Qualifying Examination 2019 Artificial Intelligence

There are six questions on this exam. You must answer them all. All answers must be in the test booklets. All questions have the same value for grading.

Your grade depends on the correctness, completeness and style of your answers.

1. Consider the problem of using a truck to move boxes among warehouses in Manhattan. Boxes can be loaded on the truck or unloaded from the truck. The truck can drive from one warehouse to another, and can carry at most 10 boxes at a time.

There is one truck and 100 boxes. There are five warehouses: A, B, C, D and E. The truck starts at warehouse A. Initially there are 25 boxes in each of the warehouses B, C, D, and E (the warehouses can each hold an unlimited number of boxes). The goal is to move all the boxes to warehouse A. The distances between the warehouses are (in kilometers):

	A	**B**	**C**	**D**	**E**
A	0	1	2	4	9
B	1	0	3	6	8
C	2	3	0	4	4
D	4	6	4	0	5
E	9	8	4	5	0

You will formulate this as a planning problem using STRIPS (PDDL) action schemas.

(a) Describe the start and goal states for this problem using STRIPS notation. You don't have to write out *every* necessary literal (all 100 boxes, for example) but indicate what literals are needed and give a few examples.
(b) Write action schemas for loading a box on the truck, for driving the truck from one warehouse to another, and for unloading a box.
(c) Using your schemas from part (b), show how resolution is used to load one box on the truck at warehouse C, drive it to warehouse E then unload it.
(d) Give an admissible A* heuristic to guide the search, so as to minimize the total distance traveled by the truck.

(a)
Init(At(B1,WB) & At(B2,WB) … & At(B26,WC) … & At(B51,WD) … & At(B76,WE) & At(T,WA) & Dist(WA,WB,1) … & Dist(WE,WD,5) & Truck(T) & Box(B1) & … & Box(B100) & Warehouse(WA) & … & Warehouse(WE) & Numboxes(T,0))

Goal(At(B1,WA) & … & At(B100,WA))

(b)

Action(Load(b,t,w)
PRECOND: Box(b) & Truck(t) & Warehouse(w) & At(t,w) & At(b,w) & Numboxes(T,n) & <(n,10)
EFFECT: In(b,t) & -At(b,w) & Numboxes(T,n+1))

Action(Drive(t,x,y)
PRECOND: Truck(t) & Warehouse(x) & Warehouse(y) & At(t,x)
EFFECT: At(t,y) & -At(t,x))

Action(Unload(b,t)
PRECOND: Box(b) & Truck(t) & Warehouse(w) & At(t,w) & In(b,t) & Numboxes(T,n) & >(n,0)
EFFECT: -In(b,t) & At(b,w) & Numboxes(T,n-1))

(c)
Load(B1,T,WB)
Box(B1) & Truck(T) & Warehouse(WB) & At(T,WB) & At(B1,WB) & Numboxes(T,0)
-> Box(B1) & Truck(T) & Warehouse(WB) & At(T,WB) & In(B1,T) & Numboxes(T,1)

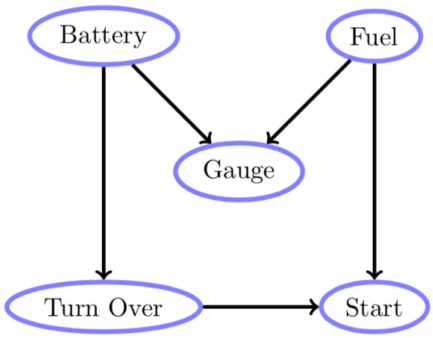
Drive(T,WB,WA)
Box(B1) & Truck(T) & Warehouse(WB) & Warehouse(WA) & At(T,WB) & In(B1,T)
-> Box(B1) & Truck(T) & Warehouse(WB) & Warehouse(WA) & At(T,WA) & In(B1,T)

Unload(B1, T)
Box(B1) & Truck(T) & Warehouse(WA) & At(T,WA) & In(B1,T) & Numboxes(T,1)
-> Box(B1) & Truck(T) & Warehouse(WA) & At(T,WA) & At(B1,WA) & Numboxes(T,0)

(d)
One possible heuristic is to divide the number of remaining boxes at each warehouse (other than A) by 10 to get the minimum number of trips necessary from that warehouse, and then add up those numbers times the distance from each warehouse to A.

2. Bayes Nets

The network below concerns the probability of a car starting.



You know the following facts (b = battery, s = start, f = fuel, t = turn over, g = gauge):

p(b = bad) = 0.02
p(g = empty | b = good, f = not empty) = 0.04
p(g = empty | b = bad, f = not empty) = 0.1
p(t = false | b = good) = 0.03
p(s = false | t = true, f = not empty) = 0.01
p(s = false | t = false, f = not empty) = 1.0
p(f = empty) = 0.05
p(g = empty | b = good, f = empty) = 0.97
p(g = empty | b = bad, f = empty) = 0.99
p(t=false | b=bad)=0.98
p(s = false | t = true, f = empty) = 0.92
p(s = false | t = false, f = empty) = 0.99

Calculate P (f = empty | s = false), the probability of the fuel tank being empty given that the car does not start.

Answer:

Using Bayes' rule, we have:
P(f=empty | s = false) = P(s=false | f = empty) P(f=empty) / P(s = false)

We know P(f=empty) = .05 from the given, so we need compute the first term P(s=false | f=empty), and then P(s=false). To figure out s we need to sum over its predecessors, f and t, since those probabilities are given.

P(s = false | f = empty) = P(s = false | f=empty, t=true) P (t=true) +
P(s=false | f=empty, t=false) P(t=false)

To know this, we need to figure out P(t=true) and P(t=false). To figure out t we need to sum over its predecessor, which is b.

P(t = false) = P(t=false | b=good) P(b=good) + P(t=false | b=bad) P(b=bad)
= .03 * .98 + .98 * .02 = .049
P(t = true) = .951

Plug these back into the equation for P(s=false | f=empty):

P(s=false | f=empty) = .92 * .951 + .99 * .049 = .87492 + .04851 = .92343

To compute P(s=false) we'll also need P(s=false | f=not empty) to combine with the above.

P(s=false | f=not empty) = P(s=false | f=not empty, t=true) P(t=true) +
P(s=false | f=not empty, t=false) P(t=false)
= .01 * ,951 + 1 * .049 = .05851

Combine them:
P(s=false) = P(s=false | f=empty) P(f=empty) + P(s=false | f=not empty) P(f=not empty)
= ,92343 * .05 + .05851 * .95 = .0461715 + .0555845 = .1017560

Plug back into the first equation:
P(f=empty | s=false) = .92343 * .05 / .1017560 = .0461715 / .1017560 = .453747

3. Consider a village where all villagers are either healthy or have a fever and only the village doctor can determine whether each has a fever. The doctor diagnoses fever by asking patients how they feel. The villagers may only answer that they feel normal, dizzy, or cold. There are two states, "Healthy" and "Fever", but the doctor cannot observe them directly. The prior probability of being Healthy is 0.6.

On each day, there is a certain chance that the patient will tell the doctor he/she is "normal", "cold", or "dizzy", depending on their health condition. If the patient is Healthy, the probability for “normal” is 0.5, for “cold” is 0.4, and for “dizzy” is 0.1. If the patient has a Fever, the probability for “normal” is 0.1, for “cold” is 0.3, and for “dizzy” is 0.6.

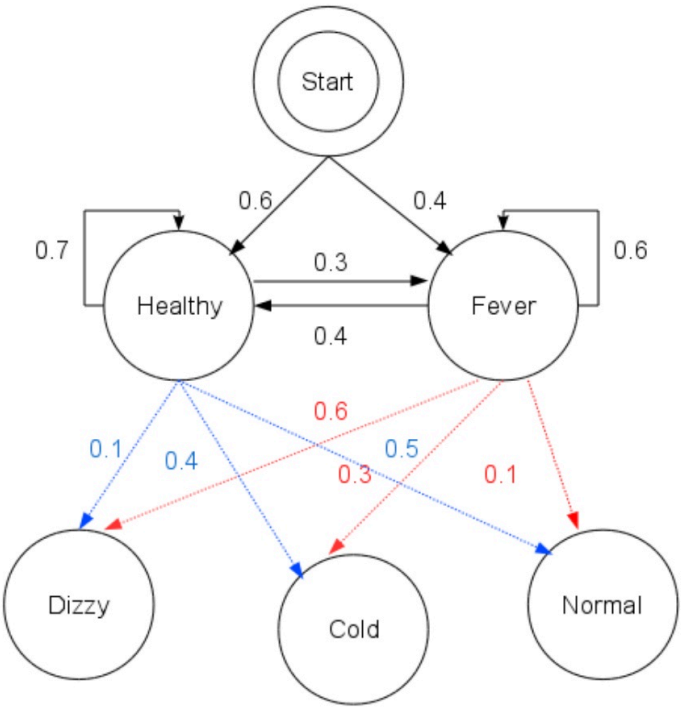
A Healthy patient remains Healthy on the next day 70% of the time, and a patient with a Fever has a Fever the next day 60% of the time.

a. Show how this can be formulated as an HMM, including the probability tables.

b. A patient visits the doctor three days in a row and tells the doctor that that on the first day he feels normal, on the second day he feels cold, on the third day he feels dizzy. Use the filtering algorithm to determine the probability of being in each state for t = 1, 2, 3.

c. What is the time complexity of filtering in an HMM?

Answer:



Healthy: 0.30000 0.08400 0.00588
Fever: 0.04000 0.02700 0.01512
The steps of states are Healthy Healthy Fever with highest probability of 0.01512

All eight possible state trajectories:

Healthy - Healthy - Healthy:

P(Healthy) = 0.6 * P(Normal I Healthy) = 0.5 -> answer = 0.3

P(Healthy I Healthy) = 0.7 * P(Cold I Healthy) = 0.4 = 0.28 * previous answer = 0.3 = 0.084

P(Healthy I Healthy) = 0.7 * P(Dizzy I Healthy) = 0.1 = 0.07 * previous answer = **0.00588**

Healthy - Healthy - Fever:

P(Healthy) = 0.6 * P(Normal I Healthy) = 0.5 -> answer = 0.3

P(Healthy I Healthy) = 0.7 * P(Cold I Healthy) = 0.4 = 0.28 * previous answer = 0.3 = 0.084

P(Fever I Healthy) = 0.3 * P(Dizzy I Fever) = 0.6 = 0.18 * previous answer = **0.01512**

Healthy - Fever - Healthy:

P(Healthy) = 0.6 * P(Normal | Healthy) = 0.5 -> answer = 0.3

P(Fever | Healthy) = 0.3 * P(Cold | Fever) = 0.3 = 0.09 * previous answer = 0.3 = 0.027

P(Healthy | Fever) = 0.4 * P(Dizzy | Healthy) = 0.1 = 0.04 * previous answer = **0.0108**

Healthy - Fever - Fever:

P(Healthy) = 0.6 * P(Normal | Healthy) = 0.5 -> answer = 0.3

P(Fever | Healthy) = 0.3 * P(Cold | Fever) = 0.3 = 0.09 * previous answer = 0.3 = 0.027

P(Fever | Fever) = 0.6 * P(Dizzy | Fever) = 0.6 = 0.036 * previous answer = **0.00972**

Fever - Healthy - Healthy:

P(Fever) = 0.4 * P(Normal | Fever) = 0.1 -> answer = 0.04

P(Healthy | Fever) = 0.4 * P(Cold | Healthy) = 0.4 = 0.16 * previous answer = 0.3 = 0.048

P(Healthy | Healthy) = 0.7 * P(Dizzy | Healthy) = 0.1 = 0.07 * previous answer = **0.00336**

Fever - Healthy - Fever:

P(Fever) = 0.4 * P(Normal | Fever) = 0.1 -> answer = 0.04

P(Healthy | Fever) = 0.4 * P(Cold | Healthy) = 0.4 = 0.16 * previous answer = 0.3 = 0.048

P(Fever | Healthy) = 0.3 * P(Dizzy | Fever) = 0.6 = 0.18 * previous answer = **0.00864**

Fever - Fever - Healthy:

P(Fever) = 0.4 * P(Normal | Fever) = 0.1 -> answer = 0.04

P(Fever | Fever) = 0.6 * P(Cold | Fever) = 0.3 = 0.18 * prev answer = 0.0072

P(Healthy | Fever) = 0.4 * P(Dizzy | Healthy) = 0.1 = 0.04 * prev answer = **0.00288**

Fever - Fever - Fever:

P(Fever) = 0.4 * P(Normal | Fever) = 0.1 -> answer = 0.04

P(Fever | Fever) = 0.6 * P(Cold | Fever) = 0.3 = 0.18 * prev answer = 0.0072

P(Fever | Fever) = 0.6 * P(Dizzy | Fever) = 0.6 = 0.36 * prev answer = **0.002592**

The highest probability for Healthy on day 1 is 0.3 (shared by 4 cases)
and the highest for Fever is 0.04 (shared by 4 cases).
The highest probability for Healthy on day 2 is 0.084 (2 cases)
and the highest for Fever is 0.027 (2 cases).
The highest probability for Healthy on day 3 is 0.00588
and the highest for Fever is 0.01512.

4. Markov Decision Processes

The Bellman equation:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

a. An MDP may have cyclic paths in its states. What keeps us from performing infinite calculations when we solve such MDPs?

Answer: discount < 1

b. What is the time complexity of solving the Bellman equation for a system with n states?

Answer: O(n^3)

c. Is there an optimal policy for every MDP?

Answer: Yes

d. Why are iterative solution methods necessary for the Bellman equation?

Answer: It's nonlinear so it cannot be solved in closed form.

e. Name three iterative solution methods for the Bellman equation.

Answer: policy iteration, value iteration, Q-learning

5. In the following, we describe whether a person is ill. We use a representation based on conjunctive constraints (three per subject) to describe each individual person. These constraints

are “running nose”, “coughing”, and “reddened skin”, each of which can take the value true (‘+’) or false (‘–’). We say that somebody is ill, if he is coughing and has a runny nose — each single symptom individually does not mean that the person is ill.

a. Specify the space of hypotheses that is being managed by the version space approach. To do so, arrange all hypotheses in a graph structure using the more-specific-than relation. I hypotheses are vectors of constraints, denoted by <N, C, R> with N, C, R = {–, +, ∅, ∗}.
b. Apply the candidate elimination (CE) algorithm to the sequence of training examples specified in the table and name the contents of the sets S and G after each step.

Training Example	N (running nose)	C (coughing)	R (reddened skin)	**Classification**
d_1	+	+	+	positive (ill)
d_2	+	+	–	positive (ill)
d_3	+	–	+	negative (healthy)
d_4	–	+	+	negative (healthy)
d_5	–	–	+	negative (healthy)
d_6	–	–	–	negative (healthy)

c. We now extend the number of constraints used for describing training instances by one additional constraint named “fever”. We say that somebody is ill, if he has a running nose and is coughing (as we did before), or if he has fever.
What steps does the CE algorithm perform now, and what is the result, given the training examples specified below?

Training Example	N (running nose)	C (coughing)	R (reddened skin)	F (fever)	**Classification**
d_1	+	+	+	–	positive (ill)
d_2	+	+	–	–	positive (ill)
d_3	–	–	+	+	positive (ill)
d_4	+	–	–	–	negative (healthy)
d_5	–	–	–	–	negative (healthy)
d_6	–	+	+	–	negative (healthy)

d. What is the reason for this result? What modification to the hypothesis space will fix things?

Answer:

Start (init): G = {<∗ ∗ ∗>}, S = {<∅ ∅ ∅>}

foreach d ∈ D do

d1 = [<+ + +>, pos] ⇒ G = {<∗ ∗ ∗>}, S = {<+ + +>}

d2 = [<+ + –>, pos] ⇒ G = {<∗ ∗ ∗>}, S = {<+ + ∗>}

d3 = [<+ – +>, neg]

no change to S: S = {<+ + ∗>}

specializations of G: G = {<– ∗ ∗>,<∗ + ∗>,<∗ ∗ –>}

there is no element in S that is more specific than the first and third element of G

→ remove them from G ⇒ G = {<∗ + ∗>}

d4 = [<– + +>, neg]

no change to S: S = {<+ + *>}
specializations of G: G = {<+ + *>,<* + –>} I there is no element in S that is more specific than the second element of G → remove it from G ⇒ G = {<+ + *>}
d5 = [<– – +>, neg] ⇒ Both, G = {<+ + *>} and S = {<+ + *>} are consistent with d5.
d6 = [<– – –>, neg] ⇒ Both, G = {<+ + *>} and S = {<+ + *>} are consistent with d6.

Part d:
Initially: S = {<∅∅∅∅>}, G = {<* * * *>}
d1 = [<+ + +–>, pos] ⇒ S = {<+ + + –>}, G = {<* * * *>}
d2 = [<+ + ––>, pos] ⇒ S = {<+ + * –>}, G = {<* * * *>}
d3 = [<– – ++>, pos] ⇒ S = {<* * * *>}, G = {<* * * *>} → We already arrive at S = G.
d4 = [<+ – ––>, neg] ⇒ S = {<* * * *>}, G = {<* * * *>}
Now, S becomes empty since <* * * *> is inconsistent with d4 and is removed from S.
G would be specialized to {<– * * *>,<* + * *>,<* * + *>,<* * * +>}. But it is required that at least one element from S must be more specific than any element from G. → This requirement cannot be fulfilled since S = ∅. ⇒ G = ∅.

Reason: The informally specified target concept of an “ill person” represents a disjunctive concept.
Fix by using disjunctive concepts.

6. Q-learning

a. Q-learning is called “model-free”. Why?

b. Q-learning is called “off-policy”. Why?

c. Suppose we create a very optimistic prior distribution of values over the states. How does this affect a Q-learning agent?

Answers:

a. Q-learning does not use the model, i.e. the transition probabilities and the reward function.
b. An off-policy learning algorithm learns the optimal policy independently of the agent’s actions. Q-learning does this.
c. Increases exploration.