Ciência de Computadores

Exame de Sistemas Inteligentes

Segunda Parte (duração: 1 hora)

Data: 11 de Junho de 2013

1) Numa rede Bayesiana:

(a) nós são condicionalmente dependentes dados os pais

(b) nós são condicionalmente independentes dados os pais

(c) nós são condicionalmente dependentes dados os filhos

(d) nós são condicionalmente independentes dados os filhos

(e) nós são sempre condicionalmente dependentes

(f) nós são sempre condicionalmente independentes

2) O resultado da unificação entre os termos `maisNovo(X,pai(X))` com o termo `maisNovo(X,X)`, em Prolog, onde os nomes de variáveis são iniciados por letra maiúscula e as constantes por letra minúscula, é:

(a) `X=pai(X)`

(b) `X=X`

(c) é um ciclo

(d) não é um ciclo, mas os dois termos não unificam

3) Qual é a melhor interpretação para o programa abaixo:

```
p(Xs,Ys) :- p(Xs,[],Ys).

p([X|Xs],R,Ys) :- p(Xs,[X|R],Ys).
p([],Ys,Ys).
```

(a) p é um predicado que implementa o conceito de caminho, onde cada elemento do caminho é guardado numa lista

(b) p é um predicado que implementa a concatenação de duas listas

(c) p é um predicado que implementa o reverso de uma lista

(d) p é um predicado que implementa a adição de novos elementos numa lista

(e) nenhuma das alternativas anteriores corresponde à interpretação de p

4) Dada a fórmula de cálculo de ganho de informação abaixo, indique qual é o atributo na Tabela 1 que será escolhido como raiz da árvore de decisão induzida.

$$\text{ganho de informação}(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Restante(A)$$

$$Restante(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$$

v é o número de valores do atributo A, p é o número de instâncias de uma classe e n é o número de instâncias da outra classe.

Tabela 1: Tabela de atributos do problema de indução de árvore de decisão

Instância	Tempo	Temperatura	Umidade	Vento	Classe
1	ensolarado	quente	alta	falso	N
2	ensolarado	quente	alta	verdadeiro	N
3	nublado	quente	alta	falso	P
4	chuvoso	médio	alta	falso	P
5	chuvoso	frio	normal	falso	P
6	chuvoso	frio	normal	verdadeiro	N

(a) Tempo

(b) Temperatura

(c) Umidade

(d) Vento

(e) dois atributos têm a mesma chance de serem escolhidos como raiz da árvore de decisão

(f) três atributos têm a mesma chance de serem escolhidos como raiz da árvore de decisão

(g) qualquer atributo tem a mesma chance de ser escolhido como raiz da árvore de decisão

5) No seguinte resultado produzido pelo pacote WEKA, usou-se um algoritmo de árvore de decisão para gerar um classificador de tipos de plantas. As plantas podem ser do tipo T1, T2 ou T3.

```
 a  b  c   <-- classified as
49  2  0 |  a = T1
 2 47  3 |  b = T2
 0  2 48 |  c = T3
```

O número de plantas do tipo T1, T2 e T3, respectivamente, é:

(a) 49, 47, 48

(b) 51, 52, 50

(c) 51, 51, 51

(d) nenhuma das alternativas anteriores está correta

6) O algoritmo *hill climbing*:

(a) nunca encontra uma solução ótima para um problema

(b) sempre encontra uma solução ótima para um problema se a função de custo for negativa

(c) sempre encontra uma solução ótima para um problema se a função de custo for positiva

(d) pode ou não encontrar uma solução ótima

(e) nenhuma das afirmativas anteriores está correta

7) A diferença entre o algoritmo *random restart hill climbing* e o *simulated annealing* é:

(a) *random restart hill climbing* aceita uma solução pior se souber que mais tarde pode enontrar uma solução ótima

(b) *simulated annealing* aceita uma solução pior com uma certa probabilidade

(c) *random restart hill climbing* aceita uma solução pior com uma certa probabilidade

(d) não há diferenças fundamentais entre estes dois algoritmos

8) Quais são as condições para que um plano seja uma solução?

(a) todas as suas ações precisam ter as pós-condições e pré-condições resolvidas, além de ter todas as ameaças resolvidas. Além disto, também precisa ter todas as variáveis instanciadas

(b) todas as suas ações precisam ter as pós-condições resolvidas, além de ter todas as ameaças resolvidas. Além disto, também precisa ter todas as variáveis instanciadas

(c) todas as suas ações precisam ter as pré-condições resolvidas, além de ter todas as ameaças resolvidas. Além disto, também precisa ter todas as variáveis instanciadas

(d) todas as suas ações precisam ter as pós-condições resolvidas, além de ter todas as ameaças resolvidas, mas não precisa ter todas as variáveis instanciadas

(e) todas as suas ações precisam ter as pré-condições resolvidas, além de ter todas as ameaças resolvidas, mas não precisa ter todas as variáveis instanciadas

(f) nenhuma das condições acima reúne os critérios para que um plano seja uma solução

9) Dados os pseudo-códigos abaixo, qual deles implementa corretamente o algoritmo de poda alfa-beta, relativamente ao procedimento de MAX?

(a)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(b)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v <= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(c)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(d)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
```

```
if TERMINAL_TEST(state) then return UTILITY(state)
v <- -infinito
for a, s in SUCCESSORS(state) do
  v <- MAX(v, MIN-VALUE(s,alfa,beta))
  if (v >= alfa ) then return v % momento da poda
  beta <- MAX(beta,v)
return v
```

(e) nenhuma das alternativas anteriores implementa corretamente o passo de MAX da poda alfa-beta

10) Na implementação do agendamento de vôos com a sintaxe:

```
timetable(edinburgh,london,
[ 9:40/10:50/ba4733/alldays,
13:40/14:50/ba4773/alldays,
19:40/20:50/ba4833/[mo,tu,we,th,fr,su]]).
```

qual é a melhor definição de operadores que permite a correta unificação de um termo 9:40/10:50/ba4733 com o termo X/Y/Z?

(a) `:- op(500,xfy,/). :- op(400,xfy,:).`

(b) `:- op(400,xfy,/). :- op(500,xfy,:).`

(c) `:- op(500,xfx,/). :- op(400,xfy,:).`

(d) `:- op(500,yfx,/). :- op(400,xfy,:).`

(e) `:- op(500,xfy,/). :- op(400,yfx,:).`

(f) `:- op(400,xfy,/). :- op(500,yfx,:).`

(g) `:- op(400,yfx,/). :- op(500,xfy,:).`

(h) nenhuma das formas acima permite a unificação correta