Solutions

Probabilistic Models

[↑]

Problem 1

To find the probability of both A and B happening, or $P(A, B)$, you first need the probability of one of them happening. Given that one has happened, you then need to multiply by the probability of the other happening. For example, we can first look at the probability of B happening, $P(B)$. The probability of A given that B has happened is $P(A|B)$. The combined probability $P(A|B)P(B)$ is equal to the probability of both A and B. Note that you could do this starting with A rather than B. That is,

$$P(A, B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

Similar reasoning follows if we wish to manipulate $P(A, B \mid C)$. We first need the probability of either A or B happening given C. Let us choose B. This is $P(B \mid C)$. Given that B and C have now happened, we multiply by the probability of A given both B and C, $P(A \mid B, C)$. This yields a final solution of

$$P(A, B \mid C) = P(A \mid B, C)P(B \mid C)$$

Problem 2

Using Bayes' rule, the probability that a woman has breast cancer (C) given that she tested positive ($+$) is

$$P(C \mid +) = \frac{P(+ \mid C)P(C)}{P(+)}$$

Using the information given in the problem, $P(+ \mid C) = 0.8$, $P(+ \mid \neg C) = 0.096$, and $P(C) = 0.01$. The denominator can be found using the law of total probability:

$$P(+) = P(C)P(+ \mid C) + P(\neg C)P(+ \mid \neg C)$$

where $\neg C$ means the woman does not have cancer. The final expression is

$$P(C \mid +) = \frac{P(+ \mid C)P(C)}{P(C)P(+ \mid C) + P(\neg C)P(+ \mid \neg C)} = \frac{(0.8)(0.01)}{(0.01)(0.8) + (0.99)(0.096)} = 0.078$$

There is a 7.8% chance that a woman with a positive test has cancer. Only 15% of doctors are able to answer this question correctly (Casscells, Schoenberger, and Grayboys 1978; Eddy 1982; Gigerenzer and Hoffrage 1995). The most common mistake is to ignore the original fraction of women with breast cancer, and the fraction of women without breast cancer who receive false positives, and focus only on the fraction of women with breast cancer who get positive results.

Problem 3

We can represent the joint distribution in the table below.

	Water	No Water
Life	0.5	0.25
No Life	0	0.25

(images/mars_table.png)

Three of the four entries are given in the problem statement. The fourth entry, P(No Life, Water), is 0 because the entries in the table must sum to 1. If we use the notation l^1 for 'life', l^0 for 'no life', and w^1 for 'water', and w^0 for 'no water', we can say that our table represents $P(L, W)$ and we wish to find $P(l^1 \mid w^1)$.

We know from the law of conditional probability that

$$P(l^1 \mid w^1) = \frac{P(l^1, w^1)}{P(w^1)}$$

.

The numerator can be read directly from the table ($P(l^1, w^1) = 0.5$). The denominator can be found using the law of total probability:

$$P(w^1) = \sum_l P(l, w^1)$$

.

Summing over the first column of the table, we see that $P(w^1) = 0.5 + 0 = 0.5$. The probability of life on Mars given that there is water is therefore 0.5 / 0.5 = 1.

Peoblem 4

Six parameters are needed if X is binary and has two parents, one that can take on three values and another that can take on two. This is one parameter for every combination of the parents of X. Only one is needed because X only takes on two values, and by knowing the probability of one you know the probability of the other.

If another parent with four possible values were added to X, 24 parameters would be needed.

If X can take on three values, but still has the three parents from above, then 48 parameters would be needed. This is two for every possible combination of the parents.

The number of parameters needed to represent the probability of a discrete variable X with n discrete parents is

$$(|X| - 1) \prod_{i=1}^{n} | \left(\mathrm{Pa}_X\right)_i |$$

where $|X|$ is the number of possible values of the variable X.

Problem 5

1. True
2. False
3. False
4. False

Problem 6

The linear Gaussian distribution has the following form:

$$P(k \mid l) = N\left(k \mid \theta_1 l + \theta_2, \theta_3\right)$$

where θ_3 is the variance of the distribution. The standard deviation is 200 kg so the variance $\theta_3 = 40000$.

Using the information given in the problem, we can see that the mean krill consumption k as a function of whale length l is $k = 20l + 1600$. Thus $\theta_1 = 20$ and $\theta_2 = 1600$. The distribution is therefore

$$P(k \mid l) = N\left(k \mid 20l + 1600, 40000\right)$$

Problem 7

First, apply Bayes' rule to $P(s_t \mid o_{0:t})$:

$$P(s_t \mid o_{0:t}) \propto P(o_{0:t} \mid s_t) P(s_t)$$

Here we realize that the probability of $o_{0:t}$ is the joint probability of o_t and $o_{0:t-1}$.

$$P(s_t \mid o_{0:t}) \propto P(o_t, o_{0:t-1} \mid s_t) P(s_t)$$

Apply Bayes' rule to the first term:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid o_{0:t-1}, s_t) P(o_{0:t-1} \mid s_t) P(s_t)$$

Then, apply Bayes' rule to the last two terms:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid o_{0:t-1}, s_t) P(s_t \mid o_{0:t-1})$$

Given knowledge of the current state s_t, the current observation o_t is independent of previous observations $o_{0:t-1}$, and we have the first expression:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) P(s_t \mid o_{0:t-1})$$

Now, use the law of total probability to add a new variable:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t, s_{t-1} \mid o_{0:t-1})$$

Apply the law of conditional probability to the term inside the summation:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}, o_{0:t-1}) P(s_{t-1} \mid o_{0:t-1})$$

Given the state at time $t-1$, the state at time t is independent of the previous observations $o_{0:t-1}$:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}) P(s_{t-1} \mid o_{0:t-1})$$

Problem 8

For some node o_t, the Markov blanket is s_t because s_t d-separates o_t from all other nodes.

Problem 9

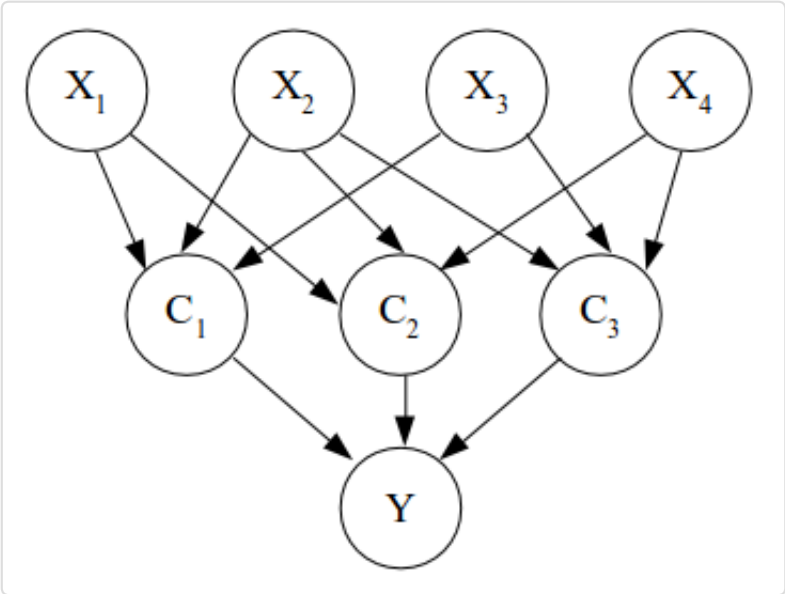
$$P(A \mid C) = \sum_{B \in B_{set}} P(A \mid B, C) P(B \mid C)$$

Problem 10

A topological sort is a list of the nodes of a Bayesian network sorted so that the parents of some node A always appear before A. You want to do a topological sort before sampling from a Bayesian network so that you sample from the parents of a node before sampling from the node itself. A topological sort always exists, but it may not be unique.

Problem 11

A Bayesian network representation of the given 3SAT problem is shown below. Each of the X_i nodes represents a variable, and each of the C_i nodes represents a clause. If a clause is true, it will assign a probability of 1 to true. The last node will be true with probability 1 given that its parents are true.



(images/bayes_3sat.png)

The 3SAT problem is NP-complete and we have shown that inference in Bayesian networks is at least this hard. This means that inference in Bayesian networks is NP-hard. Intuitive explanations of the complexity classes NP-complete and NP-hard can be found here (http://stackoverflow.com/questions/1857244/np-vs-np-complete-vs-np-hard-what-does-it-all-mean).

Problem 12

The goal of inference is to find a distribution over some unobserved variables given a set of observed variables. Inference might be used when the structure and parameters of the Bayesian network are known. This includes classification tasks, where you want to infer a class given a set of observations.

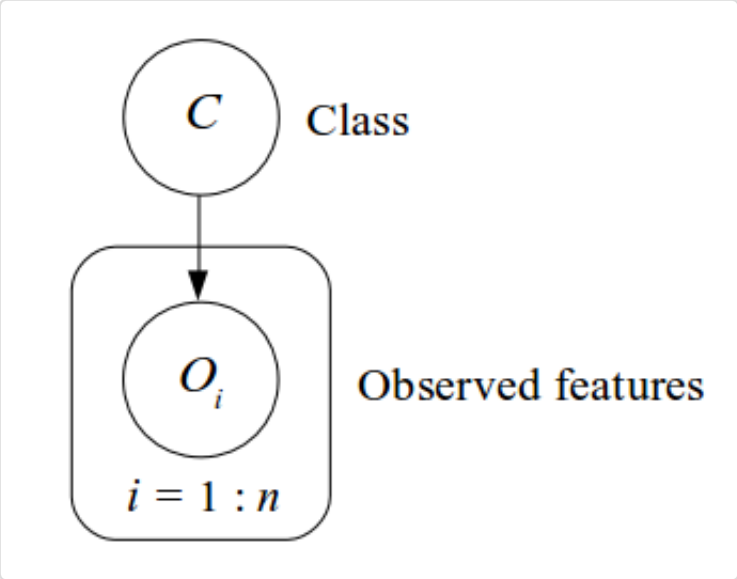
The goal of parameter learning is find the parameters of the distributions that determine the conditional probabilities between variables. Parameter learning might be used if the structure of the Bayesian network is known, but the conditional probabilities are unknown.

The goal of structure learning is to find the (unknown) structure of the Bayesian network. This could arise if you had a large dataset and wanted to find the relationships between variables.

Problem 13

In a classification task, you must predict a class given some observations. Radar target classification is an example given in the textbook. Given some radar observations like speed and heading fluctuation, the goal is to correctly identify an object as a bird or a plane.

A naive Bayes model assumes that the observed features (evidence variables) are independent given the class. The compact representation shown below is called a plate representation.



(images/plate_representation.png)

Problem 14.1

Maximum likelihood estimation runs into problems when data is limited. The example in the textbook refers to estimating the probability of mid-air collisions. Because such events are rare, there might not be a single collision in your dataset. Maximum likelihood estimation would suggest the probability of mid-air collisions to be zero, which is not correct.

Problem 14.2

Some of the advantages of Bayesian parameter learning:
1. We can incorporate a prior on the parameters, which can leverage expert knowledge (especially useful when data is sparse)
2. By computing a posterior distribution, we not only obtain an estimate for the parameters, but also a confidence in our estimate.

As for the drawbacks:
1. We need to come up with a prior distribution. Depending on the prior, the posterior will be affected. This might lead to the use of non-informative priors in the hope of letting the data drive the posterior through the likelihood function, which in turn can lead to results that are similar to using maximum likelihood.
2. Computing the posterior can only be done in some cases, but in general, we have to use numerical integration or Monte Carlo methods.

Problem 15

The gamma function is a real-valued generalization of the factorial.

$$\Gamma(n) = (n-1)!$$

$$\Gamma(5) = 24$$

Problem 16

If you knew nothing about the two teams, a uniform prior would be appropriate. This might look like Beta(1,1).

If you were sure that the two teams were evenly matched, Beta(9,9) would be more appropriate than Beta(2,2) because this gives more weight to values near $\theta = 0.5$.

If you are confident that Team A is more likely to win, you might use Beta(5,1) for your prior. The pseudocount for Team A winning should be higher if they are more likely to win.

If the two teams are going to play many games against each other, the prior you select is less important. The real counts will become more important than the pseudocounts in determining the distribution.

Problem 17
Suppose you have a lot of data for flipping a coin and that θ represents the probability of the coin landing on heads. You are fairly certain that the probability of heads is 50%. This would be represented by more weight at $\theta = 0.5$.

Say you only have two data points: one heads and one tails. If you did not know anything about how coins work, you could not be certain that the probability of heads is 50%. There is not enough data to justify that conclusion.

Pseudocounts are very similar to observed counts. In fact, updating your distribution involves adding observed counts to the pseudocounts. So, if you have high but even pseudocounts, you are essentially saying you are certain that the probability of heads is 50%. This is why there will be more weight at $\theta = 0.5$.

Problem 18
In general, a sparse network is better if there is not much data. This is because data is needed to justify connections between nodes, and it is likely that there are not enough samples to justify this if there is not much data. However, if there is a lot of data, a highly connected graph will likely do better.

Problem 19
Zero. Two conditions must be satisfied for a directed acyclic graph to be in the same Markov equivalence class: the graph must have the same edges (regardless of direction), and it must have the same v-structures. If you try to create a directed acyclic graph from the partially directed graph given, you will create a new v-structure (regardless of which direction you choose). Thus, any directed acyclic graphs created from the given partially directed graph will not be members of the same Markov equivalence class.

Problem 20
Subsequent samples are correlated. This can be handled by using only every k samples. Gibbs sampling is also only guaranteed to converge to the true distribution in the limit, thus the first n samples must be discarded (e.g. the burn-in period)

Problem 21
A, F, B, D, C, E

Decision Problems

[↑]

Problem 1

An agent is rational if it adheres to the *principle of maximum expected utility*. This is an important principle in economics, game theory, and artificial intelligence. It states that a rational agent will always choose the action that maximizes its expected utility. This action is

$$a^* = \text{argmax}_a EU(a \mid o)$$

Problem 2

The value of information is the increase in expected value gained by using an observation to make a better decision.

If an observation does not change the optimal action, then the value of that observation is zero.

If the optimal action changes after an observation, then the value of that information is positive. Making that observation enabled the agent to choose an action with higher expected utility than its previously chosen action.

Problem 3

There is no longer a dominant strategy equilibrium. There are, however, two Nash equilibria at (Testify, Refuse) and (Refuse, Testify) shown below as (-4,0) and (0,-4). Given that the two agents are at either of these equilibrium points, it does not make sense to change action. Imagine that you are agent 1 and the equilibrium is currently the lower left corner (-4,0). If you decide to change your action and testify, you will receive a score of -5, so it does not make sense to change. This differs from the original prisoners dilemma, in which the lower left corner was (-10,0). In that case it makes sense to change your action and testify because doing so results in a better score of -5.

-5, -5	0,-4
-4,0	-1,-1

(images/modified_prisoner_nash.png)

Problem 4

Please refer to the explanation and the payoff matrix here (http://en.wikipedia.org/wiki/Traveler's_dilemma).

Sequential Problems

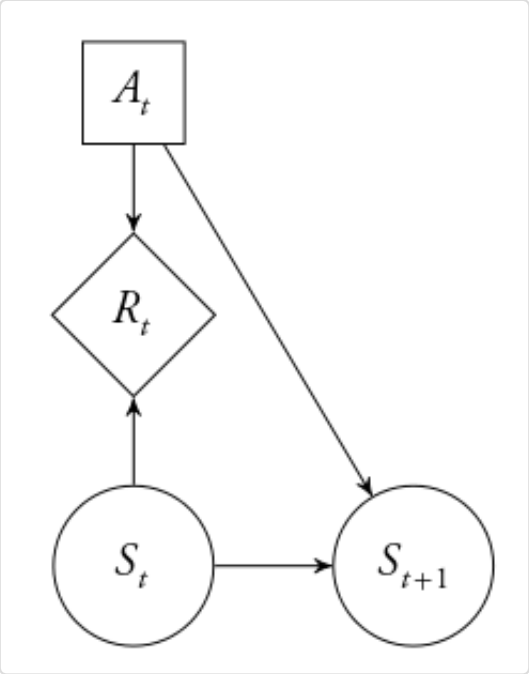
[↑]

Problem 1

The Markov assumption is that your current state only depends on your previous state and the action you took to get there.

A Markov decision process consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $T(s' \mid s, a)$, and a reward function $R(s, a)$.

A stationary MDP, shown below, is one where the transition and reward functions do not change with time. We will be dealing mostly with this type of MDP in this class.



(images/mdp_structure.png)

Problem 2

The purpose of the discount factor is to give a higher value to plans that reach a reward sooner. It also bounds the utility of a state, ensuring it does not reach infinity. We typically denote the discount factor with γ, and it can hold values between 0 and 1.

A small discount factor discounts future rewards more heavily. A solution with a small discount factor will be *greedy*, meaning that it will prefer immediate rewards more than long-term rewards. The opposite is true for a large discount factor.

An alternative to using a discount factor is to use the average reward:

$$\lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n-1} r_t$$

Problem 3

The optimal policy isn't necessarily unique, but the optimal value for each state is unique.

Problem 4

The Bellman equation is

$$U^*(s) = \max_a \left(R(s,a) + \gamma \sum T(s' \mid s,a) U^*(s') \right)$$

If the transition function is deterministic, we do not need to sum over all possible s'. There is only one possible s' and the probability of transitioning to it is 1. Thus, we drop the summation and transition function and get

$$U^*(s) = \max_a (R(s,a) + \gamma U^*(s'))$$

Problem 5

$\mathbf{T}^\pi$ is the transition function for only the policy π. The transition function changes from $P(s' \mid s, a)$ to $P(s' \mid s, \pi(s))$. For a given policy, the next state s' is now only a function of the current state s. This effectively converts the Markov decision process into a Markov chain.

Problem 6

Dynamic programming is a method for solving problems by breaking them into subproblems. There are many examples of dynamic programming such as longest common subsequence (https://en.wikipedia.org/wiki/Longest_common_subsequence_problem) and the Bellman-Ford algorithm (https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm). Dynamic programming is more efficient than brute force methods because it leverages the solutions of subproblems.

Problem 7

Policy iteration and value iteration are algorithms used to find optimal policies. Both are forms of dynamic programming. Policy iteration directly updates the policy. Value iteration updates the expected utility of each state using the Bellman equation and retrieves the optimal policy from those utilities.

Problem 8

Open-loop planning methods do not account for future state information. In control systems engineering this future state information is called *feedback*. Closed-loop methods plan by anticipating feedback from the environment and and considering the ability to take actions in future states.

Problem 9

We can represent the utilities with a vector: $[s_1, s_2, s_3, s_4]$. We can initialize the utilities to $[0, 0, 0, 0]$.

After the first iteration, the utilities are $[100, 0, 0, 0]$.

After the second iteration, the utilities are $[100, 90, 0, 0]$.

After the third iteration, the utilities are $[100, 90, 81, 0]$.

After the fourth iteration, the utilities are $[100, 90, 81, 72.9]$.

Problem 10

In asynchronous value iteration, only a subset of the states may be updated per iteration. The state ordering is important because different orderings can take a different number of iterations to converge to the optimal value function.

When using the first ordering s_1, s_2, s_3, s_4, Gauss-Seidel value iteration converges after a single iteration. We first initialize the utilities to 0 for all states. When we consider s_1, we see that it has an immediate reward of 100 if it goes left. Going right yields no immediate reward, and the state to the right has utility 0 (for now). Thus, we update the value of s_1 to be 100. We then consider the next state in our list, which is s_2. No direction yields an immediate reward, but we know there is a utility of 100 of being in the state to the left. Applying our discount factor, we determine that the utility of being in s_2 is 90. We folow the same procedure to get 81 and 72.9 for the utilities of s_3 and s_4, respectively. Note that this took only one iteration. As we swept through our ordering of states, going left to right, the reward from the left-most state propagated to the right.

If we use the reverse order, it takes four iterations, the same as normal value iteration. We first consider s_4, and see that it gets no immediate reward, and the only other state it can go to has a utility of 0, so it has an updated utility of 0. We proceed this way down the states until we reach s_1, which will be updated to a value of 100. Because we are working against the direction of reward propagation, the process takes four iterations to converge.

Problem 11
Dynamic programming can be used for problems with small state and action spaces and has the advantage of being guaranteed to converge to the optimal policy. Approximate dynamic programming is often used for problems with large or continuous state and action spaces. In these problems, dynamic programming may be intractable so an approximation to the optimal policy is often good enough. Online methods are used for problems with very large or continous state and action spaces where finding a good approximation to the optimal policy over the entire state space is intractable. In an online method, the optimal policy is approximated for only the current state. This approximation greatly reduces the computational complexity but also requires computation every time a new state is reached.

Model Uncertainty

[↑]

Problem 1
Reinforcement learning is typically used in problems in which there is model uncertainty, that is, an unknown transition and reward model. Solution methods include model-based and model-free approaches.

Problem 2
Exploration versus exploitation describes the decision between exploiting information you know and exploring to find new information. If you exploit the knowledge you have, but never explore any new options, you might be missing out on even better rewards. If you spend all your time exploring, and never exploit what you know, you might not get any reward.

A multi-armed bandit is a slot machine with multiple levers. There are n levers, and each has some probability θ_i of returning a payoff of \$1 and probability $1 - \theta_i$ of returning a payoff of \$0.

For the situation described, you can continue exploiting the payout from the lever you know about, or you could explore the untested lever. It is possible that the untested lever pays out with a probability of 0.95, and is better than the other lever. You cannot know this without exploring. However, it is possible that the unexplored lever

only returns a payout with a much lower probability. In that case, you would have been better off exploiting the lever you knew about.

Problem 3

ρ_i is our estimate of θ_i, which is the payout rate for lever i.

If you were using an ϵ-greedy strategy with $\epsilon = 0.5$, 50% of the time you would pick a random lever. The other 50% of the time you would pick the lever with the highest estimated payout rate. In the example presented, you might flip a coin. If it came up heads, you would close your eyes and pick a lever. If it came up tails, you would pull the first lever, with the higher estimated payout rate of 0.7.

If you were using an interval exploration strategy with 95% confidence intervals, you would pick the second lever. It has a lower estimated payout rate of 0.7, but the upper bound on its confidence interval is higher. This means we suspect it can have a higher payout rate than the first lever. Therefore, we pursue this option until the upper bound on this confidence interval becomes lower than the other lever (if that ever happens).

Problem 4

$Q(s, a)$ represents the utility of taking action a in state s. It assumes after taking this action, you act optimally afterwards. Q represents the utility of a specific state-action combination. On the other hand, $U(s)$ is the utility of being in a specific state regardless of action. Each U is the best Q-value for a given state, considering all possible actions.

If you had a model and wanted to run value iteration with Q-values instead of utilities, you would use the following equation:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) \max_{a'} Q(s', a')$$

Problem 5

The central equation in incremental estimation is

$$\hat{x} = \hat{x} + \alpha(x - \hat{x})$$

The learning rate is α and the temporal difference error is $x - \hat{x}$. The temporal difference error is the difference between a sample and our previous estimate.

In the example given, our estimate is 3. We observe a new value of 7. If we have an α of 0.1 we will get a new estimate of $3 + 0.1(7 - 3) = 3.4$. If our α is 0.5, our new estimate will be 5. A larger learning rate means new samples have a greater effect on the current estimate.

Problem 6

Q-learning and Sarsa both work on the principle of incremental estimation. In Q-learning, we update the Q-value by maximizing over actions in the next state s_{t+1}. Sarsa differs from Q-learning by using the Q-value corresponding to the action we actually take at s_{t+1}. This has the advantage of not needing to iterate over all possible actions.

Problem 7

We start with modifying the Bellman equation for Q-values:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) \max_{a'} Q(s',a')$$

The incremental update equation is

$$\hat{x} = \hat{x} + \alpha(x - \hat{x})$$

We wish to apply this update equation to Q; we want to incrementally update our estimate of Q:

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha \left(Q(s,a) - \hat{Q}(s,a) \right)$$

The value of a new sample $Q(s,a)$ is based on the reward r and next observed state s'.

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - \hat{Q}(s,a) \right)$$

We usually just drop the hats and add time subscripts, leaving the update equation for Q-learning:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha \left(r_t + \gamma \max_{a} Q(s_{t+1},a) - Q(s_t,a_t) \right)$$

In Sarsa, we do not actually maximize over all possible actions, we just use the Q-value associated with the action we actually took at $t+1$. This is the reason for the name Sarsa $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$.

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha \left(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right)$$

Problem 8

Sarsa(λ) is a modified version of Sarsa that assigns credit to states and actions that were encountered on the way to a reward state. This assignment of reward to intermediate states and actions is referred to as using eligibility traces. Eligibility traces can speed up learning for problems that have sparse rewards such games that result in a win or loss at the end.

Problem 9

Model-based approaches can usually find a better policy than model-free approaches at the cost of being more computationally expensive.

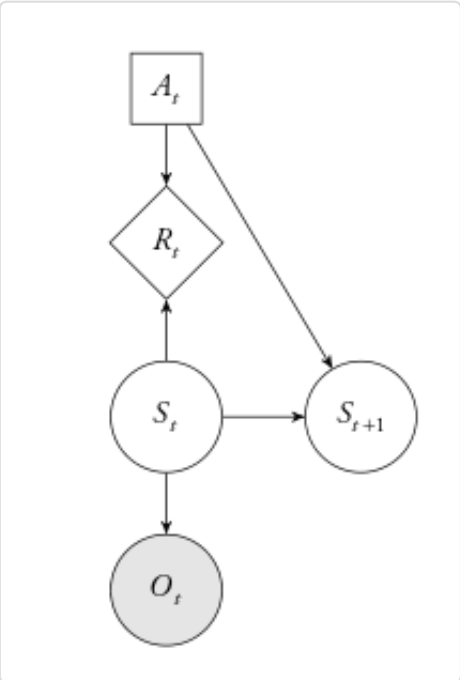
State Uncertainty

[↑]

Problem 1

A POMDP is a partially observable Markov decision process. It is an extension of the Markov decision processes that allows you to model the fact that you may not be able to observe your state directly. Instead, you make observations that give you some knowledge about your state.

The structure of a POMDP is shown below. Note that it has an extra node corresponding to observations. The observations you make depend on the state you are in. Your observations can also depend on your action, which is not shown in this image.



(images/pomdp_structure.png)

Problem 2

The state of the left gridworld could be represented by a single number. We could say $s = 3$.

In the right gridworld, we would represent our "state" as a belief, or distribution over these states. This distribution might look like a vector, where the ith element is the probability you are in the ith state. For our example, it might look something like:

$$b(s) = \begin{pmatrix} 0 \\ 0.25 \\ 0.5 \\ 0 \\ 0 \\ 0.25 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Our "state" in the right gridworld is a probability distribution over the states. In POMDPs, we call this a belief. This is why POMDPs are sometimes called "belief-state MDPs".

This example shows why POMDPs are difficult to solve. There are a total of 9 possible states in the MDP, but there are infinte possible beliefs in the POMDP. Value iteration requires looping over all states. If we consider POMDPs to be "belief-state" MDPs, then value iteration would need to loop over the infinite possible beliefs, which is impossible in finite time. We could discretize the belief space, but even if we discretized each probability to the nearest tenth, we would end up with 10^9 possible states for our problem, which is much larger than the state space of the MDP.

Problem 3

Three belief updating methods are the discrete state filter, the linear-Gaussian filter, and the particle filter.

The discrete state filter is appropriate for small, discrete state spaces. The linear-Gaussian filter can handle continuous dynamics and observation models as long as the system dynamics are linear-Gaussian. In this case, we can perform exact belief updates.

If discrete filters or linear-Gaussian filters aren't feasible, we typically use an approximate sampling-based method like a particle filter. Particle filters are preferred when the state space is large or continuous and not well approximated by a linear-Gaussian model.

Problem 4

Starting from the definition of a belief update

$$b'(s') = P(s' \mid o, a, b)$$

we can use Bayes's Law to swap o and s':

$$b'(s') \propto P(o \mid s', a, b)P(s' \mid a, b)$$

Note that the observation depends only on the new state, s', and the action taken to reach it, a. This yields the observation function, O.

$$b'(s') \propto O(o \mid s', a)P(s' \mid a, b)$$

If we apply the law of total probability to the right-most term, we can add a new query variable, as long as we sum it out:

$$b'(s') \propto O(o \mid s', a) \sum_s P(s', s \mid a, b)$$

Applying Bayes's Law to the right-most term, we get

$$b'(s') \propto O(o \mid s', a) \sum_s P(s' \mid s, a, b)P(s \mid a, b)$$

Note that s' only depends on the state, s, and action, a. This yields the transition function, T:

$$b'(s') \propto O(o \mid s', a) \sum_s T(s' \mid s, a)P(s \mid a, b)$$

Noting that the previous state is independent of the current action and that the probability of the previous state is encoded in the belief $b(s)$, we get

$$b'(s') \propto O(o \mid s', a) \sum_s T(s' \mid s, a) b(s)$$

Problem 5

If you use a particle filter with rejection, you need a generative model that can sample an s' and o' given an s and a. You draw from this generative model until the o' matches the actual observation you made, o. An issue with this type of particle filter is that you might have to draw many times from the generative model until the generated o' matches the actual observation.

An alternative is to use a particle filter without rejection. This type of particle filter requires a generative model that can sample an s' given an s and a, as well as an observation function $O(o \mid s, a)$. The observation function is used to update the belief by weighting the sampled s''s using the observation function, the action taken, and the actual observation.

More particles is better because the filter will more closely approximate the true belief. Even with a large number of particles, you can still encounter *particle deprivation*, which means there are no particles near the true state. You can help prevent this by adding noise to the particles.

Problem 6

The answer to this problem is provided in the textbook.

Problem 7

POMDP policies can be represented as a set of alpha vectors $\alpha_1, \ldots, \alpha_n$. Each alpha vector corresponds to an action and has as many elements as there are states. The utility of a belief state is the maximum value of $\alpha_i \cdot b(s)$. The best action is the one that corresponds to this maximum dot product.

Problem 8

Let us denote the probability of the exam taking place as P_E. The probability that the exam doesn't happen is therefore $1 - P_E$. Our belief of the world can be represented as the belief vector

$$b = \begin{pmatrix} P_E \\ 1 - P_E \end{pmatrix}$$

The alpha vectors are

$$\alpha_{\text{study}} = \begin{pmatrix} 100 \\ 0 \end{pmatrix}$$

$$\alpha_{\text{relax}} = \begin{pmatrix} -100 \\ 100 \end{pmatrix}$$

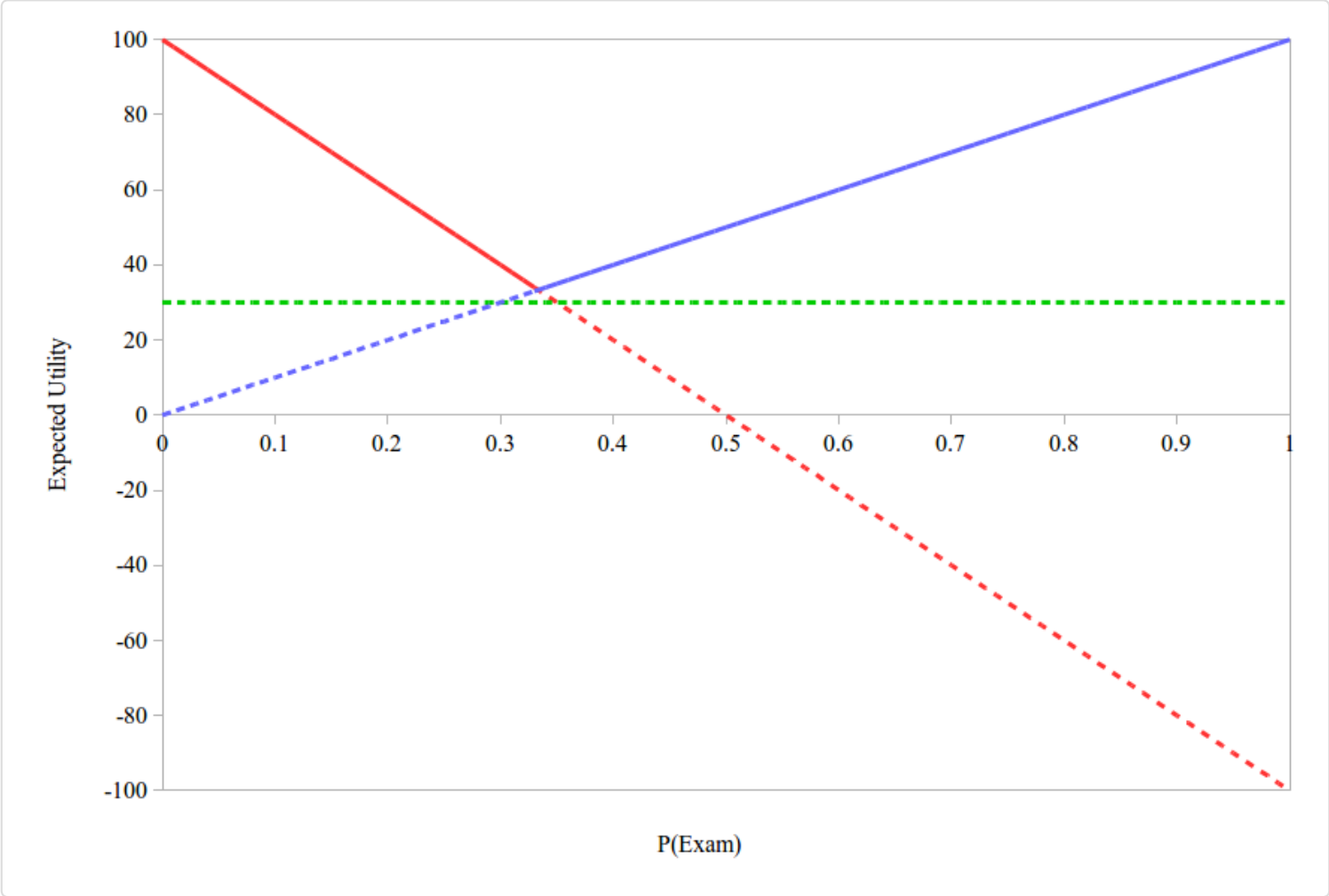
where the first element corresponds to the state where the exam does take place and the second element corresponds to the state where the exam doesn't happen.

The best action is the action corresponding to the greater of the two dot products $(\alpha_{\text{study}} \cdot b(s), \alpha_{\text{relax}} \cdot b(s))$. By setting these dot products equal and solving for P_E,

$$100P_E = -100P_E + 100(1 - P_E)$$

we find that $P_E = 0.33$. It makes sense to relax if we think the chance of an exam is less than 33 percent. These alpha vectors can also be represented graphically, as shown below. The red line is the utility of relaxing, the blue line is the utility of studying, and the horizontal axis the probability that the exam is given. The lines are solid when that alpha vector corresponds to the action with the highest expected utility. The study and relax lines lines intersect at $P_E = 0.33$.

The green line represents the utility of dropping out and living in the wilderness. The value of this action is 30 regardless of P_E. Note that we can always choose an action that results in higher utility if we know the value of P_E. This action is therefore *dominated*, and there is generally no point in storing this alpha vector since it doesn't give us information that improves our policy.



(images/alpha_vecs.png)

Problem 9

The policy is valid. You can have multiple alpha vectors per action.

The first step is to represent the belief as a vector:

$$b = \begin{pmatrix} 0 \\ 0.7 \\ 0.3 \end{pmatrix}$$

To find the optimal action given our policy, simply take the dot product of every alpha vector with the belief state.

$$\alpha_1 \cdot b = 70, \alpha_2 \cdot b = 37, \alpha_3 \cdot b = 50$$

We see that the first alpha vector yielded the highest value, and we know this corresponds to action 1. Therefore, we take action 1.

Problem 10

Solving a POMDP offline means doing the computation before execution. This is nice because we then have a policy we can use at execution time. All we need to do is loop through the alpha vectors, dotting them with the current belief vector in order to determine which action to take. This means we do not need a lot of computational power at execution time.

Solving a POMDP online means doing the computation as you execute. Thisy requires more computational power as you execute, which is a drawback. However, you can typically handle larger problems with online solvers. This is because you can use information about your current state or belief to limit the belief space you search over. If you solve the problem offline, your solution needs to be good over the total range of beliefs you might see. If you solve online, your solution only needs to be as good over the range of beliefs you can reach from the current belief.

The three main approaches we discussed in class for solving POMDPs offline are QMDP, fast informed bound (FIB), and point-based value iteration. Computationally, QMDP is the easiest. It makes the assumption that all states will be fully observable after the next step. This makes the problem relatively easy to solve. However, it does not work well in information-gathering problems. If you make the assumption that everything will be observable after the next step, you will never bother taking any actions that provide information.

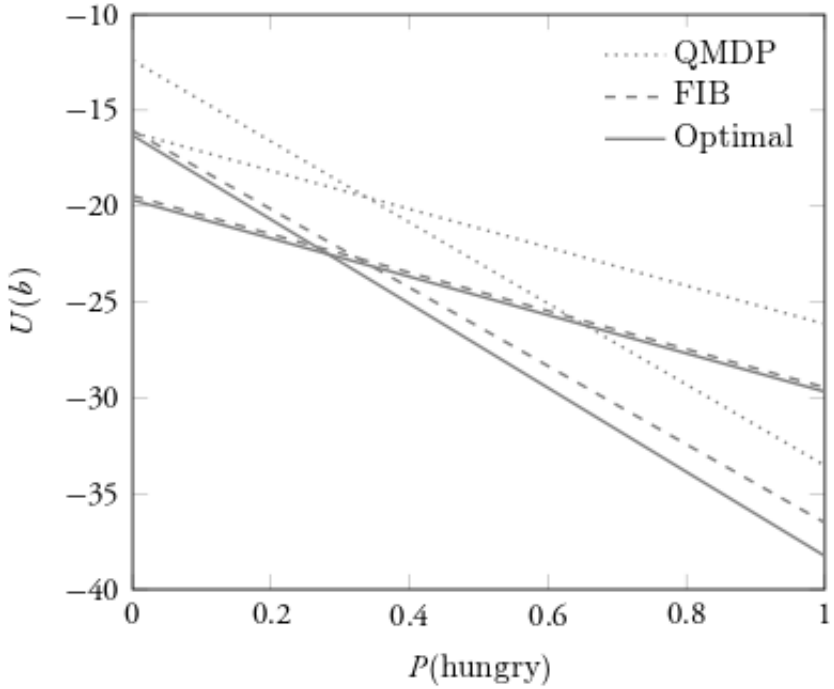
FIB does not fully ignore partial observability, so it is better than QMDP in that regard. However, it is also more computationally complex. Point-based value iteration discretizes the belief space into a set of belief points and performs value iteration over that discretization.

Problem 11

For QMDP, each iteration requires $O(|A|^2|S|^2)$ operations. For each iteration, we must iterate over all of the alpha vectors. Because there is an alpha vector for each action, this results in $|A|$ iterations. For each alpha vector, we must loop through each states, yielding another factor of $|S|$. At each of those states, we run the update equation. This forces us to iterate over all possible s', which yields another factor of $|S|$. In reality, you only have to loop over the reachable states, which is often smaller than the total number of states. You must then also find the maximum across the alpha vectors, forcing you to look through all actions, yielding another factor of $|A|$. We have two factors of $|A|$ and $|S|$ each, yielding a total computational complexity per iteration of $O(|A|^2|S|^2)$.

For FIB, each iteration requires $O(|A|^2|S|^2|O|)$ operations. Showing this follows the same procedure we used for QMDP. However, note that we must also loop over all possible observations, yielding an additional factor of $|O|$.

If you write up some code to solve QMDP and FIB for the crying baby problem, you will get the alpha vectors shown below.



(images/baby_solution.png)