CSE4403 3.0 & CSE6002E - Soft Computing
Winter Semester, 2011

THE BIG ASSIGNMENT (answers)

Due: 1 December 2011 (in class)

Logic, Fuzzy Logic & Sets

1. Andrew, Bernard, Claude, Donald, and Eugene have summer houses along the Atlantic coast. Each wanted to name his house after the daughter of one of his friends - that is, Anne, Belle, Cecilia, Donna, and Eve (but not necessarily in that order). To be sure that their houses would have different names the friends met to make their choices together. Claude and Bernard both wanted to name their house Donna. They drew lots and Bernard won. Claude named his house Anne. Andrew named his house Belle. Eve's father hadn't come, and Eugene phoned to tell him to name his house Cecilia. Belle's father named his house Eve.

What is the name of each friend's daughter? What is the name of his house?

Well, we know that:

- Andrew's house is Belle.
- Bernard's house is Donna.
- Claude's house is Anne.

Their daughters are not so named. Claude cannot be Donna's father and Eugene cannot be Eve's father. Belle's father, who named his villa Eve, can only be Donald or Eugene. Similarly, Eve's father is Donald or Eugene. Since he phoned to the last one, he is Donald. His house is Cecilia. Eugene is Belle's father, Andrew is Donna's father, Bernard is Anne's father, and Claude is Cecilia's father. Thus we have:

Father	Daughter	House
Andrew	Donna	Belle
Bernard	Anne	Donna
Claude	Cecilia	Anne
Donald	Eve	Cecilia
Eugene	Belle	Eve

Isn't deduction wonderful! Imagine the reasoning strategies you must develop when the situation described above becomes many times more complicated. Think of an analogous situation of sorting numbers: bubble sort is intuitive, easy to program and computationally expensive, quick sort is less intuitive but very efficient. When reasoning of the sort described above becomes more complex, our ability to reason clearly becomes clouded by detail. What is called for are procedures which we know will work more efficiently but are perhaps seemingly more abstract initially.

2. Fuzzy logic has been successfully used in control systems methodology. How does fuzzy logic differ from conventional control systems methods?

Fuzzy lolgic incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving control problem rather than attempting to model a system mathematically. The fuzzy logic model is empirically-based, relying on an operator's experience rather than their technical understanding of the system. For example, rather than dealing with temperature control in terms such as "SP =500F", "T

<1000F", or "210C <TEMP <220C", terms like "IF (process is too cool) AND (process is getting colder) THEN (add heat to the process)" or "IF (process is too hot) AND (process is heating rapidly) THEN (cool the process quickly)" are used. These terms are imprecise and yet very descriptive of what must actually happen. Consider what you do in the shower if the temperature is too cold: you will make the water comfortable very quickly with little trouble. Fuzzy logic is capable of mimicking this type of behavior but at very high rate.

Rough Sets

3. Name five features that rough sets theory allows:

- characterization of set of objects in terms of attribute values
- finding dependencies (total or partial) between attributes
- reduction of superfluous attributes (data)
- finding significance attributes
- decision rule generation and others.

4. Given the following information system

Make_model	Cyl	Door	Displace	Compress	Power	Trans	Weight	Milage
USA	6	2	medium	high	high	auto	medium	medium
USA	6	4	medium	medium	medium	manual	medium	medium
USA	4	2	small	high	medium	auto	medium	medium
USA	4	2	medium	medium	medium	manual	medium	medium
USA	4	2	medium	medium	high	manual	medium	medium
USA	6	4	medium	medium	high	auto	medium	medium
USA	4	2	medium	medium	high	auto	medium	medium
USA	4	2	medium	high	high	manual	light	high
Japan	4	2	small	high	low	manual	light	high
Japan	4	2	medium	medium	medium	manual	medium	high
Japan	4	2	small	high	high	manual	medium	high
Japan	4	2	small	medium	low	manual	medium	high
Japan	4	2	small	high	medium	manual	medium	high
USA	4	2	small	high	medium	manual	medium	high

The genetic algorithm (in Rosetta) generates four reducts. What are they?

No.	Reduct Set
1	{make_model, compress, power, trans}
2	{make_model, cyl, compress, trans}
3	{make_model, displace, compress, trans}
4	{make_model, cyl, door, displace, trans, weight}

What is the core?

{make_model, trans}

Neural Networks & Perceptrons

5. How much information does it take to describe a two-input perceptron? The classical description uses a vector of three real-valued parameters: $\mathbf{w} = \langle w_0, w_1, w_2 \rangle$. But the perceptron's decision boundary is a line, which can be uniquely specified with just two parameters, e.g., slope and intercept.

Jack says: "A perceptron can be described with less information than three real numbers. Here's how I would do it: set $s_0 = w_0/w_2$, and $s_1 = w_1/w_2$. From the description $\langle s_0, s_1 \rangle$, I can construct a weight vector $\langle s_0, s_1, 1 \rangle$ that describes a line with the same slope and intercept as the one described by **w**. So it should behave the same as **w** for all inputs."

Jill replies: "A perceptron requires more than that to describe. A line is not an inequality. Consider the case where w_2 is negative. How will that effect the results of your transformation?"

(a) Whose claim is correct, and why? **(b)** How much information does it really take to correctly describe a two-input perceptron? (Don't worry about the case where $w_2 = 0$.)

Answer. Jill's claim is closer to correct. Jack has the right idea initially when he says that three real numbers is overkill; the description of the line is over-determined since w_0, w_1 and w_2 are not independent. But in settling on just two real numbers he is throwing away too much information, because the vectors **w** and **~w** have the same representation in Jack's scheme, even though they produce opposite results. (When w_2 is negative, dividing through by w_2 requires us to exchange $\leq$ for $>$ and vice versa, otherwise the resulting perceptron will misclassify every training point.) So it would appear to take two real numbers plus a sign bit to describe a two-input perceptron: two real numbers to describe the line, as Jack proposes, and one bit to specify which side of the line is class 1 and which is class 0.

However, there is an alternative solution which can do the job with two real numbers: the decision boundary can be described by an angle θ plus a y-intercept. Negating **w** alters θ by 180 degrees.

6. Perceptrons cannot handle tasks which are not separable. A set of points in n-dimensional space are linearly separable if there is a hyper-plane of (n-1) dimensions that separates the sets. In 2-d space, the sets can separated by a straight line.

Can a perceptron learn to classify the following problem correctly? Why?

Input x1	Input x2	output
0	0	0
1	1	0
0	1	1
1	0	1

Even parity ● means even number of 1 bits in the input

Odd parity ○ means odd number of 1 bits in the input

There is no way to draw a single straight line so that the circles are on one side and the dots are on the other side. Thus the perceptrron is unable to find a line separating even parity input patterns from odd parity patterns.

Genetic Algorithms and Evolutionary Computing

7. With regards to Genetic Algorithms

a) Describe a parent selection technique and describe the algorithm.

The technique described is called roulette wheel selection. The idea behind the roulette wheel selection technique is that each individual is given a chance to become a parent in proportion to its fitness. It is called roulette wheel selection as the chances of selecting a parent can be seen as spinning a roulette wheel with the size of the slot for each parent being proportional to its fitness. Obviously those with the largest fitness (slot sizes) have more chance of being chosen.
Roulette wheel selection can be implemented as follows

1. Sum the fitnesses of all the population members. Call this TF (total fitness).
2. Generate a random number n, between 0 and TF.
3. Return the first population member whose fitness added to the preceding population members is greater than or equal to n.

b) Explain the concept of mutation with regard to GA's. Why is it important to have such an operator?

If we use a crossover operator such as one-point crossover we may get better and better chromosomes but the problem is, if the two parents (or worse – the entire population) has the same value in a certain position there is nothing that will change that that value. Mutation is designed to overcome this problem and so add some diversity to the population.

c) If you were developing a GA for The Travelling Salesman Problem (TSP), what type of crossover operator would you use? Show how this crossover method works and explain why you would use it in preference to other crossover operators?

For the TSP we need to use a crossover operator such as *Order-Based Crossover*. This operator ensures that illegal solutions to the TSP problem are not produced (e.g. duplicate and lost cities in this case).

(In the case of the TSP a chromosome will be coded as a list of towns. If we allow the one-point crossover operator we can (and almost definitely will) produce an illegal solution by duplicating some cities and deleting others).

The problem with other operators (such as one-point crossover) is that it can (and invariably will) lead to illegal solutions.

Order-based crossover works as follows (assume the coding scheme represent cities)

Parent 1	A	B	C	D	E	F	G
Parent 2	E	B	D	C	F	G	A
Template	0	1	1	0	0	1	0

Child 1	E	B	C	D	G	F	A
Child 2	A	B	D	C	E	G	F

- Select two parents
- A template is created which consists of random bits
- Fill in some of the bits for child 1 by taking the genes from parent 1 where there is a one on the template (at this point we have child 1 partially filled, but it has some "gaps").

- Make a list of the genes in parent 1 that have a zero in the template
- Sort these genes so that they appear in the same order as in parent 2
- Fill in the gaps in child 1 using this sorted list.
- Create child 2 using a similar process

8. a) Given the following parents, P_1 and P_2, and the template T

P_1	A	B	C	D	E	F	G	H	I	J
P_2	E	F	J	H	B	C	I	A	D	G
T	1	0	1	1	0	0	0	1	0	1

Assume that C_n, are crossover points where $C_n = 0$ means that the crossover point is at the extreme left of the parent.

Show how the following crossover operators work
- one point crossover (using $C_1 = 4$)
- two point crossover (using $C_2 = 2$ and $C_3 = 8$)
- uniform crossover
- order-based crossover

with regards to genetic algorithms

One Point

P_1	A	B	C	D	E	F	G	H	I	J
P_2	E	F	J	H	B	C	I	A	D	G

Assume crossover point at the position shown. The two children would be

C_1	A	B	C	D	B	C	I	A	D	G
C_2	E	F	J	H	E	F	G	H	I	J

Two Point

P_1	A	B	C	D	E	F	G	H	I	J
P_2	E	F	J	H	B	C	I	A	D	G

Assume crossover at the points shown

C_1	A	B	J	H	B	C	I	A	I	J
C_2	E	F	C	D	E	F	G	H	D	G

1	0	1	1	0	0	0	1	0	1

Uniform

P_1	A	B	C	D	E	F	G	H	I	J
P_2	E	F	J	H	B	C	I	A	D	G
T	1	0	1	1	0	0	0	1	0	1

C_1	A	F	C	D	B	C	I	H	D	J
C_2	E	B	J	H	E	F	G	A	I	G

Order-Based

P_1	A	B	C	D	E	F	G	H	I	J
P_2	E	F	J	H	B	C	I	A	D	G
T	1	0	1	1	0	0	0	1	0	1

C_1	A	E	C	D	F	B	I	H	G	J
C_2	E	B	J	H	C	D	F	A	I	G

b) Why is mutation important with regards to genetic algorithms? Give an example to demonstrate your point.

It is important to add diversity to the population. The worst case scenario is that a certain bit position (assuming a bit representation) is the same in every parent. This bit, using an operator such as one point crossover, can never be changed.

For example, given two parents (which we will assume are the entire population)

P_1	0	0	1	0	0	0	1	0	1	1
P_2	1	1	0	1	0	1	0	1	0	0

The shaded bits can never get set to one unless we allow mutation.

c) Genetic algorithms are an example of a population based approach to problem solving. Give your ideas as to other potential population based approaches that could be used.

One example is an approach based on the way ants forage for food (ant algorithms). I am really looking for other ideas from the real world that could be converted into search techniques. Maybe they have ideas on the behaviour of bees, flocking birds etc.

I am looking for more than a description of the real world aspect of the problem. I would like some idea as to how it could be converted into a search algorithm (for example, how do bees find their way from the hive to the flowers – which is similar to ants looking for food).

I do not want meta-heuristic approaches such as simulated annealing, tabu search etc. as these are not population based.

General (includes extra credit points)

9. a) Describe and show a hill climbing algorithm?

The algorithm presented in the lectures is given below. It is taken from the course textbook.
I will accept any suitable algorithm that considers all the neighbourhood states and accepts the best one.

Taking into account part (c) of this question, it would be nice if the student introduced the concept of an ***accept*** function.

Function HILL-CLIMBING(*Problem*) **returns** a solution state
    Inputs: *Problem*, problem
    Local variables: *Current*, a node
                Next, a node
Current = MAKE-NODE(INITIAL-STATE[*Problem*])
Loop do
    Next = a highest-valued successor of *Current*
    If VALUE[Next] < VALUE[Current] **then return** *Current*
    Current = *Next*
End

b) Describe the idea behind simulated annealing?

Simulated Annealing (SA) is motivated by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis etc al in 1953 [Metropolis, 1953). The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing.
If you heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled slowly enough, large crystals will be formed. However, if the liquid is cooled quickly (quenched) the crystals will contain imperfections.
Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, *frozen* state.

In 1982, Kirkpatrick et al (Kirkpatrick, 1983) took the idea of the Metropolis algorithm and applied it to optimisation problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution.

The main parameters to the algorithm are the change in the cost function and the temperature of the system. These dictate if a worse move is to be accepted. The lower the temperature and the greater the difference between the current cost and the potential new cost, the less probability there is of the new state being accepted.

The main idea behind the algorithm is that, unlike hill climbing, it is able to escape from local optima.

c) How would you change the algorithm developed in (a) so that simulated annealing can be used as a search method?

The algorithm presented in the course textbook is given below. However, I would like to see the students pick up from what I said in the lectures that the main difference is that the accept function should be changed so that instead of only taking better moves, worse moves are taken with some probability. In fact, hill climbing could be emulated by running simulated annealing with a temperature of zero.

I will give full marks for making the SA/HC comparison with regards to the accept function. If they simply produce an SA algorithm, then half marks will be given

Function SIMULATED-ANNEALING(*Problem*, *Schedule*) **returns** a solution state
    Inputs : *Problem*, a problem
        Schedule, a mapping from time to temperature
    Local Variables : *Current*, a node
        Next, a node
        T, a "temperature" controlling the probability of downward steps
Current = MAKE-NODE(INITIAL-STATE[*Problem*])
For t = 1 **to** ∞ **do**
    T = *Schedule*[t]
    If T = 0 **then return** C*urrent*
    Next = a randomly selected successor of *Current*
    ΛE = VALUE[*Next*] – VALUE[*Current*]
    if ΛE > 0 **then** *Current* = *Next*
    else *Current* = *Next* only with probability exp(-ΛE/T)

d) Show, by way of an example, how changing the parameters to the simulated annealing algorithm, affect whether it will accept a given move?

The table below shows how different temperatures and differences in the evaluation function affect the probability of worse moves being accepted (better moves are always accepted). You can see that at lower temperatures, with the same change in the evaluation function, the probability of worse moves being accepted are less.

It is important to know the acceptance function, i.e., exp(-*C*/*T*), where *C* is the change in the evaluation function and *T* is the current temperature of the system.

This table implements this function.

Probability of accepting with high temperature				Probability of accepting with low temperature		
Change in Evaluation Function	**Temperature of System**	**exp(-C/T)**		**Change in Evaluation Function**	**Temperature of System**	**exp(-C/T)**
10	100	0.904837418		10	10	0.367879441
20	100	0.818730753		20	10	0.135335283
30	100	0.740818221		30	10	0.049787068
40	100	0.670320046		40	10	0.018315639
50	100	0.60653066		50	10	0.006737947
60	100	0.548811636		60	10	0.002478752
70	100	0.496585304		70	10	0.000911882
80	100	0.449328964		80	10	0.000335463
90	100	0.40656966		90	10	0.00012341
100	100	0.367879441		100	10	4.53999E-05