Cooperating Intelligent Systems – Written Exam January 2010

You must achieve at least 50% of the points on this written exam to continue to the oral exam.

The total number of points is 60.

No books, mobile phones or calculators are permitted during the exam.

1 Search

Pac-Man is an old arcade game where a "Pac-Man" figure (the figure that looks like a pie chart in the figure below) tries to eat as many dots as possible in a short time (before the "ghosts" come out of their house and catch "Pac-Man"). The game is shown in the figure below.

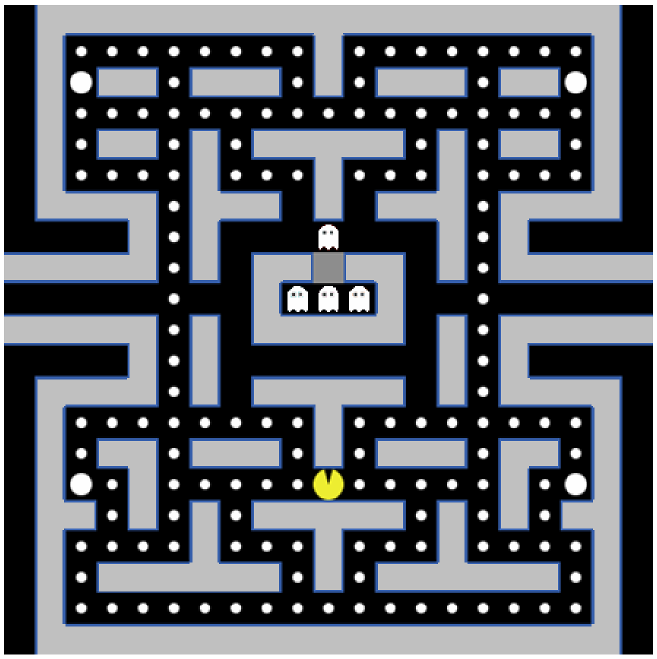


Figure 1: Pac-Man game.

The task is to control Pac-Man to eat as many dots as possible (big dots are worth more than small dots in the real game). The idea is to have Pac-Man eat all dots on the board before being caught by a ghost.

We consider a simplified version of the game here: there are no ghosts (no adversary) and all dots have the same value.

(a) Define the state space for the game (i.e. a notation for every possible state in the game). [2p]

(b) What is the maximum branching factor for the game? [1p]

(c) For a game with *N* squares where Pac-Man can be (and that can be occupied by dots), what is the size of the state space? [1p]

(d) What is the goal test? [1p]

(e) Formulate a suitable heuristic for the game. Explain why it is a suitable heuristic. [2p]

(f) Explain the greedy best first search and the A* search. Will it make any difference which of these informed search methods you use? [5p]

SOLUTION:

(a) Define the state space for the game (i.e. a notation for every possible state in the game).

Each square can be empty, occupied by a dot or by Pac-Man, i.e. we can have the values $\{e, d, p\}$ for every square. The constraint is that there is only one Pac-Man (only one square with value *p*). The task is to have all dots eaten (i.e. no square with value *d*).

(There are other possible notations, this is just a suggestion.)

(b) What is the maximum branching factor for the game?

Pac-Man can move (at most) up, down, left and right. This means that the maximum branching factor is 4.

(c) For a game with *N* squares where Pac-Man can be (and that can be occupied by dots), what is the size of the state space?

There are three possible states for every square so the size is 3^N.

(d) What is the goal test?

That there is no dot left in any square (i.e. the number of squares with value *d* is zero).

(e) Formulate a suitable heuristic for the game. Explain why it is a suitable heuristic.

The number of squares with value *d*, this is a minimum number of moves that Pac-Man must do to empty all the squares.

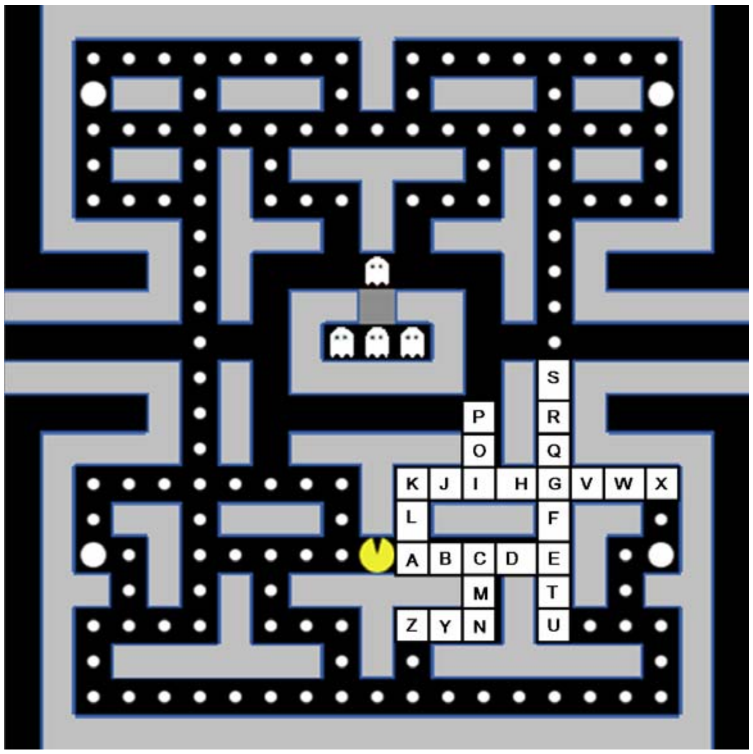
(f) Explain the greedy best first search and the A* search. Will it make any difference which of these informed search methods you use?

With the following notation: $h(n)$ = heuristic value for node *n* (i.e. estimated distance to the goal from node *n*) and $g(n)$ = cost for going to node *n*.

GBFS: Expand the node with minimum $f(n) = h(n)$.

A*: Expand the node with minimum *f(n)* = *h(n)* + *g(n)*. The heuristic must be admissible for A* to give an optimal solution. Admissible = never overestimate the true cost to the goal.

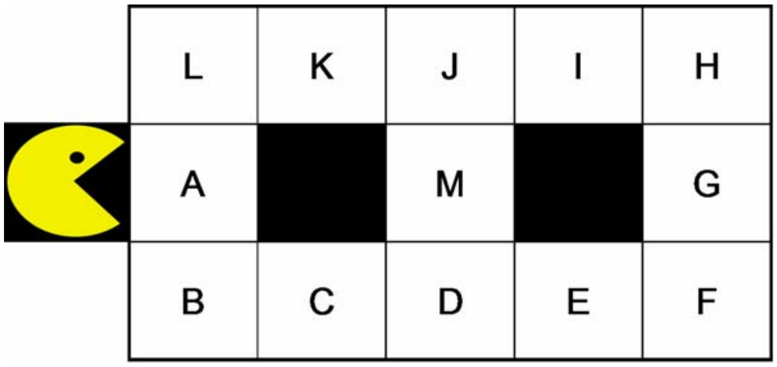
The question about "any difference" is tricky. We know that A* is always at least as good as GBFS but perhaps GBFS is as good as A* in this particular case? The answer depends on the search horizon. The difference between GBFS and A* in this case depends on how they treat "left over" dots.



The situation is illustrated in the figure above. The trick to emptying the board quickly is how to treat dots like the dot in the square labeled "F". If Pac-Man needs to go back a long way to pick it up then the path will not be optimal. However, it may take a long time to discover this so the search algorithm might not discover the problem in time.

Of course, a smarter heuristic might improve the situation but the smarter heuristic might be much more complicated to compute.

A simplified example is shown below.

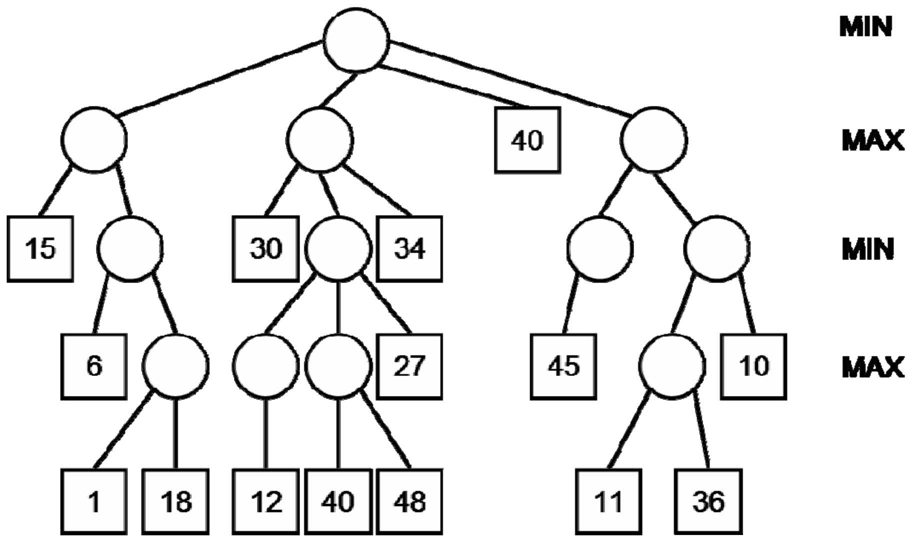


Pac-Man is about to eat the dots in the squares A through M, the question is just what the optimal order is (with minimum number of steps). A human would quickly see that an optimal order is (e.g.) A-B-C-D-E-F-G-H-I-J-M-J-K-L = 14 moves. A suboptimal order is (e.g.) A-B-C-D-E-F-G-H-I-J-K-L-K-J-M = 15 moves. The problem is that the suboptimality of the latter order is not discovered until the 14th move.

So, the answer is that A* is better than GBFS (we know this from theory) but that there may be no difference between their performance on this task if the search horizon is not deep enough.

2 Game playing

The tree below is a game tree for a hypothetical game. The square nodes are goal nodes.



(a) Fill in the minimax values for the empty nodes. [2p]

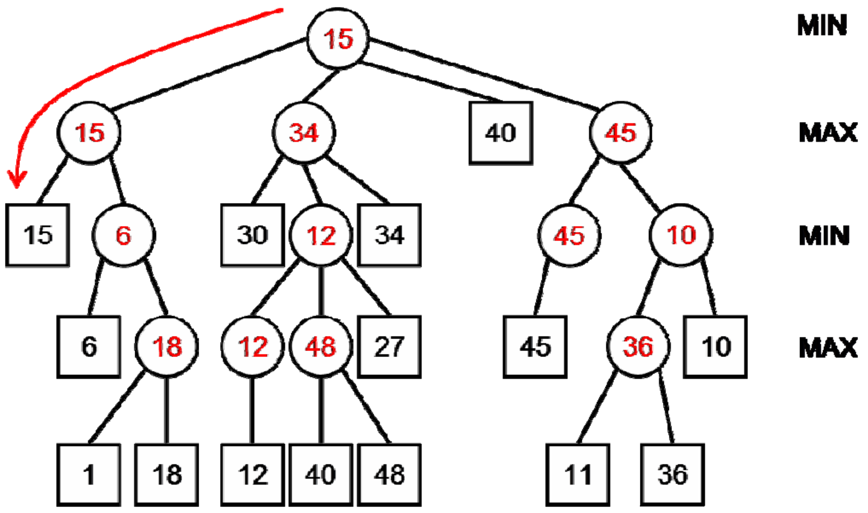
(b) Mark the optimal path for the game. [1p]

(c) Write the rules for alpha-beta pruning. [2p]

(d) Mark the nodes that will not be expanded when alpha-beta pruning is used. [3p]

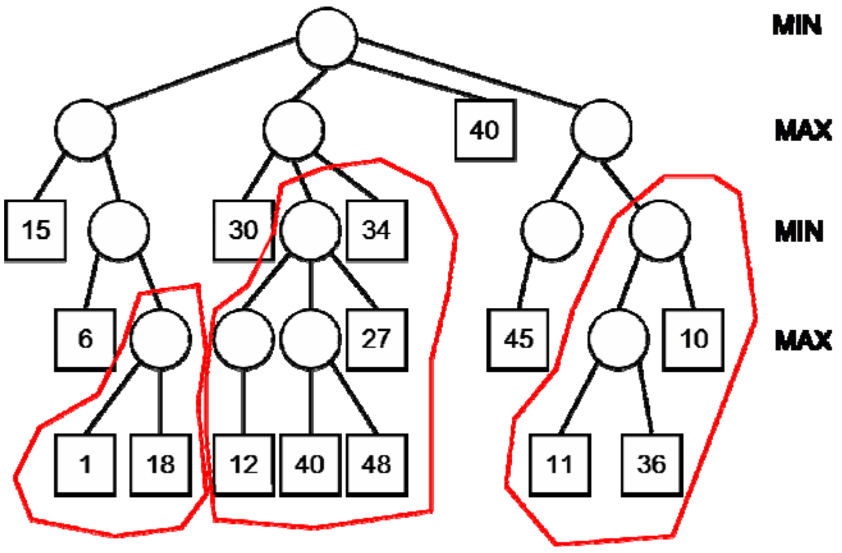
SOLUTION:

(a) + (b) The minimax values are in red below. The optimal game path is shown with an arrow.

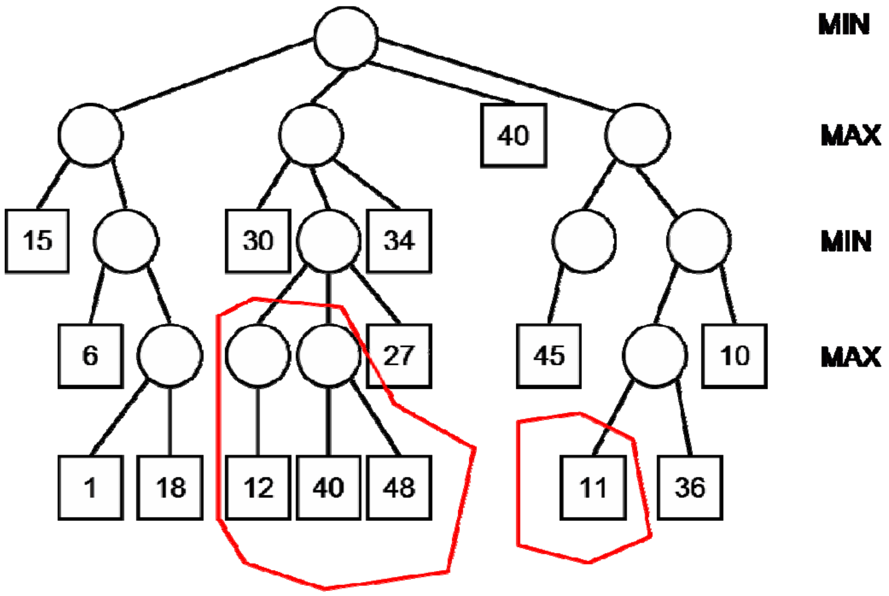


(c) If $\beta < \alpha$ higher up in the tree then stop expanding. If $\alpha > \beta$ higher up in the tree then stop expanding.

(d) The solution below assumes that the nodes are expanded from left to right.



The solution below assumes that the nodes are expanded from right to left.



3 Logic

(a) Translate the following four English sentences to first order logic (FOL). [4p]

1. Anyone passing his history exams and winning the lottery is happy.
2. Anyone who studies or is lucky can pass all his exams.
3. John did not study but John is lucky.
4. Anyone who is lucky wins the lottery.

(b) Convert them to conjunctive normal form (CNF). [6p]

(c) Answer the query “Is John happy?”. Use the resolution refutation algorithm. [5p]

SOLUTION:

(a) Translate the following four English sentences to first order logic (FOL). [4p]

1. Anyone passing his history exams and winning the lottery is happy.

$\forall x\, Pass(x, HistoryExam) \wedge Win(x, Lottery) \Rightarrow Happy(x)$

2. Anyone who studies or is lucky can pass all his exams.

$\forall x \forall y\, Study(x) \vee Lucky(x) \Rightarrow Pass(x, y)$

3. John did not study but John is lucky.

$\neg Study(John) \wedge Lucky(John)$

4. Anyone who is lucky wins the lottery.

$\forall x\, Lucky(x) \Rightarrow Win(x, Lottery)$

(b) Convert them to conjunctive normal form (CNF). [8p]

First, implication elimination (note the placement of the negation)

1 $\forall x\, \neg\left[Pass(x, HistoryExam) \wedge Win(x, Lottery)\right] \vee Happy(x)$

2 $\forall x \forall y\, \neg\left[Study(x) \vee Lucky(x)\right] \vee Pass(x, y)$

3 $\neg Study(John) \wedge Lucky(John)$

4 $\forall x\, \neg Lucky(x) \vee Win(x, Lottery)$

Then, drop $\forall$ quantifiers

1 $\neg[Pass(x, HistoryExam) \wedge Win(x, Lottery)] \vee Happy(x)$

2 $\neg[Study(x) \vee Lucky(x)] \vee Pass(x, y)$

3 $\neg Study(John) \wedge Lucky(John)$

4 $\neg Lucky(x) \vee Win(x, Lottery)$

Then, move the negation ($\neg$) inwards ("De Morgan")

1 $[\neg Pass(x, HistoryExam) \vee \neg Win(x, Lottery)] \vee Happy(x)$

2 $[\neg Study(x) \wedge \neg Lucky(x)] \vee Pass(x, y)$

3 $\neg Study(John) \wedge Lucky(John)$

4 $\neg Lucky(x) \vee Win(x, Lottery)$

Then, distribute the or ($\vee$), or just drop the parentheses if possible

1 $\neg Pass(x, HistoryExam) \vee \neg Win(x, Lottery) \vee Happy(x)$

2 $[\neg Study(x) \vee Pass(x, y)] \wedge [\neg Lucky(x) \vee Pass(x, y)]$

3 $\neg Study(John) \wedge Lucky(John)$

4 $\neg Lucky(x) \vee Win(x, Lottery)$

Then, separate the sentences with $\wedge$ into separate statements

1 $\neg Pass(x, HistoryExam) \vee \neg Win(x, Lottery) \vee Happy(x)$

2a $\neg Study(x) \vee Pass(x, y)$
2b $\neg Lucky(x) \vee Pass(x, y)$

3a $\neg Study(John)$
3b $Lucky(John)$

4 $\neg Lucky(x) \vee Win(x, Lottery)$

This is the KB in CNF. The variables should also be standardized separately (e.g. the *x* in 2a is not the same as *x* in 4) but there is no deduction of points if this is not done.

(c) Answer the query “Is John happy?”. Use the resolution refutation algorithm. [5p]

This means combining the negation of the query with the KB and see if we get the empty set if we apply the resolution rule repeatedly.

The negation of the query is $\neg Happy(John)$. We call this statement Q. The list below shows the result of applying the resolution rule with the statements listed on the left. The result is shown in the center and the rightmost column shows the reference number of the result.

Q + 1	$\neg Pass(John, HistoryExam) \vee \neg Win(John, Lottery)$	5
5 + 2b	$\neg Lucky(John) \vee \neg Win(John, Lottery)$	6
6 + 4	$\neg Lucky(John)$	7
7 + 3b	$\varnothing$	

We have proved that John is happy.

4 Bayesian networks

It is possible to do fault detection and sensor validation on a machine with the help of Bayesian networks. Here we will study the (simplified) case of a gas turbine. Denote the power generated in the turbine by MW, the combustion temperature in the turbine by T, the pressure in the turbine by PT, the flow of gas by FG, the gas valve position by V, the gas fuel pressure supply by PS and the pressure at the compressor output by PC (see sketch below).

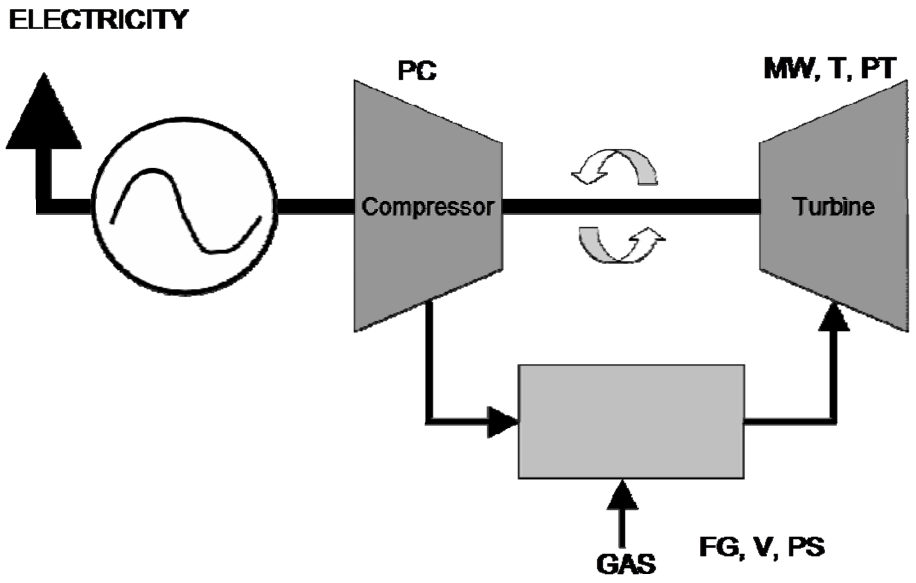


Figure 2: Sketch of gas turbine for generating electricity.

(a) The following holds (approximately): The power generated (MW) in the turbine depends on the temperature (T) and pressure (PT) in the turbine. The temperature (T) depends on the flow of gas (FG), which depends on the gas valve position (V) and the gas fuel pressure supply (PS). The pressure at the turbine (PT) depends on the output pressure from the compressor (PC).

Draw a Bayesian network with the seven variables (PC, FG, V, PS, MW, T and PT) that expresses this. [3p]

(b) The temperature (T) in the turbine can be estimated from the other measurements. Write down the expression for the conditional probability P(T | MW, PT, PS, V, FG, PC) that follows from the Bayesian network. [1p]

(c) The estimated temperature can be compared to the actual measured temperature in the turbine. If the measured temperature is very unlikely (has low probability) then this can be a signal that the temperature sensor is malfunctioning or that the combustion system is malfunctioning. However, it can also be caused by another malfunctioning sensor.

To figure out what could be causing the faulty temperature readout in the turbine we only need to consider the nodes that actually influence the temperature estimate. This means that we only need to consider the temperature node (T) plus the Markov blanket for the temperature node.

Write down the definition of the Markov blanket. [1p]

Mark the nodes in the Bayesian network that constitute the Markov blanket for the temperature node T. [4p]

(d) The variables in the network can be either continuous or discrete, depending on how you choose to implement it.

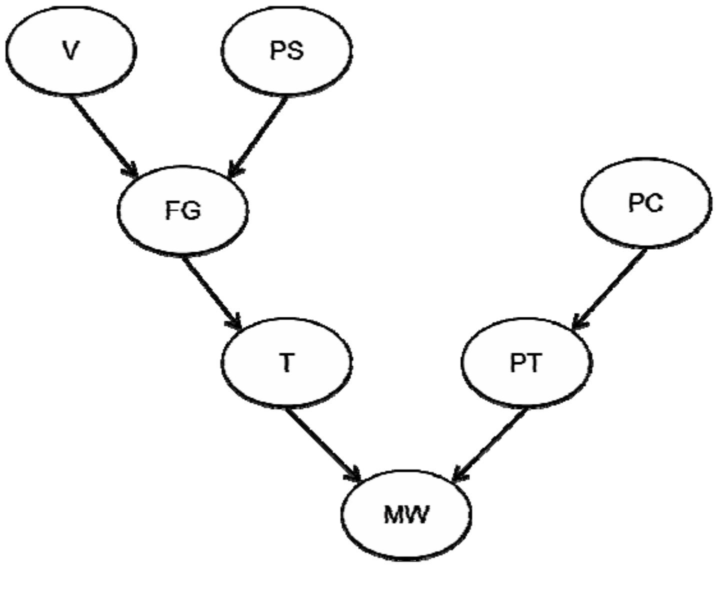
Write down and motivate the functions you would use if you choose to implement this with discrete variables. [3p]

Write down and motivate the functions you would use if you choose to implement this with continuous variables. [3p]

SOLUTION:

(a) The following holds (approximately): The power generated (MW) in the turbine depends on the temperature (T) and pressure (PT) in the turbine. The temperature (T) depends on the flow of gas (FG), which depends on the gas valve position (V) and the gas fuel pressure supply (PS). The pressure at the turbine (PT) depends on the output pressure from the compressor (PC).

Draw a Bayesian network with the seven variables (PC, FG, V, PS, MW, T and PT) that expresses this.



(b) The temperature (T) in the turbine can be estimated from the other measurements. Write down the expression for the conditional probability P(T | MW, PT, PS, V, FG, PC) that follows from the Bayesian network. [1p]

$$P(T \mid MW, PT, FG, PC, V, PS) = \frac{P(T, MW, PT, FG, PC, V, PS)}{P(MW, PT, FG, PC, V, PS)} =$$

$$\frac{P(MW \mid PT, T)P(PT \mid PC)P(T \mid FG)P(FG \mid V, PS)P(PC)P(V)P(PS)}{\sum_T P(MW \mid PT, T)P(PT \mid PC)P(T \mid FG)P(FG \mid V, PS)P(PC)P(V)P(PS)} =$$

$$\frac{P(MW \mid PT, T)P(T \mid FG)}{\sum_T P(MW \mid PT, T)P(T \mid FG)}$$

The sum over *T* is an integral over *T* if *T* is a continuous variable.

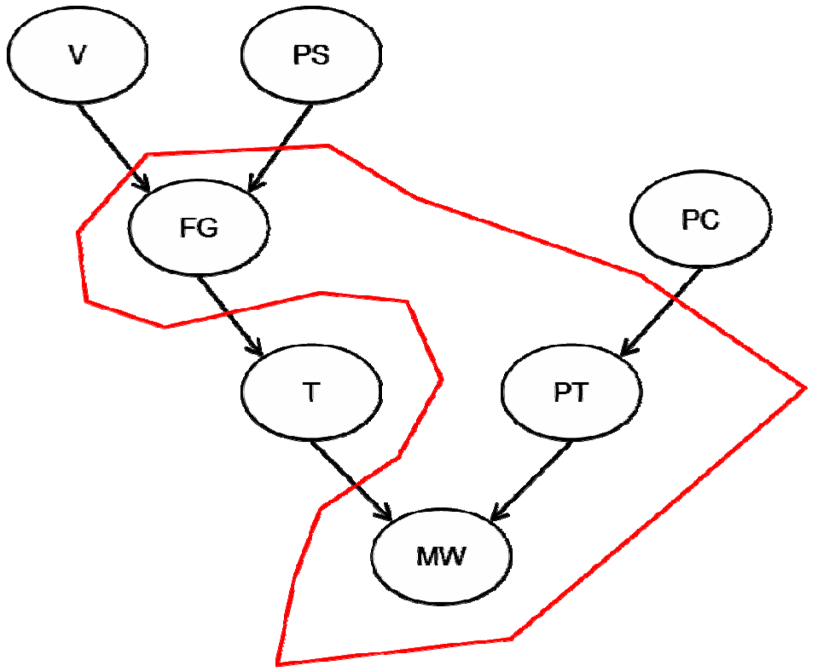
(c) The estimated temperature can be compared to the actual measured temperature in the turbine. If the measured temperature is very unlikely (has low probability) then this can be a signal that the temperature sensor is malfunctioning or that the combustion system is malfunctioning. However, it can also be caused by another malfunctioning sensor.

To figure out what could be causing the faulty temperature readout in the turbine we only need to consider the nodes that actually influence the temperature estimate. This means that we only need to consider the temperature node (T) plus the Markov blanket for the temperature node.

Write down the definition of the Markov blanket. [1p]

Mark the nodes in the Bayesian network that constitute the Markov blanket for the temperature node T. [4p]

The Markov blanket of a node is the parents, the children and the children's parents. (This is hinted in the answer to (b) above). They are shown in red in the figure below.



(d) The variables in the network can be either continuous or discrete, depending on how you choose to implement it.

Write down and motivate the functions you would use if you choose to implement this with discrete variables. [3p]

Write down and motivate the functions you would use if you choose to implement this with continuous variables. [3p]

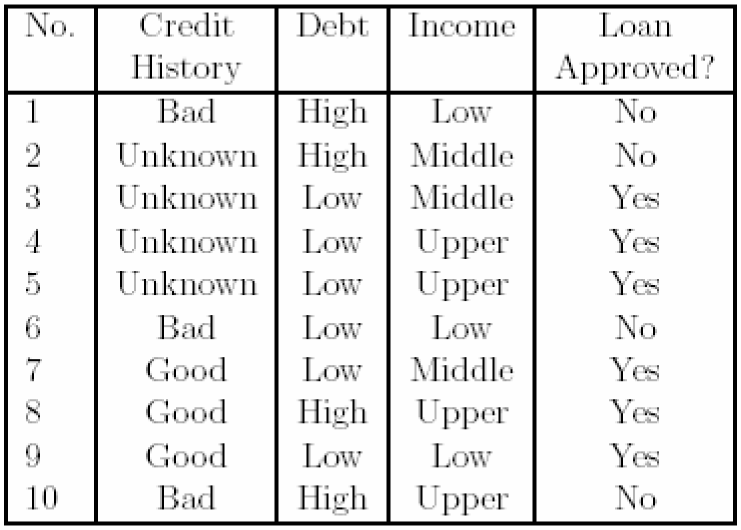
With discrete variables we can for example use lookup tables (like the probability tables we have used in the course) or noisy-OR (and/or noisy-MAX).

With continuous variables we can for example use different parametric probability densities (like the normal density). With mixtures of discrete and continuous variables we can use (e.g.) logistic functions.

5 Decision trees

Consider the training set given in the table below from a loan approval application. The ten training examples are characterized in terms of three attributes. The rightmost column gives the correct answer: whether or not a loan is approved. All the possible values for each attribute are in the table.

Show how you build a decision tree that is able to determine whether to approve a loan or not. Explain all the steps you take. Draw the final decision tree. [10p]



No.	Credit History	Debt	Income	Loan Approved?
1	Bad	High	Low	No
2	Unknown	High	Middle	No
3	Unknown	Low	Middle	Yes
4	Unknown	Low	Upper	Yes
5	Unknown	Low	Upper	Yes
6	Bad	Low	Low	No
7	Good	Low	Middle	Yes
8	Good	High	Upper	Yes
9	Good	Low	Low	Yes
10	Bad	High	Upper	No

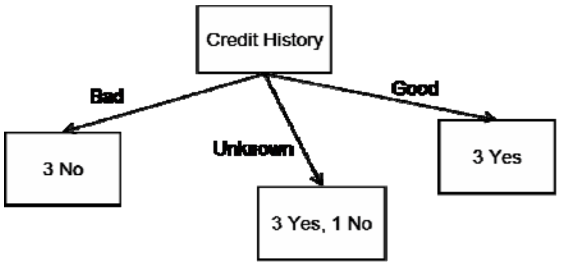
SOLUTION:

We must go through the different possible splits on variables and see which ones that lead to the highest purity. We measure purity with entropy (we could also use other measures like the Gini index but entropy is the one we have covered in the course).

There are 10 observations, 4 with No and 6 with Yes. The initial entropy is thus (entropy is here denoted by *S*).

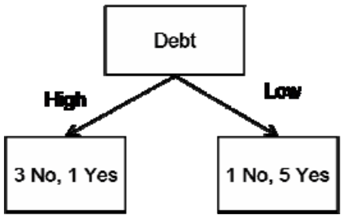
$$S_{init} = -\frac{4}{10}\log\left(\frac{4}{10}\right) - \frac{6}{10}\log\left(\frac{6}{10}\right)$$

We have tree variables: *Credit History*, *Debt* and *Income.* The resulting entropy when making a first split on these is shown below.

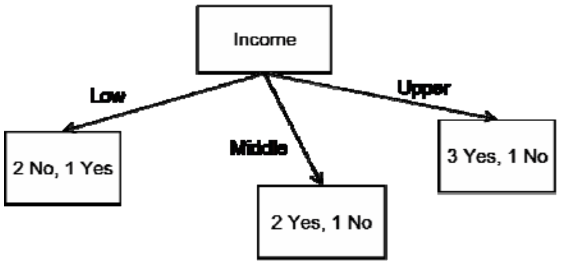


$$S_{CreditHistory} = \frac{4}{10}\left(-\frac{3}{4}\log\left(\frac{3}{4}\right)-\frac{1}{4}\log\left(\frac{1}{4}\right)\right)$$

Note: The entropy is zero for completely pure nodes.



$$S_{Debt} = \frac{4}{10}\left(-\frac{3}{4}\log\left(\frac{3}{4}\right)-\frac{1}{4}\log\left(\frac{1}{4}\right)\right)+$$
$$\frac{6}{10}\left(-\frac{1}{5}\log\left(\frac{1}{5}\right)-\frac{4}{5}\log\left(\frac{4}{5}\right)\right)$$

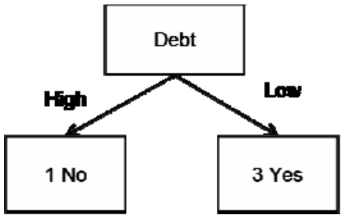


$$S_{Income} = \frac{3}{10}\left(-\frac{2}{3}\log\left(\frac{2}{3}\right)-\frac{1}{3}\log\left(\frac{1}{3}\right)\right)+$$
$$\frac{3}{10}\left(-\frac{2}{3}\log\left(\frac{2}{3}\right)-\frac{1}{3}\log\left(\frac{1}{3}\right)\right)+$$
$$\frac{4}{10}\left(-\frac{3}{4}\log\left(\frac{3}{4}\right)-\frac{1}{4}\log\left(\frac{1}{4}\right)\right)$$

It is obvious that the split on *Credit History* leads to the smallest entropy (largest decrease in entropy). So, the first split is on *Credit History*. The entropy after this split is

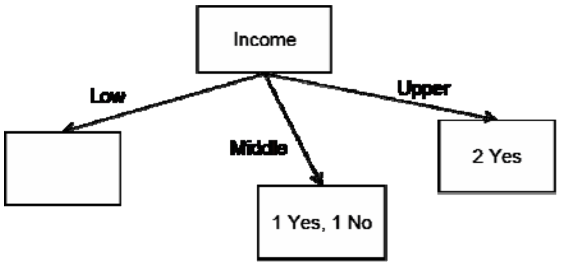
$$S_{CreditHistory} = \frac{4}{10}\left(-\frac{3}{4}\log\left(\frac{3}{4}\right)-\frac{1}{4}\log\left(\frac{1}{4}\right)\right).$$

The resulting entropy when making a second split on any of the two remaining variables is shown below. We of course do not make any further splits on the pure nodes.



$$S_{Debt,2} = 0$$

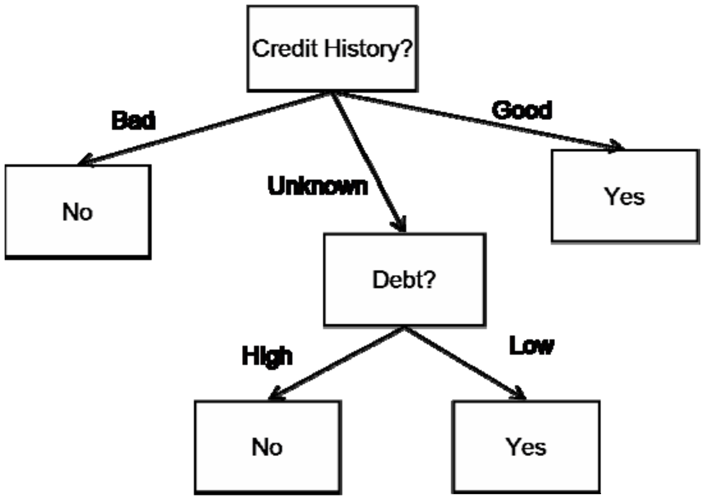
Since the entropy of a pure node is zero.



$$S_{Income,2} = \frac{2}{4}\left(-\frac{1}{2}\log\left(\frac{1}{2}\right)-\frac{1}{2}\log\left(\frac{1}{2}\right)\right)$$

Again, the result is obvious. The lowest entropy (zero) is achieved by making a second split on Debt.

We don't need to make any third split; all nodes are pure after only two splits. The final decision tree is below.


Credit History?
Bad
Unknown
Good
No
Yes
Debt?
High
Low
No
Yes