6.4110
Representation, Inference and Reasoning in AI

Quiz 2

Solutions

March 18, 2026

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

You are permitted to use a single sheet of paper with notes on (both sides). You may not use a calculator. **Box all answers** for free response questions.

Name: _____

MIT email: _____

Question	Points	Score
1	20	
2	20	
3	30	
4	18	
5	12	
Total:	100	

6.4110
Quiz 2, March 18, 2026

1 Continuing resolution

1. We are going to consider several temporal stochastic processes. For each one, we supply the transition model $P(s_t \mid s_{t-1})$ and observation model $P(o_t \mid s_t)$ in the form of Python-ish functions that provide samples from the relevant distributions. The `normal_variate(m, s)` procedure draws a random sample from a normal distribution with mean `m` and variance `s**2`.

Process A

```
STATES = range(10)
def transition_A(s):
    if rand() < 0.5:
        return max(0, s - 1)
    else:
        return min(9, s + 1)

def observe_A(s_t):
    return random_choice([s, s-1, s+1])
```

Process B

```
def transition_B(x):
    if rand() < 0.5:
        return x - 1 + normal_variate(0, 1)
    else:
        return x + 1 + normal_variate(0, 1)

def observe_B(x):
    return normal_variate(x+1, 2)
```

Process C

```
def transition_C(x):
    return x**2 + normal_variate(0, 1)

def observe_C(x):
    return normal_variate(x, 2)
```

Process D

```
def transition_D(x):
    return x + normal_variate(0, 1)

def observe_D(x):
    return normal_variate(x, 0.01)
```

Process E

```
def transition_E(x, x_prev):
    return = 2*x + x_prev + normal_variate(0, 1)

def observe_E(x):
    return normal_variate(x, 1)
```

(a) (15 points) We would like to build filters for each of these models. We are considering three modeling and filtering methods: HMM filter, Kalman filter, and particle filter. For each process, indicate which of these methods could be used effectively to model the problem. For each method that *you do not check*, explain in one sentence why it is not suitable.

i. Process A √ **HMM filter** ◯ Kalman filter √ **Particle filter**

> **Solution:** Kalman filter is not suitable because it assumes continuous linear-Gaussian states and dynamics, while this process has discrete states.

ii. Process B ◯ HMM filter ◯ Kalman filter √ **Particle filter**

> **Solution:** HMM is not suitable because it assumes a finite discrete state space while this process has a continuous state. Kalman filter is not suitable because the transition distribution is bi-modal and therefore not Gaussian.

iii. Process C ◯ HMM filter ◯ Kalman filter √ **Particle filter**

> **Solution:** HMM is not suitable because the state is continuous rather than discrete. Kalman filter is not suitable because the dynamics $x_t = x_{t-1}^2 + \epsilon$ are nonlinear.

iv. Process D ◯ HMM filter √ **Kalman filter** √ **Particle filter**

> **Solution:** HMM is not suitable because it assumes a finite discrete state space while this process has a continuous state.

v. Process E ◯ HMM filter ◯ Kalman filter √ **Particle filter**

> **Solution:** HMM is not suitable because it assumes a finite discrete state space while this process has a continuous state. Kalman filter assumes dependence only on the previous state.
> Also okay to say Kalman works if you put the last two x's in the state space.

(b) (5 points) We are interested in using a particle-filter approximation to process D. Assume our initial distribution is uniform on the interval $[1, 1000]$. Describe **concretely** what difficulty we are likely to encounter.

> **Solution:** The main difficulty is particle degeneracy. Since the initial particles are spread uniformly over the very large interval $[1, 1000]$ but the observation model is extremely peaked, after incorporating an observation almost all particles will receive nearly zero weight and only a very small number of particles near the observed value will have significant weight. After resampling, the particle set will collapse to very few distinct particles, so the filter loses diversity and gives a poor approximation unless we use a very large number of particles.

2 Lattice entertain you!

2. You are modeling the state of a 3D solid as an equally spaced 3D grid of random variables, v_{ijk} where each i, j, k is in $\{0, \dots M-1\}$. Each location can be in **one of two** states.

   (a) (3 points) How many possible states does this system have?

   Solution: There are M^3 random variables, and each variable has 2 possible states, so the full joint has size
   $$2^{M^3}.$$

   You use a Markov Random Field (MRF) to model a distribution over possible states of your solid. You include a pairwise potential (factor) between each neighboring pair of v's (which share two indices in common).

   (b) (4 points) What is the approximate time required to compute an exact marginal distribution on one of the variables by computing the joint and marginalizing? Choose one answer and provide a short justification.
   ◯ $O((2^{M^3})^2)$ ◯ $O(M^6)$ ◯ $O(M^5)$ √ $O(2^{M^3})$ ◯ $O(3^{M^2})$

   Solution: The full joint distribution has 2^{M^3} entries. If we compute an exact marginal by explicitly forming the joint and then summing out all other variables, the computation is dominated by the size of that joint table. Thus the computation time is approximately
   $$O(2^{M^3}).$$

   (c) (4 points) What is the approximate computation time for one iteration of loopy belief propagation? Choose one answer and provide a short justification.
   ◯ $O(1)$ ◯ $O(2^{M^3})$ √ $O(M^3)$ ◯ $O(3^{M^2})$ ◯ $O(M^2)$

   Solution: There are M^3 variables in the grid, each with a constant number of neighbors (at most 6), so the number of edges is $O(M^3)$.
   One iteration of loopy belief propagation updates one message along each directed edge. Each message update takes constant time because variables are binary (for each of the 2 possible values of the receiving variable, we sum over the 2 possible values of the sending variable).
   Therefore, the total computation time for one full iteration is
   $$O(M^3).$$

(d) (5 points) Suppose the pairwise potential between neighboring variables is given by

a	b	$\phi(a,b)$
0	0	3
0	1	2
1	0	2
1	1	3

Assume the current values of the six neighbors of V_{345} are

$$V_{245}=1, \quad V_{445}=1, \quad V_{335}=0, \quad V_{355}=0, \quad V_{344}=0, \quad V_{346}=1.$$

What probability distribution would you use to sample a new value of V_{345} in Gibbs sampling?

Solution: ** Update because I changed the factor table to be symmetric **
In Gibbs sampling, we resample V_{345} from its conditional distribution given its Markov blanket, which here consists of its six neighbors. Thus $P(v_{345}=x \mid \text{rest}) \propto \prod_{u\in\mathcal{N}(345)} \phi(x,u)$ for $x \in \{0,1\}$.
The six neighbor values are three 1's and three 0's:

$$V_{245}=1, \quad V_{445}=1, \quad V_{335}=0, \quad V_{355}=0, \quad V_{344}=0, \quad V_{346}=1.$$

So for $v_{345}=0$,

$$P(v_{345}=0 \mid \text{rest}) \propto \phi(0,1)\phi(0,1)\phi(0,0)\phi(0,0)\phi(0,0)\phi(0,1) = 2^3 \cdot 3^3 = 216.$$

For $v_{345}=1$,

$$P(v_{345}=1 \mid \text{rest}) \propto \phi(1,1)\phi(1,1)\phi(1,0)\phi(1,0)\phi(1,0)\phi(1,1) = 4^3 \cdot 1^3 = 64.$$

Normalizing,

$$P(v_{345}=0 \mid \text{rest}) = \frac{216}{216+64} = \frac{27}{35}, \qquad P(v_{345}=1 \mid \text{rest}) = \frac{64}{216+64} = \frac{8}{35}.$$

So the Gibbs sampling distribution is

$$v_{345} \sim \left(\frac{27}{35}, \frac{8}{35}\right)$$

over states $(0,1)$.

(e) (4 points) You would like to compute the probability that more than half of the locations of the solid are in state 1. Here is incomplete pseudocode for using Gibbs sampling to solve this problem. Fill in the empty boxes with Python-like code.
You don't need to worry about burn-in or skipping samples.

1. Initialize values $\bar{v} = (v_1, \ldots, v_{M^3})$ at random from $\{0,1\}$
2.

Solution: count = 0

3. For N iterations:
   - Choose i, j, k randomly from $1, \ldots, M$
   - Set v_{ijk} to a sample from

$$P(V_{ijk} \mid (V \setminus V_{ijk}) = (\bar{v} \setminus v_{ijk}))$$

   -

> **Solution:** if sum($\bar{v}$ == 1) > $M^3/2$: count += 1

4. Return

> **Solution:** count / N

3 Staying dry

3. You are at MIT, where it might be raining, and you need to get to class! You can move from building to building outside and risk getting wet or go between some buildings via tunnels and stay dry.

   You have:

   - A set of buildings $\mathcal{B}$
   - The locations of the buildings, which we will treat as points in x, y space for simplicity, defined by $L(b)$ for $b \in \mathcal{B}$
   - A function $F(b_1, b_2)$ specifying the length of the shortest tunnel between buildings b_1 and b_2, which has value ∞ if there is no tunnel connecting them
   - A *perfectly accurate* weather forecast w: the value of $w(t_1, t_2)$ is the number of seconds, during the time interval $[t_1, t_2]$, it will be raining (the output will be between 0 and $t_2 - t_1$)
   - A fixed walking velocity v

   You goal will be to find a path from a starting building b_0 to a target building b_g that minimizes $d + 3r$ where d is the length of time you travel in which you're *not* getting rained on and r is the length of time you travel in which you *are* getting rained on. If you travel between buildings outdoors, you'll take the shortest Euclidean path.

   (a) (3 points) Assuming that you are not allowed to just wait around (this is MIT!) and have to leave each building as soon as you arrive, until you reach your destination, what is the action space in this problem?

   Solution: Starting from buildng b, an action specifies both the destination building and the route type:

   $$a = (b', m)$$

   where $b' \in \mathcal{B} \setminus \{b\}$ is the destination building and $m \in \{\text{covered}, \text{outside}\}$ is the travel mode. The action $(b', \text{covered})$ is available only if $F(b, b') < \infty$, while $(b', \text{outside})$ is always available.

   (b) (5 points) Assume the state space is defined as $\mathcal{B} \times \mathcal{T}$ where $\mathcal{T}$ is the set of times, modeled as real numbers. So, a state has the form (b, t).
   Given a state (b, t) and the action of walking *outside* to building b', define the next state in terms of b and b'.

   Solution: Let the state be (b, t). Define the outdoor Euclidean distance

   $$D(b, b') = \|L(b) - L(b')\|.$$

   The next state is

   $$(b',\ t + \frac{D(b, b')}{v})$$

   (c) Given a state (b, t) and the action of walking *outside* to building b', define the step cost in terms of b, and b'.

   Solution: Let

   $$D(b, b') = \|L(b) - L(b')\|$$

   and define the arrival time

   $$t' = t + \frac{D(b, b')}{v}.$$

The rainy portion of the trip is

$$r = w(t, t'),$$

and the non-rainy portion is

$$d = (t' - t) - w(t, t').$$

So the step cost is

$$d + 3r = \big((t' - t) - w(t, t')\big) + 3w(t, t') = (t' - t) + 2w(t, t').$$

Equivalently,

$$\text{step-cost} = \frac{D(b, b')}{v} + 2\, w\left(t,\ t + \frac{D(b, b')}{v}\right).$$

(d) (5 points) Bob is worried that this problem can't be well formalized as a min-cost path search problem because we made the state space continuous by including time. Alice disagrees. Alice is right! Explain why.

Solution: At each node in the search tree there are a fixed, finite number of children.
The cost is additive.
An optimal search will expand nodes in order of cost. So we'll eventually hit the min cost node and terminate.

(e) (5 points) Bob is convinced by Alice's argument, but says he thinks it would be better if he were allowed to wait inside buildings for the rain to stop. But he's worried that if we do that, then the problem can't be well formalized as a min-cost path search problem. This time, Bob is right! Explain why.

Solution: The action space is infinite.

(f) (12 points) Which of the following search methods is guaranteed to find a cost-optimal solution in our original problem, without the ability to wait?

If you answer "Not optimal," explain why.

i. uniform-cost search

√ **optimal** ◯ not optimal

> **Solution:** No explanation needed, but here is the explanation: Uniform-cost search always expands the node with the smallest path cost so far. Since all step costs are nonnegative, it is guaranteed to find a cost-optimal solution.

ii. A* search with $h((b, t))$ equal to 3 times the amount of time required to walk a shortest path from b to the goal

◯ optimal √ **not optimal**

> **Solution:** **T**his heuristic is not admissible. It assumes all remaining travel is rainy and costs 3 times the travel time, but some of the remaining path could be through tunnels, which would cost only 1 times the travel time.

iii. A* search, with $h((b, t))$ equal to the cost of the optimal solution to a version of the problem where there are no tunnels

◯ optimal √ **not optimal**

> **Solution:** **T**his heuristic comes from a more restricted problem where tunnels are removed. Removing options can only increase the optimal cost, so this heuristic may overestimate the true remaining cost in the real problem. Therefore it is not guaranteed to be admissible.

iv. A* search, with $h((b, t))$ equal to the cost of the optimal solution to a version of the problem where the shortest path between each building pair goes through a tunnel

√ **optimal** ◯ not optimal

> **Solution:** No explanation needed, but here is the explanation: This is a relaxed version of the problem that can only make the remaining cost smaller, since it adds covered connections between every pair of buildings. So its optimal cost is a lower bound on the true remaining cost, making the heuristic admissible.

4 Shop the job

4. Let's return to our simple job-shop scheduling problem, with some modifications.

   - There are two machines: painter, dryer
   - Each part must be painted and then dried, in that order.
   - The painter can only accommodate one part at a time.
   - The dryer can accommodate two parts at a time.

   We'll use the following predicates:

   ```
   (painted ?part)
   (dried ?part)
   (location ?part ?location)
   ```

   As a reminder of the syntax, here is a PDDL operator description for moving a part from one location to another. You can assume we have locations: `storage-shelf, painter, dryer`.

   ```
   (:action move
       :parameters (?part ?from ?to)
       :precondition (and
           (location ?part ?from))
       :effect (and
           (not (location ?part ?from))
           (location ?part ?to)))
   ```

   For simplicity, in the above operator we are omitting modeling the fact that there must be space in the machine in order to move a part into it.

   (a) (5 points) Write the PDDL operator description for drying two parts simultaneously.

   Solution:
   Drying two parts:

   ```
   (:action dry-two
       :parameters (?p1 ?p2)
       :precondition (and
           (painted ?p1)
           (painted ?p2)
           (location ?p1 dryer)
           (location ?p2 dryer)
           (not (= ?p1 ?p2))
       )
       :effect (and
           (dried ?p1)
           (dried ?p2)
       )
   )
   ```

   OK to include (not (dried ?p1)), (not (dried ?p2)) as pre-condition.

   (b) (3 points) Provide a plan with the minimal number of steps for producing 2 dried parts, assuming the parts all start neither painted, nor dried, on the storage shelf. Available actions are `move, paint, dry-two`.
   For ease of grading, write your answer as a sequence of grounded PDDL actions, for example:

   ```
   (paint part1)
   (move part2 storage-shelf dryer)
   ```

Solution: The minimal plan has **7 steps**.
Reason: each part must

1. move from the storage shelf to the painter,
2. be painted,
3. move to the dryer.

That is 3 steps per part, so 6 steps for two parts. Since the dryer can dry two parts at once, both parts can be dried together in one final step, for a total of $6 + 1 = 7$ steps.
One valid minimal plan is:

1. `(move part1 storage-shelf painter)`
2. `(paint part1)`
3. `(move part1 painter dryer)`
4. `(move part2 storage-shelf painter)`
5. `(paint part2)`
6. `(move part1 painter dryer)`
7. `(dry-two part1 part2)`

For the remaining questions, assume the following PDDL problem description.

```
(define (problem finish-two-parts)
 (:domain job-shop)
 (:objects p1 p2 storage-shelf painter dryer)
 (:init
    (location p1 storage-shelf)
    (location p2 storage-shelf))
 (:goal
    (dried p1) (dried p2)))
```

(c) (4 points) The relaxed plan graph lets us compute the sets of new fluents achievable at level l, F_l, for all the levels. For each level, list the fluents newly achievable at that level. Stop when you have included the goal fluents.

Solution:

```
F0:
(location p1 storage-shelf)
(location p2 storage-shelf)

F1:
(location p1 storage-shelf)
(location p2 storage-shelf)
(location p1 painter)
(location p1 dryer)
(location p2 painter)
(location p2 dryer)
```

```
F2:
(all facts from F1)
(painted p1)
(painted p2)

F3:
(all facts from F2)
(dried p1)
(dried p2)
```

(d) (2 points) What is $h_{\max}$ for `(and (dried p1) (dried p2))` ?

Solution: Each goal literal, `(dried p1)` and `(dried p2)`, first appears at level F_3. Therefore

$$h_{\max} = \max(3, 3) = 3.$$

(e) (2 points) What is h_{add} for `(and (dried p1) (dried p2))` ?

Solution:

$$h_{add}((and\ (dried\ p1)\ (dried\ p2))) = 3 + 3 = 6.$$

(f) (2 points) Which, if any, of these two heuristics is admissible in general? Explain your answer.
√ $h_{\mathbf{max}}$ ◯ h_{add}

Solution:
$h_{\max}$ is admissible because it takes the maximum cost of achieving any single goal, so it never overestimates the true cost of achieving all goals together.
h_{add} is not admissible in general because it adds goal costs independently and can double-count shared actions or subgoals. That can make it larger than the true optimal cost.

5 MCTStar

5. We're going to apply UCT to a minimum-cost-path problem! We will:

   - Add a new state to our problem, called `done`
   - If we reach any state $s \in \mathcal{G}$, we go to state `done`
   - Assume every action taken from the `done` state returns to `done`
   - Assume the cost of each action to 1, so $R(s, a) = -1$ for all actions in all states, except $R(\texttt{done}, a) = 0$
   - Define `UCT_path(s0, (A, T, R, H), M)`, to calls `UCT(s0, (A, T, R, H), M)` and return the path in the tree constructed so far that has the largest summed reward

   The pseudocode for UCT from the notes is on the next page for reference.

   (a) Generally, in a minimum-cost path problem, we are not sure how many steps the optimal path will have. We have come up with several strategies for handling this. For each, explain whether it will *eventually* (assuming we never terminate the outer loop) find an optimal path to a goal state and if not why not.

   i. (3 points) Make a call to `UCT_path` with $H = \infty$, and $M = \infty$, letting all of the random roll-outs run until they hit a state in $\mathcal{G}$.

   ◯ Guaranteed optimal √ **Not guaranteed optimal**

   Solution:
   A random rollout may never reach a goal if there are "dead end" states from which a goal state cannot be reached, and therefore run forever

   ii. (3 points) Pick a number M of samples, and then do a nested loop increasing H and starting the search fresh each time, and remembering the best solution found for each H. The question is: will `sols` eventually contain an optimal solution?

```
H = 1 ; sols = {}
while True:
    sols[H] = UCT_path(s0, (A, T, R, H), M)
    H = H + 1
```

   ◯ Guaranteed optimal √ **Not guaranteed optimal**

   Solution:
   Although the correct horizon will eventually be reached, only M samples are taken before statistics are erased. The optimal path may not be discovered in those samples.

   iii. (3 points) Pick a number M of samples, and then do a nested loop, increasing H each time, but retaining the tree including all of the N and U values from the previous iteration

```
H = 1
sol = UCT_path(s0, (A, T, R, H), M)
while True:
    if sol reaches G:
        return sol
    H = H + 1
    Retaining the existing UCT tree, run M more
    iterations with the increased H
```

   ◯ Guaranteed optimal √ **Not guaranteed optimal**

Solution:
Paths from short H will have small costs and paths with later H will have large costs and the statistics really shouldn't be mixed together.

(b) (3 points) We happen to have an admissible heuristic h. Your friend suggests that, instead of random roll-outs, we should behave greedily with respect to h to roll-out a leaf node–that is, in line 3 of SIMULATE, choose the action that produces an s' that minimizes $h(s')$. Is that a good or bad idea? Explain.

Solution: Bad idea.
Even though h is admissible, it may give poor local guidance. If rollouts always act greedily with respect to h, they may repeatedly follow the same suboptimal path and fail to explore alternatives. Random rollouts help maintain exploration and reduce this bias.

A UCT pseudocode

```
MCTS(s_0, (A, T, R, H), iters)
root = NODE(s_0, horizon = H, parent = None, children = { }, U = 0, N = 0)
for iter ∈ {1, ..., iters}:
    leaf = SELECT(root)
    child = EXPAND(leaf, A, T)
    value = SIMULATE(child, A, T, R)
    BACKUP(child, value)
max_child = max(root.children, key = λn. n.U/n.N)
return root.children[max_child]            // Returns the associated action

SELECT(n)
    // Follow optimistically best path through tree
if n.children
    return SELECT(max(n.children, key = λc.UCB(n.N, c.N, c.U)))
else
    return n

EXPAND(n, A, T)
    // Unless remaining horizon is 0, add child nodes and return one
if n.horizon = 0:
    return n
else
    for a ∈ A:
        s' = T(n.s, a)
        n' = NODE(s', n.horizon − 1, parent = n, children = { }, U = 0, N = 0)
        n.children[n'] = a
    return RANDOM_CHOICE(n.children)

SIMULATE(n, A, T, R)
    // Randomly finish path and return cumulative reward
s = n.s; total_reward = 0
for h ∈ (n.horizon, ..., 1):
    a = RANDOM_CHOICE(A)
    s' = T(s, a)
    total_reward += R(s, a, s')
    s = s'
return total_reward

BACKUP(n, v_below)
    // Add value v to n's statistics and pass it up
n.N += 1
if n.parent:
    a = n.parent.children[n]                    // Action that led to n
    v = v_below + R(n.parent.s, a, n.s)  // Value of executing a in parent
    n.U += v
    BACKUP(n.parent, v)
```