SOLUTION TO

TRIAL*examination* TDT4171 Artificial Intelligence Methods

<u>Question 1</u> – Bayesian Networks and Influence diagrams (30%)

a) Describe the syntax and semantics of a Bayesian Network.

Syntax:

- Nodes and links between them
- Nodes represent random variables, one node per variable
- Nodes have a finite number of states
- Links are *directed*, and the links between the nodes together define a *directed, acyclic graph*.
- Nodes are augmented with "local" conditional probability distributions (Probability of that variable given its parents)

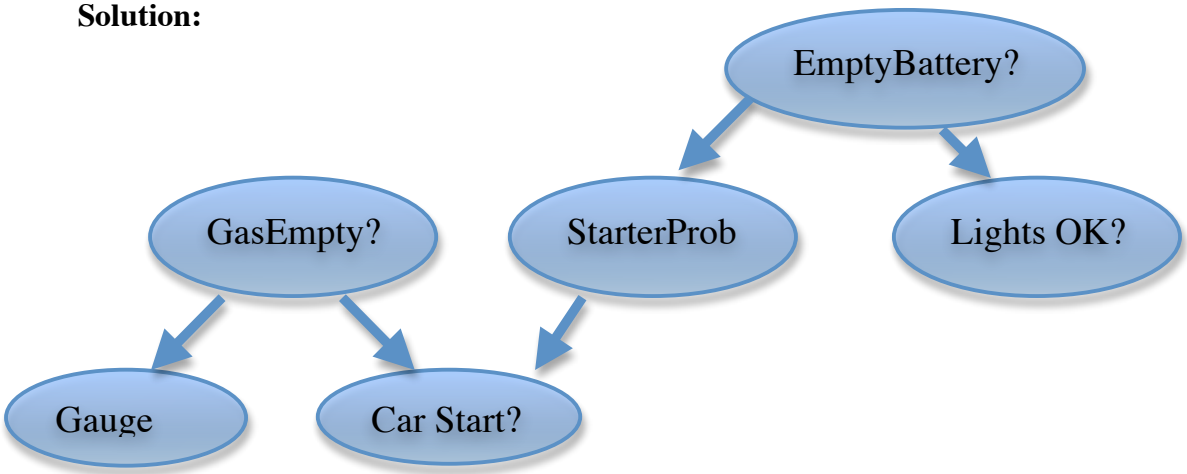
Semantics:

Here it is sufficient to show the "global semantics" of the BN (the factorization of the joint). If you want to, you can show off with information about local semantics, conditional independences, etc., but all of this is actually implied by the global semantics.

b) Model the following problem using a Bayesian network (only the structure is required, the quantitative part should not be given):

If my car does not start, it is eithe due to a problem with the starting motor or that I have run empty on gas. I can read off the level of gas in the tank by looking at the gauge (but the gauge does not always work). The most common reason for the starter motor too fail is that the battery is empty. I can check the battery by looking at the head lights: If they shine the battery is OK, if not it is usually because the battery is flat (but there may also be other reasons).

Solution:



It is a good idea to comment on the model you build: Why are the variables as they are? Why have you included the links that you have? You may also say something regarding the inferences, e.g., something like "If the lights are NOT OK, this will influence the belief that the battery is empty, which in turn makes us update our belief about the starter motor being OK, and finally tell us something about whether or not the car will start." This helps us understand your model (in case it is not identical to the one in the sensors' solution.

If it is difficult to understand the states of some of your variables, it may be good to say what states you have thought up. If the question asks for STRUCTURE you should not waste your time on describing the NUMBERS in the model.

c) *Explain the syntax of an influence diagram. What is the maximum expected utility principle, and how does it relate to an influence diagram?*

Syntax is as for BNs, but with additional nodes:

- Boxes for decisions
- Diamonds for utilities
- There **must be** a directed path tying all decisions together
- Utility nodes have no children
- Decision nodes have finite set of states (representing decision alternatives)
- Utilities do not have states

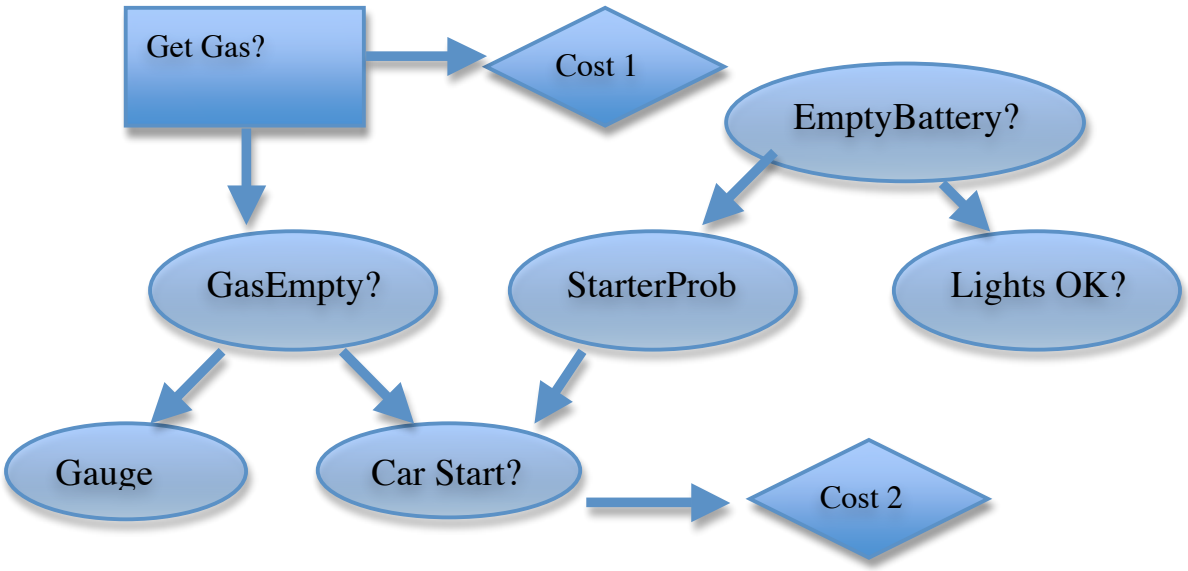
MEU is the principle that any rational agent must comply with, basically it says that when acting rationally, there exists a utility function that allocates a real number to any situation. The function is such that the agent always prefers states with a high utility to those with lower utility. Moreover, if the utility an agent will obtain is uncertain, any rational agent MUST prefer the scenario that gives the highest utility on average (or, as the principle states, Maximum Expected Utility). The good thing about this is that MEU *defines* rational behaviour, and MEU is therefore the behaviour we want from our agent.

Influence diagrams solve decision problems by calculating expected utility of any decision (or sequence of decisions) that can be made, and therefore is a solution technique that will help our agent being rational.

d) *Extend the Bayesian Network from part (b) to incorporate the following decision problem:*

When my car does not start, I have two strategies to get to work: I can walk to the gas station and buy a can of gas and try to fill the tank. This takes 10 minutes; alternatively I can walk to the office, which takes 25 minutes. Driving to office takes 5 min. I wonder whether I should walk should go to the gas station, get gas, and try to get the car to work or walk directly to the office, when I want to get there as quickly as possible.

Solution:



Question 2 - Learning (15%)

a) *What are the main reasons for letting an agent learn from its environment (as opposed to hardcode all the agent's reasoning by hand)*

Solution:

The following three reasons are mentioned in the book, it is also good if you come up with additional (relevant) reasons for learning on your , too.

- Essential for unknown domains
- Important in "changing" environments
- Useful system construction method in its own right (due to "laziness" of constructor)

b) *Describe the DT learning algorithm informally. What is Occam's razor, how does it come into play when learning decision trees?*

Solution:

- Decision Trees are learned Top-Down, and it is a recursive learning technique. The pseudo-code'ish description here shows the poorest level of detail you can get away with at the examination; it is not ready-to-implement, as you can see. If you want to, you are free to make more formal descriptions, and this is something you will get paid for – but only up to a point.

  Algorithm:

  1. Use all relevant data to find which attribute you want to split on. Call this attribute <best>
  2. Make one leaf for each of the values <best> can take.
  3. For each leaf <i>: Restrict the data so that you only use data where <best> = <i>; if there is only one class in this restricted set, make a leaf node labelled by that

class, otherwise go to 1 with the restricted dataset, and create a new tree below the link.

- Occams razor says that you should not complicate what can be done easily, or in our case: There is no reason for making complex decision trees when smaller ones will perform equally well. This is a very important insight, in particular, as it will help you to avoid overfitting.

- In our case, Occam's razor is relevant when it comes to choosing the "best" attribute. In ID3, the learning algorithm we used which calculates entropy and all that, the point is that we want a heuristic that biases towards SHORT DECISION TREEES, i.e., in accordance with Occam.

c) *Gradient descent is a powerful, general-purpose algorithm, which amongst other things can be used for learning the weights of a perceptron network. Describe the main steps, and the strengths and weaknesses of the learning algorithm.*

Solution:
You can solve this task either by describing gradient descent in general or for the perceptron in particular. Here I do it in the general way:

Gradient Descent is a general-purpose learning algorithm for optimization in a CONTINUOUS search space. The requirement here is that we for any position in the search (set of weights in the perceptron) can calculate the GRADIENT of the function we try to minimize (typically a penalty function or an error function).

Algorithm:
1. Start at a random position
2. Move a small step in the direction of steepest descent (in the direction opposite to the gradient). The length of the step is proportional to the length of the gradient vector.
3. As long as the gradient is not zero, go to 2

Pros: Simple to implement, *typically* reasonably fast, *often* gives good results, only one parameter (step size)
Cons: Can be stuck in local minima, *can* be very slow, can be difficult to find good value of stepsize (although one will typically use 5% rather blindly), requires that we are able to calculate the gradient

<u>Question 3</u> – Case-based reasoning (15%)

Describe the four steps in the CBR cycle: Give their names, explain what happens in each step, and explain how general domain knowledge is/can be used in each of the steps.

Solution:
The four steps in the cycle are:

1. Retrieve: Choose similar case(s) to the current problem. The new case is typically represented by attribute-value-pairs, and it should be compared with those in the case-base. Domain knowledge can be used to calculate a (knowledge-boosted) similarity between new and old cases. If no knowledge is available, one need to use ("shallow") statistically based measures.
2. Reuse: Reuse the solution(s) of the retrieved case(s); this can be the (knowledge intensive) adaption of the solution method or a (knowledge poor) voting of classification results, or solutions in-between.
3. Revise: Get a revised solution (this step involves the user more than the system)
4. Retain: Save solved case in case-base if required, update domain knowledge if required

Showing off by making a nice drawing will always be good here. It can also pay off to give a bit deeper description than what I did here, e.g., by using an example or two.

<u>Question 4</u> – Markov Decision Processes (30%)

a) What assumptions are used when modelling Markov Decision Processes (also called MDPs)

Solution:

- The Markov assumption: Future is conditionally independent of the past given the current situation.
- The domain is fully observable
- The uncertainty in the domain is due to uncertainty in the *effect of actions*, not, e.g., in which state the world is in.
- The domain is stationary, so the effect of an action A in situation S does not depend on the time where action A is used.
- There is a reward given at each time *t*, and this reward depends on the situation at time *t* and the action taken at time *t*, and not the previous history.
- We want to maximize the CUMMULATIVE, DISCOUNTED reward over an infinite time horizon.

If I give you questions like "what is the assumptions behind <whatever>", you may want to include an example or two to show domains that (approximately) satisfy the assumptions. This shows us/makes us believe that you have not just learned something by heart, by also understood it.

b) Consider a robot manoeuvring in the grid-world in Figure 1. The robot receives an immediate reward of $100 if it arrives at the top-left corner of the grid, there are no cost involved in moving between the cells of the grid, but penalties of $50 and $10 are given for entering cells (2,2) and (2,3), respectively. In any cell, the robot can choose between the actions `up`, `down`, `right`, *and* `left`. *There is a probability 0.7 for the robot to correctly implement the action it actually decides to, and probability 0.1 for doing each of the actions it chose not to implement. Hence, if the robot decides to move* `left` *we have*

P(Doing action ***left*** *| Decided to do* ***left****) = 0.7,*
P(Doing action ***right*** *| Decided to do* ***left****) = 0.1,*
P(Doing action ***up*** *| Decided to do* ***left****) = 0.1,*
P(Doing action ***down*** *| Decided to do* ***left****) = 0.1,*
… and similarly when deciding to do the other action.

Show the first two iterations of the value iteration scheme to solve this MDP. Use discount factor 0.1.

	Column 1	*Column 2*	*Column 3*
Row 1			***Gold state: $100***
Row 2		***Trap: -$50***	***Trap: -$10***
Row 3			

Solution:

The VALUE ITERATION algorithm is fairly simple to use, check the example in the slides from Lecture 5, slide 20 (page 31) and onwards. The main idea is to generate utilities for all situations (states) the agent can be in, and use this for action selection. We can find the utilities following an iterative scheme (value iteration) by the following update formula:

$$U^{j+1}(s) := R(s) + \gamma \cdot \max_a \sum_{s'} P(s' \mid a, s) U^j(s').$$

If we initialize by assuming U0 = 0 for all states, we get:

Initialization:

	Column 1	*Column 2*	*Column 3*
Row 1	*0*	*0*	*0*
Row 2	*0*	*0*	*0*
Row 3	*0*	*0*	*0*

Next, the iterations using the formula above is applied. Note that for all states *s'* we have Uo(*s'*)=0, so the maximizations will all be over zeros. The first iteration is therefore identical to the immediate rewards, U1(s) = R(s), which gives the following result:

Iteration 1:

	Column 1	*Column 2*	*Column 3*
Row 1	*0*	*0*	*100*
Row 2	*0*	*-50*	*-10*
Row 3	*0*	*0*	*0*

To get U2, we need to apply the formula above yet another time. I'll only d this in detail for position (2,2), the others are calculated in the same way:

U2({2,2}) = R({2,2}) + gamma * max (expected utility if I go UP, expected utility if I go RIGHT, expected utility if I go DOWN, expected utility if I go LEFT)

So, we need to look at the expected utility of the different actions, first look at the situation UP:

- With probability 0.7 I come to {1,2}, which is valued at 0
- With probability 0.1 I come to {2,3}, which is valued at -10
- With probability 0.1 I come to {2,1}, which is valued at 0
- With probability 0.1 I come to {3,2}, which is valued at 0

The expected value of this is .7*0+.1*(-10)+.1*0+.1*0 = -1

Moving RIGHT:

- With probability 0.7 I come to {2,3}, which is valued at -10
- With probability 0.1 I come to {2,1}, which is valued at 0
- With probability 0.1 I come to {3,2}, which is valued at 0
- With probability 0.1 I come to {1,2}, which is valued at 0

The expected value of this is .7*(-10)+.1*0+.1*0+.1*0 = -7

DOWN and LEFT both have expectation -1, so we have

U2({2,2}) = R({2,2}) + gamma * max (expected utility if I go UP, expected utility if I go RIGHT, expected utility if I go DOWN, expected utility if I go LEFT) =
-50 + 0.1 * max(-1, -7, -1, -1) = -50 + .1*(-1) = -50.1

Similar progress gives all the entries in the table:

Iteration 2:

	Column 1	*Column 2*	*Column 3*
Row 1	*0*	*6.5*	*107.9*
Row 2	*-0.5*	*-50.1*	*-3.6*
Row 3	*0*	*-0.5*	*-0.1*

If you thought this was difficult, have a look at the lecture slides from Lecture 5, where a slightly more complex example was worked out in detail (2 typos have been fixed, so download a fresh copy). Note that in the example on the slides, the game is terminated as soon as the robot reaches the "gold state", thus the utility of {3,3} does not grow. Here this assumption is not made, and U({3,3}) will grow – but not infinitely – can you see why?)

Question 5 – Mixed questions (10%)

a) *Information Retrieval systems are typically evaluated using two criteria. Define the two criteria. Why is it insufficient to use only one of them to evaluate an IR system?*

Solution:

Precision and recall: Precision is the fraction of the retrieved set that is relevant, Recall is the fraction of the relevant documents that are in the solution set.

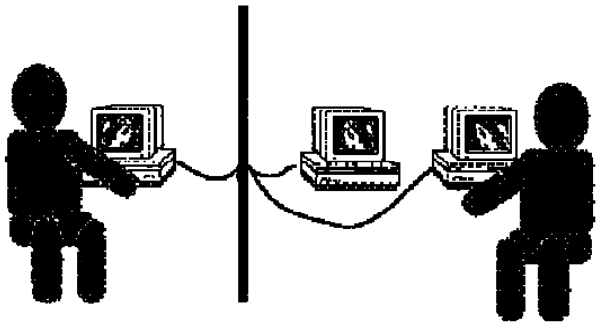
If we only have one of the two measurements, we can easily make a very "good" system by either choosing a result set which contains all documents in the document collection or which only contains a single document.

b) What is "*The Turing Test*", and how does it relate to the term "*Artificial Intelligence*"?

Solution:

Basic setup for the Turing Test:

- **Interrogator** in one room, **human** in another, **system** in a third. Interrogator **asks** questions; human and system **answer**
- After 5 minutes of discussion, the Interrogator **tries to guess** which is which
- The system has **passed** the Turing Test if the Interrogator fails 30% of the time



A discussion about "link to AI" should contain (your) thoughts on whether or not this is a good strategy for proof-of-concept for AI.