Information page for written examinations at Linköping University

Examination date	2012-01-11
Room (1) If the exam is given in different rooms you have to attach an information paper for each room and mark intended place	TER1
Time	8-12
Course code	TDDC17
Exam code	TEN1
Course name **Exam name**	Artificiell intelligens En skriftlig tentamen
Department	IDA
Number of questions in the examination	8
Teacher responsible/contact person during the exam time	Mariusz Wzorek
Contact number during the exam time	0703-88 71 22
Visit to the examination room approx.	ca kl. 10
Name and contact details to the course administrator (name + phone nr + mail)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Equipment permitted	Miniräknare/Hand Calculators
Other important information	
Which type of paper	valfritt

Linköpings Universitet
Institutionen för Datavetenskap
Patrick Doherty

Tentamen
TDDC17 Artificial Intelligence
11 january 2012 kl. 08-12

Points:

The exam consists of exercises worth 34 points.
To pass the exam you need 17 points.

Auxiliary help items:

Hand calculators.

Directions:

You can answer the questions in English or Swedish.
Use notations and methods that have been discussed in the course.
In particular, use the definitions, notations and methods in appendices 1-3.
Make reasonable assumptions when an exercise has been under-specified.
Begin each exercise on a new page.
Write only on one side of the paper.
Write clearly and concisely.

Jourhavande: Mariusz Wzorek, 0703887122. Mariusz will arrive for questions around 10.00.

1. Consider the following theory (where x, y and z are variables and history, lottery and john are constants):

$$\forall x([Pass(x, \text{history}) \wedge Win(x, \text{lottery})] \Rightarrow Happy(x)) \quad (1)$$

$$\forall x \forall y([Study(x) \vee Lucky(x)] \Rightarrow Pass(x, y)) \quad (2)$$

$$\neg Study(\text{john}) \wedge Lucky(\text{john}) \quad (3)$$

$$\forall x(Lucky(x) \Rightarrow Win(x, \text{lottery})) \quad (4)$$

   (a) Convert formulas (1) - (4) into clause form. **[1p]**

   (b) Prove that $Happy(\text{john})$ is a logical consequence of (1) - (4) using the resolution proof procedure. **[2p]**

   - Your answer should be structured using a resolution refutation tree (as used in the book).
   - Since the unifications are trivial, it suffices to simply show the binding lists at each resolution step.

2. Constraint satisfaction problems consist of a set of variables, a value domain for each variable and a set of constraints. A solution to a CS problem is a consistent set of bindings to the variables that satisfy the contraints. A standard backtracking search algorithm can be used to find solutions to CS problems. In the simplest case, the algorithm would choose variables to bind and values in the variable's domain to be bound to a variable in an arbitrary manner as the search tree is generated. This is inefficient and there are a number of strategies which can improve the search. Describe the following three strategies:

   (a) Minimum remaining value heuristic (MRV). **[1p]**

   (b) Degree heuristic. **[1p]**

   (c) Least constraining value heuristic. **[1p]**

   Constraint propagation is the general term for propagating constraints on one variable onto other variables. Describe the following:

   (d) What is the Forward Checking technique? **[1p]**

   (e) What is arc consistency? **[1p]**

3. The following questions pertain to the course article by Newell and Simon entitled *Computer Science as an Empirical Enquiry: Symbols and Search.*.

   (a) What is a *physical symbol system* (PSS) and what does it consist of? **[2p]**

   (b) What is the Physical Symbol System Hypothesis? **[1p]**

4. Use the Bayesian network in Figure 1 together with the conditional probability tables below to answer the following questions. Appendix 2 may be helpful to use.

   (a) Write the formula for the full joint probability distribution $P(A, B, C, D, E)$ in terms of (conditional) probabilities derived from the bayesian network below. **[1p]**

   (b) $P(a, \neg b, c, \neg d, e)$**[1p]**

   (c) $P(e \mid a, c, \neg b)$**[2p]**

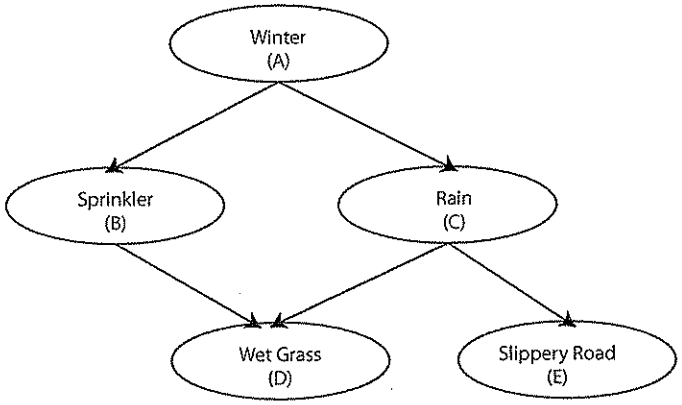


Figure 1: Bayesian Network Example

A	P(A)
T	.6
F	.4

A	B	$P(B \mid A)$
T	T	.2
T	F	.8
F	T	.75
F	F	.25

A	C	$P(C \mid A)$
T	T	.8
T	F	.2
F	T	.1
F	F	.9

B	C	D	$P(D \mid B, C)$
T	T	T	.95
T	T	F	.05
T	F	T	.9
T	F	F	.1
F	T	T	.8
F	T	F	.2
F	F	T	0
F	F	F	1

C	E	$P(E \mid C)$
T	T	.7
T	F	.3
F	T	0
F	F	1

5. A* search is the most widely-known form of best-first search. The following questions pertain to A* search:

   (a) Explain what an *admissible* heuristic function is using the notation and descriptions in (c). **[1p]**

   (b) Suppose a robot is searching for a path from one location to another in a rectangular grid of locations in which there are arcs between adjacent pairs of locations and the arcs only go in north-south (south-north) and east-west (west-east) directions. Furthermore, assume that the robot can only travel on these arcs and that some of these arcs have obstructions which prevent passage across such arcs.

   The *Manhattan distance* between two locations is the shortest distance between the locations ignoring obstructions. Is the Manhattan distance in the example above an admissible heuristic? Justify your answer explicitly. **[2p]**

   (c) Let $h(n)$ be the estimated cost of the cheapest path from a node n to the goal. Let $g(n)$ be the path cost from the start node n_0 to n. Let $f(n) = g(n) + h(n)$ be the estimated cost of the cheapest solution through n.

   Provide a sufficiently rigid proof that A* using either tree-search (seminar slides) or graph-search (in book) is optimal if $h(n)$ is admissible. If possible, use a diagram to structure the contents of the proof. **[2p]**

6. Domain-independent heuristics play an important part in many planning algorithms. Such heuristics are often based on *relaxing* the original planning problem.

   (a) Explain the concept of relaxing a problem and how this helps us find useful heuristics.**[2p]**

   (b) Provide a concrete example of this technique using an action schema in a planning domain of your choice. **[2p]**

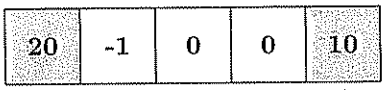
7. Modeling actions and change in incompletely represented, dynamic worlds is a central problem in knowledge representation. The following questions pertain to reasoning about action and change.

   (a) What is the frame problem? Use the Wumpus world to provide a concrete example of the problem. **[2p]**

   (b) What is the ramification problem? Use the Wumpus world to provide a concrete example of the problem. **[2p]**

   (c) What is nonmonotonic logic? How can it be used to provide solutions to the frame problem? **[2p]**

8. A learning agent moves through the *unknown* environment below in episodes, always starting from state 3 until it reaches the gray terminal states in either 1 or 5. In the terminal states the learning episode ends and no further actions are possible. The numbers in the squares represent the reward the agent is given in each state. Actions are {*West, East*} and the transition function is *deterministic* (no uncertainty). The agent uses the Q-learning update, as given in appendix 3, to attempt to calculate the utility of states and actions. It has a discount factor of 0.9 and a fixed learning rate of 0.5. The estimated values of the Q-function based on the agent's experience so far is given in the table below.
   a) Construct the policy function for an agent that just choses the action which it thinks maximizes utility based on its current experience so far, is this *policy* optimal given the problem parameters above? **[1p]**
   b) Is the agent guaranteed to eventually always find the optimal policy with the behavior above? Explain one approach that could be used to improve the agent's behavior. **[1p]**
   c) For whatever reason the agent now moves west from the start position to the terminal state. Update the Q-function with the new experience and extract a new policy for the agent under the same assumptions as in a). Show the steps of your calculation. **[2p]**



State	Action	Utility
2	W	10
2	E	0
3	W	-0.5
3	E	0
4	W	0
4	E	5

Appendix 1

Converting arbitrary wffs to clause form

1. Eliminate implication signs.
2. Reduce scopes of negation signs.
3. Standardize variables within the scopes of quantifiers (Each quantifier should have its own unique variable).
4. Eliminate existential quantifiers. This may involve introduction of Skolem constants or functions.
5. Convert to prenex form by moving all remaining quantifiers to the front of the formula.
6. Put the matrix into conjunctive normal form. Two useful rules are:
   - $\omega_1 \vee (\omega_2 \wedge \omega_3) \equiv (\omega_1 \vee \omega_2) \wedge (\omega_1 \vee \omega_3)$
   - $\omega_1 \wedge (\omega_2 \vee \omega_3) \equiv (\omega_1 \wedge \omega_2) \vee (\omega_1 \wedge \omega_3)$
7. Eliminate universal quantifiers.
8. Eliminate $\wedge$ symbols.
9. Rename variables so that no variable symbol appears in more than one clause.

Skolemization

Two specific examples. One can of course generalize the technique.
$\exists x P(x)$:
Skolemized: $P(\mathsf{c})$ where c is a fresh constant name.
$\forall x_1, \ldots, x_k, \exists y P(y)$:
Skolemized: $P(\mathsf{f}(x_1, \ldots, x_k))$, where f is a fresh function name.

Appendix 2

A generic entry in a joint probability distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. The notation $P(x_1, \ldots, x_n)$ can be used as an abbreviation for this.

The chain rule states that any entry in the full joint distribution can be represented as a product of conditional probabilities:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) \tag{5}$$

Given the independence assumptions implicit in a Bayesian network a more efficient representation of entries in the full joint distribution may be defined as

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \mathit{parents}(X_i)), \tag{6}$$

where $\mathit{parents}(X_i)$ denotes the specific values of the variables in $\mathit{Parents}(X_i)$.

Recall the following definition of a conditional probability:

$$\mathbf{P}(X \mid Y) = \frac{\mathbf{P}(X \wedge Y)}{\mathbf{P}(Y)} \tag{7}$$

The following is a useful general inference procedure:

Let X be the query variable, let $\mathbf{E}$ be the set of evidence variables, let $\mathbf{e}$ be the observed values for them, let $\mathbf{Y}$ be the remaining unobserved variables and let α be the normalization constant:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \tag{8}$$

where the summation is over all possible $\mathbf{y}$'s (i.e. all possible combinations of values of the unobserved variables $\mathbf{Y}$).

Equivalently, without the normalization constant:

$$\mathbf{P}(X \mid \mathbf{e}) = \frac{\mathbf{P}(X, \mathbf{e})}{\mathbf{P}(\mathbf{e})} = \frac{\sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})}{\sum_{\mathbf{x}} \sum_{\mathbf{y}} \mathbf{P}(\mathbf{x}, \mathbf{e}, \mathbf{y})} \tag{9}$$

Appendix 3: MDPs and Reinforcement Learning

Value iteration

Value iteration is a way to compute the utility $U(s)$ of all states in a *known* environment (MDP) under the optimal policy $\pi^*(s)$.

It is defined as:

$$U(s) = R(s) + \gamma \max_{\alpha \in \mathcal{A}} \sum_{s'} P(s'|s,a)U(s') \tag{10}$$

where $R(s)$ is the reward function, s and s' are states, γ is the discount factor. $P(s'|s,a)$ is the state transition function, the probability of ending up in state s' when taking action a in state s.

Value iteration is usually done by initializing $U(s)$ to zero and then sweeping over all states updating $U(s)$ in several iterations until it stabilizes.

A policy function defines the behavior of the agent. Once we have the utility function of the optimal policy from above, $U_{\pi^*}(s)$, we can easily extract the optimal policy $\pi^*(s)$ itself by simply taking the locally best action in each state

$$\pi^*(s) = \arg\max_{\alpha \in \mathcal{A}} \sum_{s'} P(s'|s,a)U_{\pi^*}(s')$$

Q-learning

Q-learning is a model-free reinforcement learning approach for *unknown* environments and can be defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R(s_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

where s_t and s_{t+1} are states in a sequence, $Q(s_t, a_t)$ is the estimated utility of taking action a_t in state s_t, γ is the discount factor and α is the learning rate.

The Q-function is usually initialized to zero, and each time the agent performs an action it can be updated based on the observed sequence $..., s_t, R(s_t), a_t, s_{t+1}, R(s_{t+1})....$ The Q-function can then be used to guide agent behavior, for example by extracting a policy like in value iteration above.