one can easily be mapped into the other. The symbolic representation will make the code more readable though.

In Inductive Reasoning, the regions are called *levels*. (Notice the difference with System Dynamics. In System Dynamics, a "level" denotes a continuous state variable, whereas in Inductive Reasoning, it denotes one value of a discrete state variable.) The process of discretizing continuous trajectories into discrete episodes is called *recoding*. Finally, a combination of legal levels of all state variables of a model is called a *state*. Thus, a model with n state variables, each of which is recoded into k levels has k^n legal states. An *episodical behavior* is a time history of legal states. The "episodical behavior" is the qualitative counterpart of the quantitative "trajectory behavior".

We have developed an Inductive Reasoning software, called SAPS–II, which is available as either a CTRL–C library or a MATLAB toolbox [Cellier and Yandell 1987]. In SAPS–II, levels are represented through positive integers.

The first question that we must ask ourselves is: *How* do we recode? How many levels should we select for each of our state variables? Where do we draw the borderline (i.e., where do we select the landmark) that separates two neighboring regions from each other?

Inductive Reasoning is a completely inductive approach to modeling. It operates on a set of measured data points, and identifies a "model" from the previously made observations.

From statistical considerations, we know that, in any class analysis, we would like to record each possible discrete state at least five times [Law and Kelton 1982]. Thus, there exists a relation between the possible number of legal states, and the number of data points that we require to base our modeling effort upon:

$$n_{rec} \geq 5 \cdot n_{leg} = 5 \cdot \prod_{\forall i} k_i \tag{1}$$

where n_{rec} denotes the total number of recordings, i.e., the total number of observed states, n_{leg} denotes the total number of different legal states, i is an index that loops over all variables, and k is an index that loops over all levels. If we postulate that each variables assumes the same number of levels, eq(1) can be simplified to:

$$n_{rec} \geq 5 \cdot (n_{lev})^{n_{var}} \tag{2}$$

where n_{var} denotes the number of variables, and n_{lev} denotes the chosen number of levels for each variable. The number of variables is usually given, and the number of recordings is frequently predetermined. In such a case, we can find the optimum number of levels from eq(3):

$$n_{lev} = ROUND(\sqrt[n_{var}]{\frac{n_{rec}}{5}}) \tag{3}$$

For reasons of symmetry, we often prefer an odd number of levels over an even number of levels. E.g., the five levels *much too low*, *too low*, *normal*, *too high*, and *much too high* might denote states of the heart beat of a patient undergoing surgery. By choosing an odd number of levels, we can group anomalous levels symmetrically around the normal state.

If the number of recordings is not predetermined, we might consider consulting with a human expert (e.g., the surgeon) to determine a meaningful number of levels for a given variable.

The number of levels of our variables determines the expressiveness and the predictiveness of our qualitative model. The *expressiveness* of a qualitative model is a measure for the information content that the model provides. Later in this paper, we shall present you with formulae describing the information content of a qualitative model. The *predictiveness* of a qualitative model is a measure for its forecasting power, i.e., it determines the length of time over which the model can be used to forecast the future behavior of the underlying system.

If all variables are recoded into exactly one level, the qualitative model exhibits only one legal state. It is called a "null model". It will be able to predict the future behavior of the underlying system perfectly over an infinite time span (within the framework of its model resolution). Yet, the prediction does not tell us anything useful. Thus, the null model is characterized by an infinitely high predictiveness, and by a zero expressiveness.

On the other hand, if we recode every variable into 1000 levels, the system exhibits myriads of legal states. The expressiveness (i.e., resolution) of such a model will be excellent. Each state contains a large amount of valuable information about the real system. Yet, the predictiveness of this model will be lousy, unless we possess an extremely large base of observed data. In all likelihood, this model cannot be used to predict the behavior of the real system for even a single time step into the future.

Consequently, we must compromise. For most practical applications, we found that either three or five levels were about optimal [Cellier 1987; Vesanterä and Cellier 1989].

Once we decided upon the number of levels for each variable, we must choose the landmarks that separate neighboring regions. Often, this is best done by consulting with a human expert. E.g., we may ask the surgeon what he considers a *normal* heart beat during surgery, and when he would believe that the heart beat is definitely *too low* or *too high*, and when he would consider it to be *critically too low* or *critically too high*. If we are then able to predict the future behavior of the patient in terms of these qualitative variables, we may be able to construct a heart monitor which will warn the surgeon ahead of time about a predictable problem. This clearly sounds like a worthwhile research topic.

However, if the amount of observed data is limited, it may be preferable to maximize the expressiveness of the qualitative model. This demand leads to a clearly defined optimal landmark selection algorithm. The expressiveness of the model will be maximized if each level is observed equally often. Thus, one way to find an optimal set of landmarks, is to sort the observed trajectory values into e.g. ascending order, cut the sorted vector into n_{lev} segments of equal length, and choose the landmarks anywhere between the extreme values of neighboring segments. Let us demonstrate this process by means of an example. Figure 1 shows an observed trajectory of a continuous variable:

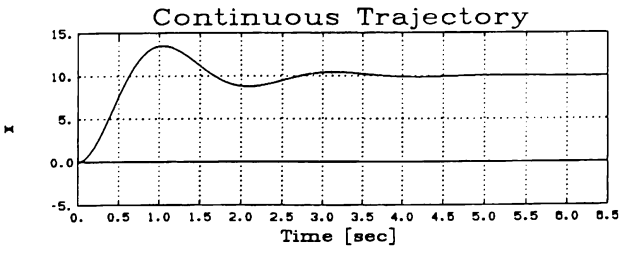

Figure 1. Trajectory behavior of a continuous variable

We first discretize the time axis (how this is done in an optimal manner, will be explained in due course). Let us say, that this process leads to a trajectory vector of length 131. The observed values range from 0.0 to 13.5. Let us assume that we wish to recode this trajectory into the three distinct levels '1', '2', and '3'. If we would simply cut the domain into equal intervals of length 4.5, i.e.:

$$\text{'1'} \longleftrightarrow [0.0, 4.5]$$
$$\text{'2'} \longleftrightarrow (4.5, 9.0]$$
$$\text{'3'} \longleftrightarrow (9.0, 13.5]$$

the levels '1' and '2' would only occur very briefly, and they would only occur during the initial phase of the episode. Thereafter, we would constantly observe a level of '3'. Figure 2 shows the recoded episode.

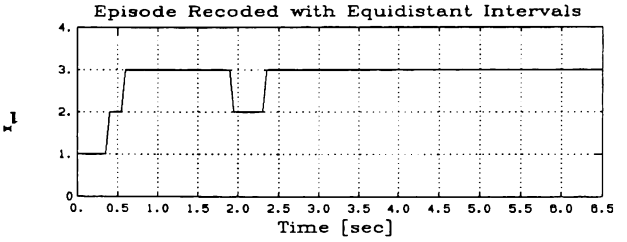

Figure 2. Episodical behavior of a recoded variable

In the process of recoding, we threw away lots of information. E.g., we lost most of the information regarding the oscillation frequency. However, if we use the above described optimal algorithm, we sort the trajectory values in ascending order such that the first value is 0.0, and the last value is 13.5. We then cut the resulting vector of length 131 into three vectors of approximately equal length. The first vector contains the elements 1 to 43, the second vector contains the elements 44 to 86, and the third vector contains the elements 87 to 131. For the given example, the following values were found:

$$x_{sorted}(43) = 9.8898$$
$$x_{sorted}(44) = 9.8969$$
$$x_{sorted}(86) = 10.0500$$
$$x_{sorted}(87) = 10.0501$$

and by using the arithmetic mean values of neighboring observed data points in different segments as our landmarks, we find:

$$LM_1 = 9.8934$$
$$LM_2 = 10.0500$$

Figure 3 shows the same continuous trajectory as Figure 1 with the two new landmarks superimposed:

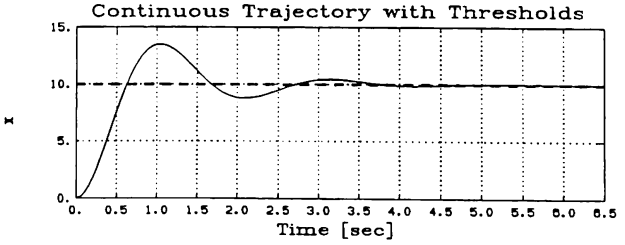

Figure 3. Trajectory behavior with landmarks superimposed

The bandwidth of level '2' is very narrow indeed. Figure 4 shows the recoded episodical behavior. Clearly, the recoding of Figure 4 has preserved more information about the real system than the recoding of Figure 2.

Which technique will work best depends heavily on the application area. For the case of the heart surgeon, the "optimized" recoding would be meaningless. His goal is to receive an early warning when the heart beat is expected to become critical, and not to observe each level equally often. Obviously, he wishes to observe level '3' (out of five levels) only, i.e., he wishes to keep his patient constantly within the *normal* range.

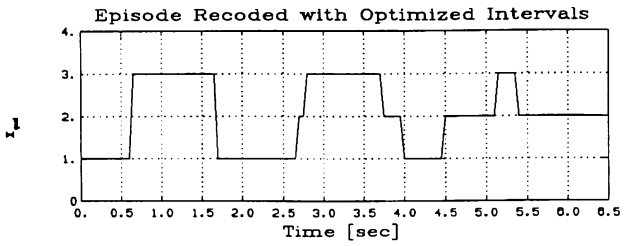

Figure 4. Episodical behavior of a recoded variable

The following example, which was taken from Hugo Uyttenhove's Ph.D. dissertation [1979], may serve as an illustration for the process of recoding. A heart monitor observes six different variables about a patient undergoing surgery. Each of these variables is being recoded into five different levels using the qualitative states:

'1' $\longleftrightarrow$ *much too low*
'2' $\longleftrightarrow$ *too low*
'3' $\longleftrightarrow$ *normal*
'4' $\longleftrightarrow$ *too high*
'5' $\longleftrightarrow$ *much too high*

Table 1 lists the six variables together with their five ranges.

Let us assume that the trajectory behavior of this six variable system has been recorded in the form of a trajectory behavior matrix *meas* with six columns denoting the six different variables, and 1001 rows denoting different measurement instants (different time values). The following SAPS–II program segment will recode these values:

```
DO saps : saps
from = [  0.0    75.0   100.0   150.0   180.0
         75.0   100.0   150.0   180.0   999.9];
to = 1 : 5;
raw = RECODE(meas(:, 1),' domain', from, to);
from = [  0.0    50.0    65.0   100.0   110.0
         50.0    65.0   100.0   110.0   999.9];
r = RECODE(meas(:, 2),' domain', from, to);
raw = [raw, r];
from = [  0.0     4.0    20.0
          4.0    20.0   999.9];
to = 2 : 4;
r = RECODE(meas(:, 3),' domain', from, to);
raw = [raw, r];
from = [  0.0     2.0     3.0     7.0
          2.0     3.0     7.0   999.9];
to = 1 : 4;
r = RECODE(meas(:, 4),' domain', from, to);
raw = [raw, r];
from = [  0.0    50.0    60.0   100.0   110.0
         50.0    60.0   100.0   110.0   999.9];
to = 1 : 5;
r = RECODE(meas(:, 5),' domain', from, to);
raw = [raw, r];
from = [  0.0     1.0     4.0    20.0
          1.0     4.0    20.0   999.9];
to = 1 : 4;
r = RECODE(meas(:, 6),' domain', from, to);
raw = [raw, r];
```

RECODE is one of the SAPS–II functions. As it is used in this example, it maps the regions (domains) specified in columns of the *from* matrix to the levels that are specified in the *to* vector. Thus, the *from* matrix and the *to* vector must have the same number of columns. In this example, each variable (column) is being recoded separately. The resulting episode r is then concatenated from the right to the previously found episodes which are stored in *raw*. The code should be fairly self–explanatory otherwise.

3. FUZZY RECODING

How big is big? Obviously, qualitative terms are somewhat subjective. In comparison with an adult, a 10 year old has usually quite a different opinion about what an "old person" is. The concept of landmarks is a treacherous one. Is it really true that a systolic blood pressure of 100.1 is "normal", whereas a systolic blood pressure of 99.9 is "too low"? Different physicians may have a different opinion altogether. One of the authors' wives has usually a systolic blood pressure of about 90. Yet, her physician always smiles when he measures her blood pressure and predicts that she will have a long life. 90 is too low for what? What the surgeon probably meant when he declared 100 the borderline between "normal" and "too low" was the following: If a "normal" patient, i.e. a patient with an average "normal" systolic blood pressure of 125 experiences a sudden drop of the blood pressure from 125 to 90 during surgery, then there is probably something wrong. Does this mean that we should look at the time derivative of the systolic blood pressure beside of the blood pressure itself? We probably should, but the surgeon couldn't tell us, because this is not the way he thinks. Most medical doctors aren't trained to think in terms of gradients and dynamical systems.

Zadeh [1985, 1986, 1987] tackled the uncertainty problem. He introduced *fuzzy measures* as a technique to deal with the uncertainty of landmarks. Instead of saying that the systolic blood pressure is "normal" for values above 100, and "too low" for values below 100, a fuzzy measure allows us to specify that, as we pass the value 100 in negative direction, the answer "normal" becomes less and less likely, while the answer "too low" becomes more and more likely.

In a graphical form, we can depict the fuzzy measure as follows:

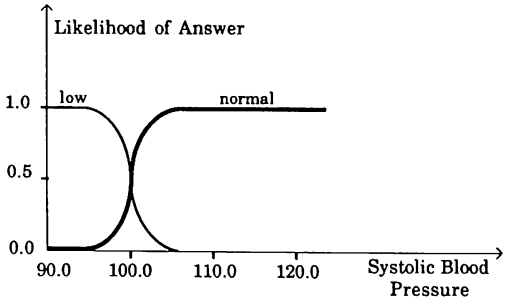

Figure 5. Fuzzy qualitative variable

Table 1. Recoding of heart variables

variable	'1'	'2'	'3'	'4'	'5'
Systolic Blood Pr.	< 75	$[75, 100)$	$[100, 150)$	$[150, 180)$	≥ 180
Mean Blood Pr.	< 50	$[50, 65)$	$[65, 100)$	$[100, 110)$	≥ 110
Central Venous Pr.		< 4	$[4, 20)$	≥ 20	
Cardiac Output	< 2	$[2, 3)$	$[3, 7)$	≥ 7	
Heart Rate	< 50	$[50, 60)$	$[60, 100)$	$[100, 110)$	≥ 110
Left Atrial Pr.	< 1	$[1, 4)$	$[4, 20)$	≥ 20	

The sigmoidal curves are called *membership functions*. How precisely these membership functions are shaped is up to the user. In SAPS–II, we have implemented only one type of membership function: a normal distribution which is 1.0 at the arithmetic mean value μ_i of any two neighboring "landmarks", and which is 0.5 at the landmarks themselves. This membership function can be easily calculated using the equation:

$$Memb_i = \exp(-k_i \cdot (x - \mu_i)^2) \tag{4}$$

where x is the continuous variable which needs to be recoded, say the systolic blood pressure, and k_i is determined such that the membership function $Memb_i$ degrades to a value of 0.5 at the neighboring landmarks. Figure 6 shows the membership functions for the systolic blood pressure:

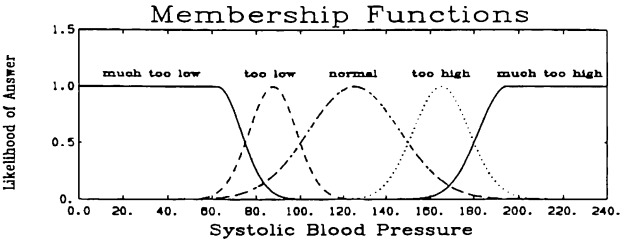

Figure 6. Membership functions of the systolic blood pressure

The first and the last membership functions are treated a little differently. Their shape (k_i value) is the same as for their immediate neighbors, and they are semi–open.

If we wish to compute the membership functions for the heart monitor example, we need to modify the previously shown program segment in the following way:

```
DO saps : saps
from = [  0.0    75.0   100.0   150.0   180.0
         75.0   100.0   150.0   180.0   999.9];
to = 1 : 5;
[raw, Memb, side] = RECODE(meas(:, 1),' fuzzy', from, to);
from = [  0.0    50.0    65.0   100.0   110.0
         50.0    65.0   100.0   110.0   999.9];
[r, m, s] = RECODE(meas(:, 2),' fuzzy', from, to);
raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
from = [  0.0     4.0    20.0
          4.0    20.0   999.9];
to = 2 : 4;
[r, m, s] = RECODE(meas(:, 3),' fuzzy', from, to);
raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
from = [  0.0     2.0     3.0     7.0
          2.0     3.0     7.0   999.9];
to = 1 : 4;
[r, m, s] = RECODE(meas(:, 4),' fuzzy', from, to);
raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
from = [  0.0    50.0    60.0   100.0   110.0
         50.0    60.0   100.0   110.0   999.9];
to = 1 : 5;
[r, m, s] = RECODE(meas(:, 5),' fuzzy', from, to);
raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
from = [  0.0     1.0     4.0    20.0
          1.0     4.0    20.0   999.9];
to = 1 : 4;
[r, m, s] = RECODE(meas(:, 6),' fuzzy', from, to);
raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
```

The *raw* matrix will be exactly the same as before (since *RECODE* will always pick the most likely answer), but in addition, we obtain the fuzzy memberships of all our qualitative variables which are stored in the *Memb* matrix. The third matrix *side* contains a value of 0 whenever the measured data point coincides with the mean value of the neighboring landmarks, it assumes a value of -1 if the measured variable is smaller than the mean between the landmarks, and it is $+1$ if the measured variable is larger than the mean between the landmarks.

In the process of recoding, a large amount of valuable information about our real system is usually discarded. The fuzzy membership retains some of this information which will prove useful in due course. In fact, up to this point, no information has been lost at all. The original continuous signal can be *regenerated* precisely using the SAPS function:

$$meas = REGENERATE(raw, Memb, side, from, to)$$

where the meaning of the *from* and *to* parameters is opposite from before.

4. INPUT/OUTPUT BEHAVIOR AND MASKING

By now, we have recoded our trajectory behavior into a discrete episodical behavior. In SAPS–II, the episodical behavior is stored in a *raw data matrix*. Each column of the raw data matrix represents one of the observed variables, and each row of the raw data matrix represents one time point, i.e., one recording of all variables, i.e., one recorded state. The values of the raw data matrix are in the set of legal levels that the variables can assume, i.e., they are all positive integers, usually in the range from '1' to '5'.

How does the episodical behavior help us identify a model of our system for the purpose of forecasting the future behavior of the system for any given input stream? Any model describes relationships between variables. That is its purpose. E.g., in a state–space model, we describe the relationships between the state variables x_i and their time derivatives $\dot{x}_i$:

$$\dot{x}_i = f_i(x_1, x_2, \ldots, x_n) \tag{5}$$

If the state variables x_i have been recoded into the qualitative variables v_i, and their time derivatives have been recoded into the qualitative variables w_i, we can write:

$$w_i = \tilde{f}_i(v_1, v_2, \ldots, v_n) \tag{6}$$

i.e., while $\tilde{f}_i$ will be a different function than f_i, the fact that there exist (deterministic) relationships between the x_i and the $\dot{x}_i$ variables, can be partially preserved in the process of recoding. The $\tilde{f}_i$ functions are (possibly deterministic) relationships between the v_i and the w_i variables.

The beauty of this transformation becomes evident when we try to identify these functional relationships. While the identification (characterization) of the f_i functions is a difficult task, the identification of the $\tilde{f}_i$ functions is straightforward. Since each of the v_i variables can assume only a finite set of values, we can characterize the $\tilde{f}_i$ functions through enumeration. Let us look at an example:

$$y = \sin(x) \tag{7}$$

is a quantitative relationship between two quantitative variables x and y. Let us recode the variable x into a qualitative variable v with four states, such that:

$$\begin{array}{cc} x & v \\ \left(\begin{array}{ll} 1^{st}\ quadrant & \text{'1'} \\ 2^{nd}\ quadrant & \text{'2'} \\ 3^{rd}\ quadrant & \text{'3'} \\ 4^{th}\ quadrant & \text{'4'} \end{array}\right) \end{array} \tag{8}$$

i.e., if the angle x is anywhere between 0^o and 90^o plus or minus a multiple of 360^o, the qualitative variable v assumes a value of '1', etc. v simply denotes the quadrant of x. Let us recode the variable y into a qualitative variable w with two states, such that:

$$\begin{array}{cc} y & w \\ \left(\begin{array}{ll} negative & \text{'1'} \\ positive & \text{'2'} \end{array}\right) \end{array} \tag{9}$$

i.e., w simply denotes the sign of y. This allows us to characterize the functional relationship between the two qualitative

variables v and w as follows:

$$\begin{array}{cc} v & w \\ \begin{pmatrix} \text{'1'} & \text{'1'} \\ \text{'2'} & \text{'1'} \\ \text{'3'} & \text{'2'} \\ \text{'4'} & \text{'2'} \end{pmatrix} \end{array} \tag{10}$$

Eq(10) is the qualitative counterpart of eq(7). Qualitative functions are *finite automata* which relate the qualitative variables to each other.

However, we have to be careful that we recode all variables in a consistent fashion. E.g., had we recoded x differently:

$$\begin{array}{cc} x & v \\ \begin{pmatrix} -45°..+45° & \text{'1'} \\ +45°..+135° & \text{'2'} \\ +135°..+225° & \text{'3'} \\ +225°..+315° & \text{'4'} \end{pmatrix} \end{array} \tag{11}$$

with the same recoding for y, we would have found a non–deterministic relationship between v and w:

$$\begin{array}{ccc} v & w & prob \\ \begin{pmatrix} \text{'1'} & \text{'1'} & 50\% \\ \text{'1'} & \text{'2'} & 50\% \\ \text{'2'} & \text{'1'} & 100\% \\ \text{'3'} & \text{'1'} & 50\% \\ \text{'3'} & \text{'2'} & 50\% \\ \text{'4'} & \text{'2'} & 100\% \end{pmatrix} \end{array} \tag{12}$$

The third column denotes the *relative frequency of observation* which can be interpreted as the *conditional probability* of the output w to assume a certain value, given that the input v has already assumed an observed value.

Eq(12) can be rewritten in a slightly different form:

$$\begin{array}{c|cc} {}_{v}\backslash^{w} & \text{'1'} & \text{'2'} \\ \text{'1'} & 0.5 & 0.5 \\ \text{'2'} & 1.0 & 0.0 \\ \text{'3'} & 0.5 & 0.5 \\ \text{'4'} & 0.0 & 1.0 \end{array} \tag{13}$$

which is called a *state transition matrix* relating v to w. The values stored in the state transition matrix are the transition probabilities between a given level of v and and certain level of w, i.e., they are the conditional probabilities of w given v:

$$ST_{ij} = p\{w = \text{'}j\text{'} | v = \text{'}i\text{'}\} \tag{14}$$

The element $< i, j >$ of the state transition matrix is the conditional probability of the variable w to become 'j', assuming that v has a value of 'i'.

We are not going to assume that we know anything about our system with the exception of the observed data streams, i.e., the measured trajectory behavior. Obviously, this says that we cannot know *a priori* what it means to recode our variables "consistently". All we know is the following: For any system which can be described by a deterministic (yet unknown) arbitrarily non–linear state–space model of arbitrary order, if we are lucky enough to pick all state variables and all state derivatives as our output variables, i.e., if all these variables are included in our trajectory behavior, and if we are fortunate enough to recode all these variables in a consistent fashion, then there will exist deterministic and static relationships between the qualitative state variables and the qualitative state derivatives.

In the process of modeling, we wish to find finite automata relations between our recoded variables which are *as deterministic as possible*. If we find such a relationship for every output variable, we can forecast the behavior of our system by iterating through the state transition matrices. The more deterministic the state transition matrices are, the better will be the certainty that we predict the future behavior correctly.

Let us now look at the development of our system over time. For the moment, we shall assume that our observed trajectory behavior was produced by quantitatively simulating a state–space model over time. Let us assume that the state–space model was coded in a CSSL language, that we integrate it using a fixed step forward Euler algorithm, and that we log every time step in our trajectory behavior. In this case, we can write:

$$x_i(k+1) = x_i(k) + \Delta t \cdot \dot{x}_i(k) \tag{15}$$

The qualitative version of eq(15) is:

$$v_i(k+1) = \tilde{g}(v_i(k), w_i(k)) \tag{16}$$

Eq(15) is obviously a deterministic relationship between $x_i(k)$, $\dot{x}_i(k)$, and $x_i(k+1)$. Is $\tilde{g}$ a deterministic function? Let us assume that we choose our step size Δt very small in order to integrate accurately. In this case, the state variables will change very little from one step to the next. This means that, after the recoding, $v_i(k+1)$ is almost always equal to $v_i(k)$. Only when x_i passes through a landmark will $v_i(k+1)$ be different from $v_i(k)$. Consequently, our qualitative model will have a tough time predicting when the landmark crossing will take place. Thus, it is not a good idea to include every tiny time step in our trajectory behavior. The time distance between two logged entries of our trajectory behavior δt should be chosen such that the two terms in eq(15) are of the same order of magnitude, i.e.:

$$\|v_i\| \approx \delta t \cdot \|w_i\| \tag{17}$$

Notice that δt is the *communication interval*, whereas Δt denotes the *integration step size*. These two variables have no direct relationship with each other. The data stream could as well be the output of a digital oscilloscope observing a real physical system. In such a case, Δt has no meaning, but δt still exists, and must be chosen carefully.

Let us now forget about state–space models. All we know is that we have a recorded continuous trajectory behavior available for modeling. We want to assume furthermore that we know which are inputs into the real system, and which are the outputs that we measure. Our trajectory behavior can thus be separated into a set of input trajectories concatenated from the right with a set of output trajectories, e.g.:

$$\begin{array}{lccccc} time & u_1 & u_2 & y_1 & y_2 & y_3 \\ \begin{matrix} 0.0 \\ \delta t \\ 2\cdot\delta t \\ 3\cdot\delta t \\ \vdots \\ (n_{rec}-1)\cdot\delta t \end{matrix} & \begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \end{array} \tag{18}$$

The trajectory behavior is recoded into an episodical behavior using the techniques described in the last section. Our modeling effort now consists in finding finite automata relations between the recoded variables which make the resulting state transition matrices as deterministic as possible. Such a relation could look like:

$$y_1(t) = \tilde{f}(y_3(t-2\delta t), u_2(t-\delta t), y_1(t-\delta t), u_1(t)) \tag{19}$$

Eq(19) can be represented as follows:

$$\begin{array}{c|ccccc} {}_{t}\backslash^{x} & u_1 & u_2 & y_1 & y_2 & y_3 \\ t-2\delta t & 0 & 0 & 0 & 0 & -1 \\ t-\delta t & 0 & -2 & -3 & 0 & 0 \\ t & -4 & 0 & +1 & 0 & 0 \end{array} \tag{20}$$

The negative elements in the above matrix denote inputs of our qualitative functional relationship. The above example has four inputs. The sequence in which they are enumerated is immaterial. We usually enumerate them from the left to the right, and from the top to the bottom. The positive value is the output. Thus, eq(20) is a matrix representation of eq(19). In Inductive Reasoning, such a representation is called a *mask*. A mask

denotes a dynamic relationship between qualitative variables. In SAPS–II, masks are written as either MATLAB or CTRL–C matrices. A mask has the same number of columns as the episodical behavior to which it should be applied, and it has a certain number of rows. The number of rows of the mask matrix is called the *depth* of the mask. The mask can be used to flatten a dynamic relationship out into a static relationship. We can shift the mask over the episodical behavior, pick out the selected inputs and outputs, and write them together in one row. Figure 7 illustrates this process.

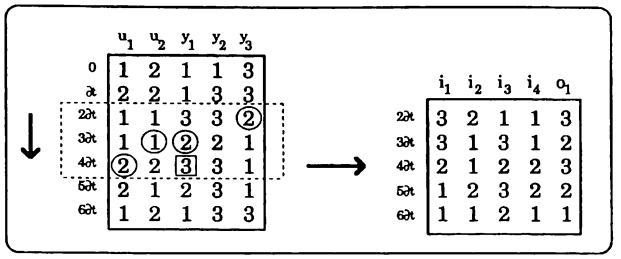

Figure 7. Flattening dynamic relationships through masking

After the mask has been applied to the raw data, the formerly dynamic episodical behavior has become static, i.e., the relationships are now contained within single rows. in SAPS–II, this operation can be performed using the *IOMODEL* function:

$$io = IOMODEL(raw, mask)$$

IOMODEL will translate the raw data matrix on the left side of Figure 7 into the flattened data matrix on the right side of Figure 7.

We still haven't discussed how δt is picked in practice. Experience has shown that the equality of eq(17) can be translated into the following general rule: The mask should cover the largest time constant that we wish to capture in our model. If the trajectory behavior stems from measurement data, we should measure a Bode diagram of the system that we wish to model. This allows us to determine the band width ω_{3dB} of the system. The largest time constant (i.e., the settling time) of the system can be computed from eq(21):

$$t_s \approx \frac{2\pi}{\omega_{3dB}} \tag{21}$$

If our chosen mask depth is 3, the mask spans a time interval of $2\delta t$, thus:

$$\delta t = \frac{t_s}{2} \tag{22}$$

The mask depth should be chosen as the ratio between the largest and the smallest time constant that we wish to capture in our model, but this ratio should better not be larger than 3 or 4. Otherwise, the Inductive Reasoner won't work very well.

5. OPTIMAL MASKS

An Inductive Reasoning model is simply a set of masks that relate the input variables and previous values of the output variables to the current values of the outputs. We shall usually forbid relations between various outputs at the current time, since if:

$$y_1(t) = \tilde{f}_1(y_2(t)) \tag{23a}$$
$$y_2(t) = \tilde{f}_2(y_1(t)) \tag{23b}$$

we got an *algebraic loop.*

The question remains: How do we find the appropriate masks. The answer to this question was already given. We need to find the masks that, within the framework of the allowable masks, presents us with the most deterministic state transition matrix since this matrix will optimize the predictiveness of our model. In SAPS–II, we have introduced the concept of a *mask candidate matrix.* A mask candidate matrix is en ensemble of all possible masks from which we choose the best one by a mechanism of *exhaustive search.* The mask candidate matrix contains -1 elements where the mask has a *potential input,* it contains a $+1$ element, where the mask has its output, and it contains 0 elements to denote forbidden connections. Thus, the mask candidate matrix for our previous five variable example will look as follows:

$$\begin{array}{c} t \backslash x \\ t-2\delta t \\ t-\delta t \\ t \end{array} \begin{array}{c} \begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & 0 & 0 \end{pmatrix} \end{array} \tag{24}$$

The SAPS–II program segment:

```
mcan = -ONES(3,5);
mcan(3,3:5) = [1,0,0];
maxcompl = 5;
mask = OPTMASK(raw,mcan,maxcompl)
```

determines the optimal mask from the set of candidate masks. *raw* is the raw data matrix, and *mcan* is the mask candidate matrix. *OPTMASK* will go through all possible masks of complexity two, i.e., all masks with one input, and find the best. It will then proceed and try all masks of complexity three, i.e., all masks with two inputs, and find the best of those, etc. The third parameter *maxcompl* enables us to limit the maximum complexity, i.e., the largest number of non–zero elements that the mask may contain. This is a useful feature. In all practical examples, the quality of the masks will first grow with increasing complexity, then reach a maximum, and then decay rapidly. Thus, by setting *maxcompl*, we can reduce the time that the optimization takes. A good value for *maxcompl* is usually five. If you wish to disable this parameter, set *maxcompl* to zero.

How do we determine the quality of a mask? Let us explain the process by means of a simple example. Let us assume that we have found the following raw data matrix (episodical trajectory) of a three variable system:

```
raw = [ 2 1 1
        1 2 3
        1 1 4
        1 1 3
        1 1 2
        2 2 2
        2 1 2
        2 2 3
        1 2 1
        1 2 1
        2 1 4 ]
```

Each mask will lead to a different state transition matrix. E.g., the mask:

```
mask = [ -1  -2   0
          0   0  +1 ]
```

will lead to the following input/output model:

```
io = [ 2 1 3
       1 2 4
       1 1 3
       1 1 2
       1 1 2
       2 2 2
       2 1 3
       2 2 1
       1 2 1
       1 2 4 ]
```

The *basic behavior* of this input/output matrix is a lexical listing of all observed states together with their observation frequencies:

$$b = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} \qquad p = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \\ 0.2 \\ 0.2 \\ 0.1 \\ 0.1 \end{bmatrix}$$

In SAPS–II, the basic behavior can be computed using the statement:

$$[b, p] = BEHAVIOR(io)$$

This gives rise to the following state transition matrix:

in \ out	'1'	'2'	'3'	'4'
'11'	0.000	0.667	0.333	0.000
'12'	0.333	0.000	0.000	0.667
'21'	0.000	0.000	1.000	0.000
'22'	0.500	0.500	0.000	0.000

which shows on the left side the combined values of the two inputs, in the top row the values of the output, and in the table itself, the conditional probabilities. The two inputs are binary variables, whereas the output has four levels. In addition, we need the absolute probabilities (observation frequencies) of the input states. For our example, the following values are found:

input	*prob*
'11'	0.3
'12'	0.3
'21'	0.2
'22'	0.2

In SAPS–II, the state transition matrix and the input probability vector can be computed using the statement:

$$[st, ip] = STMATRIX(io, 2)$$

where the second input argument denotes the number of input variables of the input/output matrix. In our example, the input/output matrix contains two inputs and one output.

Now, we can compute the Shannon entropy [Shannon and Weaver 1964] of the state transition matrix which is a measure of the information content of the state transition matrix. The Shannon entropy is computed with the following formulae:

$$HM = -\sum_{\forall i}(p\{inp = 'i'\} \cdot \sum_{\forall j}(p\{out = 'j'|inp = 'i'\} \cdot \log_2(p\{out = 'j'|inp = 'i'\}))) \tag{25}$$

In our example:

$$\begin{aligned} -HM &= 0.3 \cdot [0.667 \cdot \log_2(0.667) + 0.333 \cdot \log_2(0.333)] \\ &\quad + 0.3 \cdot [0.667 \cdot \log_2(0.667) + 0.333 \cdot \log_2(0.333)] \\ &\quad + 0.2 \cdot [1.0 \cdot \log_2(1.0)] \\ &\quad + 0.2 \cdot [0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)] \\ &= -0.275 - 0.275 + 0.0 - 0.2 \\ &= -0.75 \end{aligned}$$

and thus:

$$HM = 0.75$$

The state transition matrix is completely deterministic if it contains one +1 element in every row, while all other elements are 0. In that case, the Shannon entropy is:

$$HM_{min} = 0.0$$

The worst case occurs if all outcomes are equally probably, i.e., if the state transition matrix contains only elements of the same magnitude, in our case: 0.25 (since the output has four levels). For this case, we find the following Shannon entropy:

$$HM_{max} = 2.0$$

The maximum entropy depends on the number of rows and columns of the state transition matrix. We can introduce an *uncertainty reduction measure* which is defined as follows:

$$HR = 1.0 - \frac{HM}{HM_{max}} \tag{26}$$

In SAPS–II, the Shannon entropy and the uncertainty reduction measure of a state transition matrix can be determined using the statement:

$$[HM, HR] = ENTROPY(st, ip)$$

HR can be used as a *quality measure*. In the worst case, *HR* is equal to 0.0, while in the best case, *HR* is equal to 1.0.

However, there is still a problem with this approach. If we increase the complexity of the mask, we find that the state transition matrix becomes more and more deterministic. With growing mask complexity, there exist more and more possible input states (combinations of levels of the various input variables). Since the total number of observations n_{rec} remains constant, the observation frequency of the observed states will become smaller and smaller. Very soon, we shall be confronted with the situation where every state that has ever been observed has been observed precisely once. This leads obviously to a completely deterministic state transition matrix. Yet, the predictiveness of the model may still be very poor, since already the next predicted state has probably never before been observed, and that will be the end of our forecasting. Therefore, we must include this consideration in our quality measure.

We had explained previously that, from a statistical point of view, we would like to ensure that every state was observed at least five times. Therefore, we introduce an *observation ratio*:

$$OR = \frac{5 \cdot n_{5\times} + 4 \cdot n_{4\times} + 3 \cdot n_{3\times} + 2 \cdot n_{2\times} + n_{1\times}}{5 \cdot n_{leg}} \tag{27}$$

where:

n_{leg} = number of legal input states
$n_{1\times}$ = number of input states observed only once
$n_{2\times}$ = number of input states observed twice
$n_{3\times}$ = number of input states observed thrice
$n_{4\times}$ = number of input states observed four times
$n_{5\times}$ = number of input states observed five times or more

If every legal input state has been observed at least five times, *OR* is equal to 1.0. If no input state has been observed at all (no data), *OR* is equal to 0.0. Thus, also *OR* qualifies for a quality measure.

We define the *quality of a mask* as the product of its uncertainty reduction measure and its observation ratio:

$$Q = HR \cdot OR \tag{28}$$

The *optimal mask* is the mask with the largest Q value.

The *OPTMASK* function can be used to compute all these quantities. The full syntax of this function is as follows:

$$[mask, HM, HR, Q, mhis] = OPTMASK(raw, mcan, maxcompl)$$

mask is the optimal mask found in the optimization. *HM* is a row vector that contains the Shannon entropies of the best masks for every considered complexity. *HR* is a row vector that contains the corresponding uncertainty reduction measures. *Q* is a row vector which contains the corresponding quality measures, and *mhis* is the mask history matrix. The mask history

matrix contains, concatenated to each other from the right, the best masks at each of the considered complexities. One of these masks is the optimal mask which, for reasons of convenience, is also returned separately.

Until now, we haven't used our fuzzy membership functions yet. Remember that the fuzzy membership associated with the value of a qualitative variable is a *measure of confidence*. It specifies how confident we are that the assigned value is correct. If we compute the input/output matrix, we can assign a *confidence* to each row. The confidence of a row of the input/output matrix is the *joint membership* of all the variables which are associated with that row.

Let us demonstrate the concept by means of our simple three variable example. Assume that the following fuzzy membership matrix accompanies our raw data matrix:

$$raw = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \\ 2 & 1 & 4 \end{bmatrix} \qquad Memb = \begin{bmatrix} 0.61 & 1.00 & 0.83 \\ 0.73 & 0.77 & 0.95 \\ 0.73 & 0.88 & 1.00 \\ 0.51 & 0.91 & 1.00 \\ 0.55 & 0.92 & 0.92 \\ 0.71 & 0.77 & 0.78 \\ 0.63 & 0.91 & 0.69 \\ 0.86 & 0.83 & 0.83 \\ 0.77 & 0.97 & 0.70 \\ 0.78 & 0.93 & 0.75 \\ 0.89 & 0.81 & 1.00 \end{bmatrix}$$

The joint membership of i membership functions is defined as the smallest individual membership:

$$Memb_{joint} = \bigcap_{\forall i} Memb_i = \inf_{\forall i}(Memb_i) \stackrel{def}{=} \min_{\forall i}(Memb_i) \qquad (29)$$

The *FIOMODEL* function of SAPS–II computes the input/output matrix together with the confidence vector:

$$[io, conf] = FIOMODEL(raw, mask)$$

Applied to our example, we find:

$$io = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \end{bmatrix} \qquad conf = \begin{bmatrix} 0.61 \\ 0.73 \\ 0.73 \\ 0.51 \\ 0.55 \\ 0.69 \\ 0.63 \\ 0.70 \\ 0.75 \\ 0.78 \end{bmatrix}$$

The *conf* vector indicates how much confidence we have in the individual rows of our input/output matrix. We can now compute the basic behavior of the input/output model. Rather than counting the observation frequency, we shall accumulate the confidence. If a state has been observed more than once, we gain more and more confidence in it. Thus, we sum up the individual confidences. In SAPS–II, this can be achieved using the statement:

$$[b, c] = FBEHAVIOR(io, conf)$$

Applied to our simple example, we find:

$$b = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} \qquad c = \begin{bmatrix} 1.06 \\ 0.73 \\ 0.75 \\ 1.51 \\ 1.24 \\ 0.70 \\ 0.69 \end{bmatrix}$$

Notice that the c vector is no longer a probability. The c_i elements no longer add up to 1.0.

This leads now to a modified state transition matrix. The SAPS–II statement:

$$[st, ic] = FSTMATRIX(io, conf, 2)$$

produces the following results:

$_{in}\backslash^{out}$	'1'	'2'	'3'	'4'
'11'	0.00	1.06	0.73	0.00
'12'	0.75	0.00	0.00	1.51
'21'	0.00	0.00	1.24	0.00
'22'	0.70	0.69	0.00	0.00

which shows on the left side the combined values of the two inputs, in the top row the values of the output, and in the table itself, the confidence values. The *total input confidence* vector is:

input	*conf*
'11'	1.79
'12'	2.26
'21'	1.24
'22'	1.39

The total input confidences are computed by summing up the individual confidences of all occurrences in the basic behavior. Notice that in all these computations, the actual qualitative variables are exactly the same as before, only their assessment has changed. The previously used *probability measure* has been replaced by a *fuzzy measure*.

The optimal mask analysis can use the fuzzy measure as well. The statement:

$$[mask, HM, HR, Q, mhis] = FOPTMASK(raw, Memb, mcan, maxcompl)$$

uses the fuzzy measure to evaluate the optimal masks. In order to be able to still use the Shannon entropy, we normalize the row sums of the state transition matrices to 1.0. It can happen that *FOPTMASK* picks another mask as its optimal mask than the previously used *OPTMASK* routine. Due to the fact that we use more information about the real system, we will in most cases obtain a higher mask quality. Notice that the concept of applying the Shannon entropy to a confidence measure is somewhat dubious on theoretical grounds since the Shannon entropy was derived in the context of probabilistic measures only. For this reason, some scientists prefer to replace the Shannon entropy by other types of performance indices [Klir 1989; Shafer 1976] which have been derived in the context of the particular measure chosen. However, from a practical point of view, numerous simulation experiments performed by us have shown the Shannon entropy to work satisfactorily also in this context.

6. FORECASTING BEHAVIOR

Once we have determined the optimal mask, we can compute the input/output model resulting from applying the optimal mask to the raw data, and we can compute the corresponding state transition matrix. Now, we are ready to forecast the future behavior of our system, i.e., we are ready to perform a *qualitative simulation*.

Forecasting is a straightforward procedure. We simply loop over input states in our input/output model, and forecast new output states by reading out from the state transition matrix the most probable output given the current input. Let us explain this procedure by means of the previously used example. Given the raw data matrix:

$$raw = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \\ 2 & 1 & 4 \end{bmatrix}$$

which is assumed to consist of two input variables and one output variable. The future inputs over the next four steps are:

$$inp = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 2 & 2 \\ 2 & 1 \end{bmatrix}$$

and we wish to forecast the output vector over the same four steps. Let us assume furthermore that the optimal mask is the one used earlier:

$$mask = \begin{bmatrix} -1 & -2 & 0 \\ 0 & 0 & +1 \end{bmatrix}$$

For this case, we have already computed the input/output model and the state transition matrix. After the input/output model has been computed, the mask covers the final two rows of the raw data matrix. In order to predict the next output, we simply shift the mask one row further down. The next input set is thus: '21'. From the state transition matrix, we find that this input leads in all cases to the output '3'. Thus, we copy '3' into the data matrix at the place of the next output. We then shift the mask one row further down. At this time, the mask reads the input set '11'. From the state transition matrix, we find that the most probable output is '2', but its probability is only 66.7%. We continue in the same manner. The next input set is again '11'. Since this input set is assumed to be *statistically independent* of the previous one (an unreasonable but commonly made assumption), the joint probability is the product of the previous cumulative probability with the newly found probability, thus $p = \frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9} = 44.4\%$. The next input set is '22'. For this case, we find that the outcomes '1' and '2' are equally likely (50%). Thus, we pick arbitrarily one of those. The cumulative probability has meanwhile decreased to 22.2%.

This is exactly how, in SAPS–II, the FORECAST routine predicts future states of a recoded system.

$$[f2, p] = FORECAST(f1, mask, nrec, minprob)$$

forecasts the future behavior of a given system $f1$ where $f1$ contains the raw data model concatenated from below with the future inputs filled from the right with arbitrary zero values, thus:

$$f1 = [raw; inp, ZROW(nstp, nout)]$$

where *nstp* denotes the number of steps to be forecast, and *nout* denotes the number of output variables in the raw data model. *mask* is the optimal mask to be used in the forecasting, *nrec* denotes the number of recorded past data values, i.e., *nrec* tells the forecasting routine how many of the rows of $f1$ belong to the past, and how many belong to the future, and *minprob* instructs SAPS to terminate the forecasting process if the cumulative probability decreases below a given value. This feature can be disabled by setting *minprob* to zero.

Upon return, $f2$ contains the same information as $f1$ but augmented by the forecast outputs, i.e., some or all of the ZROW values have been replaced by forecasts. p is a column vector containing the cumulative probabilities. Of course, up to row *nrec*, the probabilities are all 1.0 since these rows contain past, i.e. factual, information.

If, during the forecasting process, an input state is encountered which has never before been recorded, the forecasting process comes to a halt. It is then the user's responsibility to either collect more data, reduce the number of levels, or pick an arbitrary output and continue with the forecasting.

How can we make use of the fuzzy memberships in the forecasting process? The procedure is very similar. However, in this case, we don't pick the output with the highest confidence. Instead, we compare the membership and side functions of the new input with the membership and side functions of all previous recordings of the same input, and pick as the output the one that belongs to the previously recorded input with the most similar membership. For this purpose, we compute a cheap approximation of the regenerated continuous signal:

$$d = 1 + side * (1 - Memb) \tag{30}$$

for every input variable of the new input set, and store the regenerated d_i values in a vector. We then repeat this reconstruction for all previous recordings of the same input set. We finally compute the L_2–norms of the difference between the d vector of the new input and the d vectors of all previous recordings of the same input, and pick the one with the smallest L_2–norm. Its *output* and *side* values are then used as forecasts for the *output* and *side* values of the current state. We proceed a little differently with the membership values. Here, we take the five previous recordings with the smallest L_2–norms, and compute a distance weighted average as the forecast for the fuzzy membership values of the current state.

In SAPS–II, this is accomplished by use of the function:

$$[f2, Memb2, side2] = FFORECAST(f1, Memb1, side1, mask, nrec)$$

The fuzzy forecasting function will usually give us a more accurate forecast than the probabilistic forecasting function. Also, if we use fuzzy forecasting, we can afterwards regenerate pseudo–continuous output signals with a relatively high quality using the REGENERATE function.

7. A LINEAR SYSTEM – AN EXAMPLE

Let us analyze once more the same third order example that we discussed in [Cellier 1987]:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -3 & -4 \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot u \end{aligned} \tag{31a}$$

$$\begin{aligned} y &= \mathbf{C} \cdot \mathbf{x} + \mathbf{d} \cdot u \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot u \end{aligned} \tag{31b}$$

We can easily determine that the band width of this system is $\omega_{3dB} \approx 1 sec^{-1}$. Therefore, the settling time is $t_s \approx 6 sec$. We wish to use a mask with a depth of three, and therefore, the communication interval should be $\delta t \approx 3 sec$.

In order to exert all frequencies of this system in an optimal manner, we shall simulate this system (directly in CTRL–C) applying a *binary random sequence* as the input signal [Cellier 1987]. We decided to recode each of the output states into three levels (the input is already binary), and therefore, the number of legal states can be computed as:

$$n_{leg} = 2 \cdot 3 \cdot 3 \cdot 3 = 54 \tag{32}$$

and therefore, the required number of recordings is:

$$n_{rec} = 5 \cdot n_{leg} = 270 \tag{33}$$

Let us simulate the system over 300 communication intervals. This is accomplished as follows:

```
t = 0 : 3 : 900;
u = ROUND(RAND(t));
x0 = ZROW(3, 1);
SIMU('ic', x0);
y = SIMU(a, b, c, d, u, t);
```

Figure 8 shows the results of the continuous–time simulation.

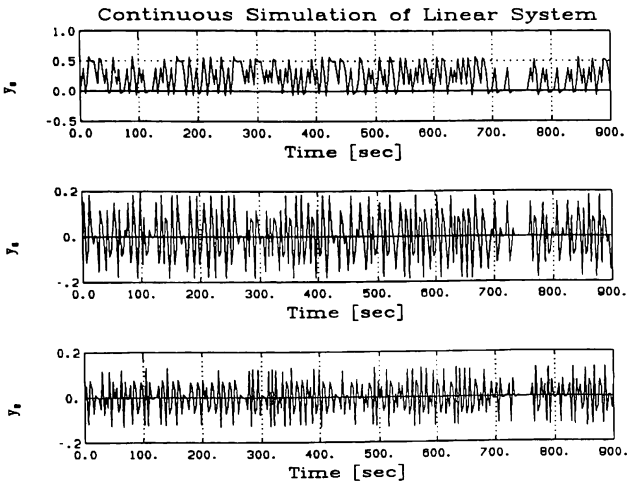

Figure 8. Continuous–time simulation of linear system

Notice that we used the continuous–time simulation here as we could have used a digital oscilloscope. We shall not make any use of the fact that we know the structure of our system.

We shall use our optimal recoding algorithm to discretize our three output variables:

```
DO saps : saps
meas = [u', y'];
m = meas;
FOR i = 2 : 4, ...
   [indx, mi] = SORT(meas(:, i)); ...
   m(:, i) = mi; ...
END
LM = m(1, :);
LM = [LM; 0.5 * (m(100, :) + m(101, :))];
LM = [LM; 0.5 * (m(200, :) + m(201, :))];
LM = [LM; m(300, :)];
raw = meas;   Memb = ONES(meas);   side = ZROW(meas);
to = 1 : 3;
FOR i = 2 : 4, ...
   from = [LM(1 : 3, i), LM(2 : 4, i)]'; ...
   [r, m, s] = RECODE(meas(:, i), 'fuzzy', from, to);
   raw(:, i) = r;   Memb(:, i) = m;   side(:, i) = s;
END
```

The above code segment sorts each trajectory (column) vector separately, then subdivides the sorted vector into three segments of equal size to determine the optimal landmark values (LM). Thereafter, the measurement data are recoded separately for each trajectory. At the end of the code segment, *raw* contains the recoded raw data matrix.

We are now ready to search for the optimal masks. We operate on three separate mask candidate matrices, one for each of the three outputs. We shall compute the quality vector and the mask history matrix since it turns out that we shall put also the suboptimal masks to good use. We shall keep the three best masks and sort them in order of decreasing quality. The following code segment shows the optimal mask analysis for the first output. The other two masks are computed accordingly. Notice that we shall use the first 270 rows of the raw data matrix only.

```
rraw = raw(1 : 270, :);
MMemb = Memb(1 : 270, :);
sside = side(1 : 270, :);
mcan = -ONES(3, 4);
mcan(3, 2 : 4) = [1, 0, 0];
[mask, hm, hr, q1, mhis1] = FOPTMASK(rraw, Mmemb, mcan, 5);
indx = SORT(q1);
m1a = mhis1(:, 4 * (indx(1) - 1) + 1 : 4 * indx(1));
m1b = mhis1(:, 4 * (indx(2) - 1) + 1 : 4 * indx(2));
m1c = mhis1(:, 4 * (indx(3) - 1) + 1 : 4 * indx(3));
m1 = [m1a, m1b, m1c];
```

At the end of this code segment, $m1$ contains the three best masks concatenated to each other from the right.

We are now ready to forecast. During the forecasting process, it will happen from time to time that an input state is encountered which has never before been recorded. In this case, the forecasting routine will come to a halt and leave it up to the user what to do next. We decided to try the following strategy: since the input state depends on the masks, we simply repeat the forecasting step with the next best mask hoping that the problem goes away. If this doesn't help, we try the third mask. This is the reason why we saved the suboptimal masks.

We coded the forecasting in a separate routine called FRC which is called from the main procedure as follows:

```
DEFF frc
inpt = raw(271 : 300, 1);
[pred, cpred] = FRC(rraw, Mmemb, sside, inpt, m1, m2, m3);
```

At this point, it should have become clear why we simulated over 300 steps although we needed only 270 recordings. The final 30 steps of the continuous–time simulation will be used to validate the forecast. *pred* contains the predicted discrete variables, whereas *cpred* contains the reconstructed continuous variables.

The FRC routine operates in the following way: We loop over the 30 steps of the forecast. In each step, we call the SAPS–II routine FFORECAST three times, once with each of the three optimal masks to forecast one value only. At the end of the step, we concatenate the new row (forecast) to the raw data from below, and repeat. If FFORECAST isn't able to predict a value since the input state has never been seen before, it returns the raw data unchanged, i.e., the number of rows upon output is the same as upon input. In that case, we repeat the FFORECAST with the next best mask. If none of the three best masks is able to predict the next step, we pick a value at random. The following code segment shows how FRC works:

```
//[frcst, cfrcst] = FRC(raw, Memb, side, inpt, m1, m2, m3);
   m1a = m1(:, 1 : 4);  m1b = m1(:, 5 : 8);  m1c = m1(:, 9 : 12);
   m2a = m2(:, 1 : 4);  m2b = m2(:, 5 : 8);  m2c = m2(:, 9 : 12);
   m3a = m3(:, 1 : 4);  m3b = m3(:, 5 : 8);  m3c = m3(:, 9 : 12);
   r = raw;   Mb = Memb;   s = side;
   [row, col] = SIZE(raw);
   [n, m] = SIZE(inpt);
   FOR i = 1 : n, ...
      in = inpt(i); ...
      fc = [in, ZROW(1, 3)]; ...
      fcc = [r; fc];   Mbb = [Mb; ONES(fc)];   ss = [s; ZROW(fc)]; ...
      [ff1, Mb1, s1] = FFORECAST(fcc, Mbb, ss, m1a, row + i - 1); ...
      [rf, cf] = SIZE(ff1); ...
      IF rf <> row + i, ...
         [ff1, Mb1, s1] = ...
            FFORECAST(fcc, Mbb, ss, m1b, row + i - 1); ...
         [rf, cf] = SIZE(ff1); ...
         IF rf <> row + i, ...
            [ff1, Mb1, s1] = ...
               FFORECAST(fcc, Mbb, ss, m1c, row + i - 1, 0); ...
            [rf, cf] = SIZE(ff1); ...
            IF rf <> row + i, ...
               ff1 = [ff1; ROUND(RAND(1, 4))]; ...
               Mb1 = Mbb;   s1 = ss; ...
            END, ...
         END, ...
      END, ...
      // ** Same code for ff2 and ff3 **
      ff = [in, ff1(row + i, 2), ff2(row + i, 3), ff3(row + i, 4)]; ...
      MM = [ONES(in), Mb1(row + i, 2), Mb2(row + i, 3), ...
         Mb3(row + i, 4)]; ...
      si = [ZROW(in), s1(row + i, 2), s2(row + i, 3), s3(row + i, 4)]; ...
      r = [r; ff];   Mb = [Mb; MM];   s = [s; si]; ...
   END
   frcst = r;
   from = 1 : 3;
   cpred = ZROW(30, 3);
   FOR i = 2 : 4, ...
      to = [LM(1 : 3, i), LM(2 : 4, i)]'; ...
      rr = r(271 : 300, i);   MMb = Mb(271 : 300, i); ...
      ss = s(271 : 300, i); ...
      c = REGENERATE(rr, MMb, ss, from, to); ...
      cpred(:, i - 1) = c; ...
   END
   cfrcst = meas;   cfrcst(271 : 300, 2 : 4) = cpred;
RETURN
```

We then compared the data from the simulation with the forecast data:

$$simdat = raw(271:300,:);$$
$$frcdat = pred(271:300,:);$$
$$error = simdat - frcdat;$$

It turned out that there was *not a single error* in the forecast in terms of the recoded (discrete) variables. The forecasting procedure worked beautifully.

Figure 9 displays the true continuous signals with the regenerated ones superimposed.

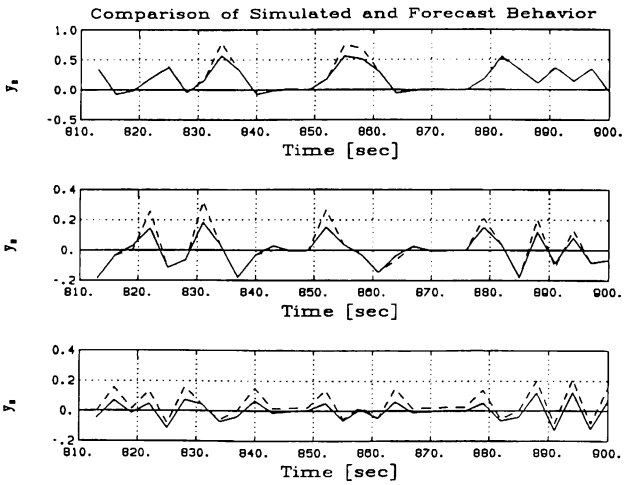

Figure 9. True and regenerated continuous–time signals

The solid lines represent the results from the continuous simulation. They look discontinuous because of the large communication interval of $3sec$ used in the simulation. CTRL–C's plot routine uses linear interpolation between communication points. The dashed lines are the pseudo–continuous signals that were regenerated from the forecast using the fuzzy membership functions. The reconstructed third output variable has a small consistent bias, whereas the reconstructed first two output variables look excellent.

8. SUMMARY

In this paper, we have introduced a number of pattern recognition techniques that can be used for qualitative simulation or forecasting of system behavior. Optimal mask analysis allows us to determine qualitative causality relations among a set of causally related variables. Optimal masks can be viewed as a sort of feature extractor. Similar patterns will lead to the same optimal mask, and therefore, the optimal mask can help us recognize similarities between patterns. These patterns can be either temporal patterns (such as time signals) or static patterns (such as images). Optimal mask analysis is still a far cry from automating the human associative reasoning capability, but it may be a first step towards mimicking this capability in a computer program.

SAPS–II was introduced as a tool to qualitatively analyze systems using these techniques. SAPS–II is available as a CTRL–C function library and also as a ProMatlab (PC–Matlab) toolbox. More details about the proposed methodologies can be found in a forthcoming book [Cellier 1990].

REFERENCES

Cellier, F.E. (1987), "Qualitative Simulation of Technical Systems Using the General System Problem Solving Framework," *International Journal of General Systems 13,* 4, 333-344.

Cellier, F.E. (1990), *Continuous–System Modeling and Simulation – Volume I: Modeling*, Springer Verlag, New York.

Grossberg, S. (1982), *Studies of Mind and Brain,* Reidel, Hingham, MA.

Hayes, P.J. (1979), "The Naïve Physics Manifesto," in *Expert Systems in the Micro–Electronic Age,* (D. Mitchie, ed.), Edinburgh University Press, Edinburgh, Scottland, 242-270.

Hopfield, J.J. (1982), "Neural Networks and Physical Systems With Emergent Collective Computational Abilities," in *Proceedings of the National Academy of Sciences USA,* National Academy of Sciences, Washington, D.C., *79,* 2554-2558.

Klir, G.J. (1985), *Architecture of Systems Problem Solving,* Plenum Press, New York.

Klir, G.J. (1989), "Inductive Systems Modelling: An Overview", *Modelling and Simulation Methodology: Knowledge Systems' Paradigms,* (M.S. Elzas, T.I. Ören, and B.P. Zeigler, eds.), Elsevier Science Publishers B.V. (North–Holland), Amsterdam, The Netherlands.

Kohonen, T. (1984), *Self–Organization and Associative Memory,* Springer Verlag, *Series in Information Sciences 8,* Berlin.

Law, A.M. and W.D. Kelton (1982), *Simulation Modeling and Analysis,* McGraw–Hill, New York.

Minsky, M. and S. Papert (1969), *Perceptrons: An Introduction to Computational Geometry,* MIT Press, Cambridge, MA.

Morgan, A.J. (1990), "Accuracy in Qualitative Descriptions of Behaviour," in *Proceedings WSC'90,* New Orleans, LA.

Narendra, K.S. and K. Parthasarathy (1990), "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks 1,* 1, 4-27.

Ritter H., T. Martinetz and K. Schulten (1990), *Neuronale Netze,* Addison–Wesley, Munich, FRG, (an English translation is currently under preparation).

Shafer, G. (1976), *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, N.J.

Shannon, C.E. and W. Weaver (1964), *The Mathematical Theory of Communication,* University of Illinois Press, Urbana, IL.

Sorsa, T., H.N. Koivo and H. Koivisto (1990), "Neural Networks in Process Fault Diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics,* submitted for publication.

Uyttenhove, H.J. (1979), *SAPS – System Approach Problem Solver,* Ph.D. Dissertation, (G.J. Klir, Adv.), SUNY Binghampton.

Vesanterä, P.J. and F.E. Cellier (1989), "Building Intelligence into an Autopilot – Using Qualitative Simulation to Support General Decision Making," *Simulation 52,* 3, 111-121.

Yuhas, B.P., M.H. Goldstein, Jr. and T.J. Sejnowski (1989), "Integration of Acoustic and Visual Speech Signals Using Neural Networks," *IEEE Communication Magazine 27,* 11, 65-71.

Zadeh, L.A. (1985), "Syllogistic Reasoning in Fuzzy Logic and its Application to Usuality and Reasoning with Dispositions," *IEEE Transactions on Systems, Man, and Cybernetics SMC–15,* 6, 754-763.

Zadeh, L.A. (1986), "A Simple View of the Dempster–Shafer Theory of Evidence and its Implication for the Rule of Combination," *The AI Magazine,* Summer Issue, 85-90.

Zadeh, L.A. (1987), "A Computational Theory of Dispositions," *International Journal of Intelligent Systems 2,* 39-63.