CS221 Exam Solutions

CS221
November 29, 2016

Name: _____
by writing my name I agree to abide by the honor code

SUNet ID: _____

Read all of the following information before starting the exam:

- This test has 4 problems and is worth 150 points total. It is your responsibility to make sure that you have all of the pages.
- Keep your answers precise and concise. Show all work, clearly and in order, or else points will be deducted, even if your final answer is correct.
- Don't spend too much time on one problem. Read through all the problems carefully and do the easy ones first. Try to understand the problems intuitively; it really helps to draw a picture.
- You cannot use any external aids except one double-sided $8\frac{1}{2}$" x 11" page of notes.
- Good luck!

Problem	Part	Max Score	Score
1	-	10	
2	a	10	
	b	10	
	c	10	
	d	10	
3	a	10	
	b	10	
	c	10	
	d	10	
	e	10	
4	a	10	
	b	10	
	c	10	
	d	10	
	e	10	

Total Score: ☐ + ☐ + ☐ = ☐

1. Warmup (*10 points*)

Let us warmup with five conceptual multiple choice questions, each worth two points. For each question, **circle all the letters that apply**.

1. Which of the following algorithms are guaranteed to compute the **global** optimum of their respective objectives (minimize cost, minimize loss, or maximize value) for the problems they are applicable to? Circle all that apply.

   a. k-means

   b. value iteration

   c. backtracking search

   d. dynamic programming

   e. iterated conditional modes

   Answer: b, c, d. K-means and iterated conditional modes are only guaranteed to converge to local optima, not global optima.

2. Which of the following are valid ways to reduce overfitting? Circle all that apply.

   a. Removing some feature templates.

   b. Performing early stopping when optimizing the training objective using SGD.

   c. Constraining the norm (length) of the weight vector to be at most 1.

   d. Setting some of the feature weights to be zero.

   e. Replacing a single more complex feature (e.g., x^{10}) with a simpler one (e.g., x).

   Answer: a, b, c, d. Only the last one is not valid because we are only changing one feature, not reducing the number of features. If we had replaced the features $x, x^2, \ldots, x^{10}$ with x, that would reduce overfitting.

3. Which of the following algorithms can be used to find the minimum number of actions needed to reach an end state from the start state in an arbitrary search problem? Circle all that apply.

   a. depth-first search

   b. breadth-first search

   c. uniform cost search

   d. dynamic programming

   e. backtracking search

   Answer: b, c, e. To minimize the number of actions, we set all action costs to a positive constant (say 1). This is exactly the condition for which breadth-first search works. Since all costs are non-negative as well, uniform cost search works. Dynamic programming requires acyclicity, which we are not guaranteed. Backtracking search always works.

4. Which of the following reinforcement learning algorithms estimate some quantity based on experience, from which we can approximately compute the **optimal policy**? Circle all that apply.

   a. model-based Monte Carlo

   b. model-free Monte Carlo

   c. SARSA

   d. Q-learning

   e. TD learning

   Answer: a, d. Model-based Monte Carlo estimates the transitions and rewards, from which we can derive everything by solving an MDP. Q-learning estimates the Q-value of the optimal policy, while the others estimate the Q-value of a fixed policy. Note that all of these algorithms are approximate in the sense that they are not guaranteed to converge to the correct values, especially after a finite number of experiences (that is, (s, a, r, s') tuples).

5. What are some advantages of variable-based models compared to state-based models? Circle all that apply.

   a. They allow you to encode problems that are not expressible using state-based models.

   b. They can lead to more efficient algorithms that can leverage the structure of a factor graph.

   c. They can be more natural to use for modeling in cases where the order of actions doesn't matter.

   d. They allow us to solve problems requiring reasoning under uncertainty.

   e. They allow us to learn the model from data.

   Answer: b, c. Anything you can do with a variable-based model you can also encode using a state-based model, but it's more natural to the language of variables and factors when the order of actions doesn't matter. Both formalisms can deal with uncertainty (e.g., MDPs and Bayesian networks), and both types of models can be learned (e.g., using structured Perceptron, reinforcement learning, EM).

2. Delivery (*40 points*)

A Star Delivery Company (ASDC) has hired you as a consultant to optimize their package delivery system. There are n locations numbered 1 to n connected by a system of roads; the set of roads is denoted R. Each road $(i, j) \in R$ connects location i with location j. Define $\mathcal{N}(i) = \{j : (i, j) \in R\}$ to be the locations reachable by a single road from i. It takes an integer number of minutes $t_{i,j} \geq 0$ to traverse the road from i to j, costing $c_{i,j} \geq 0$ dollars. We wish to find the *cheapest* path that goes from location 1 to location n that does not exceed a given time budget T (otherwise customers will be unhappy). Assume that you can only move forwards (i.e. the graph is acyclic).

For example, if $n = 3$ with roads $R = \{(1, 2), (2, 3)\}$, costs $c_{1,2} = 10$ and $c_{2,3} = 30$, and times $t_{1,2} = 20$ and $t_{2,3} = 10$, then there is one path with total cost $10 + 30$ and total time $20 + 10$.

a. (*10 points*)

Your first instinct is to define a search problem to solve the delivery problem. States look like $s = (i, t)$, where i is the current location and t is the number of minutes that has elapsed so far. Actions look like j, which means traveling to location j. Fill in the specification of the search problem. You must be precise about the valid actions.

- $s_{\text{start}} =$
- $\text{Actions}((i, t)) =$
- $\text{Succ}((i, t), j) =$
- $\text{Cost}((i, t), j) =$
- $\text{IsEnd}((i, t)) =$

Solution The main thing to keep in mind is that an action is only valid if it says within the time budget.

- $s_{\text{start}} = (1, 0)$
- $\text{Actions}((i, t)) = \{j \in \mathcal{N}(i) : t + t_{i,j} \leq T\}$ (traveling to location j)
- $\text{Succ}((i, t), j) = (j, t + t_{i,j})$
- $\text{Cost}((i, t), j) = c_{i,j}$
- $\text{IsEnd}((i, t)) = [i = n]$

b. (*10 points*)

ASDC wants to understand the tradeoff between time and money more precisely. They have asked you for a graph that plots a time budget T against the cost of the minimum cost path from location 1 to location n given time budget T. Let P_T be the search problem with time budget T. Solving P_T separately for each integer value $T = 1, \ldots, T_{\max}$ turns out to be too slow. However, you realize that the search problems for adjacent values of T are quite similar, so maybe A* is applicable.

Assume you have access to:

1. An A* implementation that takes in a search problem and a consistent heuristic and returns the minimum cost path.
2. A dynamic programming (DP) implementation that takes in a search problem and computes the future cost for each state (that is, the minimum cost to get to an end state).

Your job is to devise an algorithm that computes the minimum cost path for *each* $T = 1, \ldots, T_{\max}$. Your algorithm should loop over the values of T in some order. Recall that DP requires time linear in the number of states, which is too expensive to call at each time step, so let's run DP every 10 time steps—that is, for each $T \bmod 10 = 0$ (assume that $T_{\max}$ is divisible by 10), run dynamic programming to obtain future costs. Then, for each T, construct a consistent heuristic based on the results of DP and run A* (which should hopefully run in time much less than the number of states). This is only a sketch of the algorithm. Work out the details, write the pseudocode for your algorithm, and justify why it works.

Solution For $T = T_{\max}, \ldots, 1$:

1. If T divisible by 10, run dynamic programming on P_T to obtain future costs.
2. Run A* on P_T with heuristic $h(s) =$ future cost of s in $P_{T'}$ where $T' = 10\lceil T/10 \rceil$.

It is important to start with $T = T_{\max}$ (least constrained) and proceed to $T = 1$ (most constrained). Note that problem $P_{T'}$ is a relaxation of problem P_T because it has at least as many available actions. Therefore, the future costs of $P_{T'}$ are a consistent heuristic for P_T.

c. (*10 points*)

Everyone at ASDC is super impressed by your algorithmic skills. Customers are super happy because no packages are ever delivered late, while the company is still able to keep costs to a minimum. But there is a new problem. ASDC has become a target for some troublemakers who are blocking roads. What's apparently going on is that whenever the delivery truck arrives at a location i, an adversary chooses some road (i, j^*) to block, preventing the truck from choosing road (i, j^*). The truck must now choose some road (i, j) for $j \neq j^*$. Note that the blockage is temporary; if the truck ever revisits that i, it can choose (i, j^*) if the adversary chooses another road to block.

Let us focus on the simpler version of the problem where there are no time constraints. We simply want to go from location 1 to location n with the minimum cost. Also, assume that each location i has more than one outgoing edge so that it is impossible for the adversary to completely trap the truck at any location. Lastly, the adversary is smart enough to block a road which will make the truck spend as much time to reach the destination as possible.

Write a recurrence for $M(i)$, the cost of the minimum cost path from i to n. Remember that for each state, the adversary chooses a road to block before you choose which road to take.

Solution

$$M(i) = \begin{cases} 0 & \text{if } i = n \\ \max_{j^* \in \mathcal{N}(i)} \min_{j \in \mathcal{N}(i) \setminus j^*} \{c_{i,j} + M(j)\} & \text{otherwise.} \end{cases} \tag{1}$$

Note that the adversary is maximizing and we are minimizing.

Describe an efficient algorithm to compute $M(1)$. What is the total running time as a function of n? Assume that there could be $O(n)$ roads out of each location. Choose from one of the following: $O(1)$, $O(n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, $O(n^5)$, $O(2^n)$, $O(3^n)$, $O(4^n)$, $O(5^n)$, and justify your answer. You must find the most efficient algorithm to receive full points.

Solution Let i be a location. For each $j \in \mathcal{N}(i)$, we compute $d_{i,j} \stackrel{\text{def}}{=} c_{i,j} + M(j)$. The adversary should choose to block the j^* that obtains the smallest value in $\{d_{i,j} : j \in \mathcal{N}(i)\}$. So we simply set $M(i)$ to the second smallest value in $\{d_{i,j} : j \in \mathcal{N}(i)\}$. This algorithm is $O(n^2)$. Naively following the max-min structure of the recurrence leads to a suboptimal $O(n^3)$ algorithm.

d. (*10 points*)

ASDC is fed up with these troublemakers, so it decides that it needs to install security cameras to *cover* all the locations. Cameras are installed at locations; a location i is covered if there are at least 3 cameras installed among it and its neighboring locations $\mathcal{N}(i)$. We wish to find the minimum number of cameras to install so that each location is covered.

Let us set up a factor graph to solve this problem. Let the variables be $X_i \in \{0, 1\}$ for $i = 1, \ldots, n$, where X_i indicates whether we will install a security camera at location i.

Define the factors so that the maximum weight assignment of the resulting factor graph tells us the minimum number of cameras required to cover all the locations.

Solution We define two types of factors.

- The first takes the objective into account:

$$g_i(x_i) = \exp(-x_i) \quad \text{for } i = 1, \ldots, n.$$

  These factors are constructed so that maximizing $\prod_{i=1}^{n} f_i(x_i)$ is equivalent to minimizing $\sum_{i=1}^{n} x_i$.

- The second takes the coverage constraints into account:

$$h_i(x_i, x_{\mathcal{N}(i)}) = \left[x_i + \sum_{j \in \mathcal{N}(i)} x_j \geq 3 \right], \quad \text{for } i = 1, \ldots, n.$$

3. Taking an Exam (*50 points*)

Suppose you're taking an exam. Which problems should you work on to maximize your score? Suppose the exam has k problems, each with n parts. Part j of problem i is worth $x_{i,j}$ points.

At the beginning of each minute of the total allotted time T, you've solved some problem-part pairs (i, j) but not others. You need to choose a problem-part pair (i, j) to work on, and you can only choose (i, j) if (i, j) is unsolved and you've solved all previous parts in that problem ((i, j') for $j' < j$).

After that minute, with probability $p_{i,j}$ you solve the part and receive $x_{i,j}$ points; with probability $1-p_{i,j}$, you don't solve the part and receive no points (but can try again). Assume you know immediately whether you got points or not. The probability of solving a problem-part is independent of all previous problem-parts. Your goal is of course to maximize your total points (in expectation). You stop either (i) after T minutes or (i) when you have solved all parts of all problems (in which case you can leave the exam early).

For example, an exam might have $T = 180$ minutes, $k = 3$ problems and $n = 5$ parts per problem, with the following points:

Part	Problem 1	Problem 2	Problem 3
1	10	10	10
2	10	10	10
3	10	10	10
4	10	10	10
5	10	10	10

A possible sequence of actions is:

$$\text{(1,1):fail} \quad \text{(1,1):succeed} \quad \text{(3,1):succeed} \quad \text{(3,2):fail} \quad \text{(3,2):fail} \tag{2}$$

where (i, j) stands for part j of problem i. This sequence of actions would take 5 minutes and result in $10 + 10 = 20$ points.

a. (*10 points*)

You decide to formulate this situation as an MDP. Define a state of the MDP to be $s = (D, t)$, where D is the set of problem-part (i, j) pairs that have been solved thus far, and t is the integer number of minutes that have elapsed in the exam. An action of an MDP is an unsolved problem-part pair to work on. Fill out the MDP below. We expect detailed expressions for possible actions and successors. So be precise!

Notational hint: if you solve any problem-part (i, j) you work on with probability 1, then you should write the transition probabilities as:

$$\text{Trans}((D, t), (i, j), (D', t')) = [D' = D \cup \{(i, j)\} \wedge t' = t + 1].$$

- $s_{\text{start}} =$
- $\text{Actions}((D, t)) =$
- $\text{Trans}((D, t), (i, j), (D', t')) =$
- $\text{Reward}((D, t), (i, j), (D', t')) =$
- $\text{IsEnd}((D, t)) =$
- $\gamma =$

Solution

- $s_{\text{start}} = (\emptyset, 0)$
- $\text{Actions}((D,t)) = \{(i,j) \in (\{1,\ldots,k\} \times \{1,\ldots,n\}) - D : (i,j') \in D \text{ for all } j' < j\}$
- $T((D,t),(i,j),(D',t')) = p_{i,j}[D' = D \cup \{(i,j)\} \wedge t' = t+1] + (1-p_{i,j})[D' = D \wedge t' = t+1]$
- $\text{Reward}((D,t),(i,j),(D',t')) = x_{i,j}[(i,j) \in D' - D]$
- $\text{IsEnd}((D,t)) = [t = T \vee |D| = nk]$
- $\gamma = 1$

How many possible states are reachable from the starting state? Write your expression using big-Oh notation as a function of T, k, n. Your answer must have the correct dependence on these variables, and should be as tight as possible.

Solution At any point in time, for every problem, a prefix of the parts have be solved, and the time t could be at any point in time. So there are $O(n^k T)$ states. Naively, one might guess $O(2^{kn}T)$, since each of the kn problem-parts could either be in D or not, but this is too loose.

b. (*10 points*)

Let's try to solve the MDP now for a simple setting, where there are only two problems, each with one part. To simplify notation and terminology, we will just refer to the problem, not the problem-part pair; in other words, we will write 1 instead of $(1, 1)$ and 2 instead of $(2, 1)$.

- Problem 1 is worth 1 point and if you attempt it (action 1), you succeed with probability p, and
- Problem 2 is worth 2 points and if you attempt it (action 2), you succeed with probability q.

Define the possible sets of solved problems as follows:

$$D_0 = \emptyset, \quad D_1 = \{1\}, \quad D_2 = \{2\}, \quad D_3 = \{1, 2\}.$$

Write down the recurrence for V_{opt} and Q_{opt}, the expected utility (total number of points) of the optimal policy from a state and (state, action) pair, respectively. Below, take t, D to be any values for which $t < T$ and $D \in \{D_0, D_1, D_2, D_3\}$. You can include other V_{opt} or Q_{opt} terms in your answer. Hint: It may help to sketch out an MDP for this problem.

$V_{\text{opt}}((D_0, t)) =$ _____

$Q_{\text{opt}}((D_0, t), 1) =$ _____

$Q_{\text{opt}}((D_0, t), 2) =$ _____

$V_{\text{opt}}((D_1, t)) =$ _____

$V_{\text{opt}}((D_2, t)) =$ _____

$V_{\text{opt}}((D_3, t)) =$ _____

$V_{\text{opt}}((D, T)) =$ _____

Solution

$$
\begin{aligned}
V_{\text{opt}}((D_0, t)) &= \max(Q_{\text{opt}}((D_0, t), 1), Q_{\text{opt}}((D_0, t), 2)) \\
Q_{\text{opt}}((D_0, t), 1) &= p(1 + V_{\text{opt}}((D_1, t+1))) + (1-p)V_{\text{opt}}((D_0, t+1)) \\
Q_{\text{opt}}((D_0, t), 2) &= q(2 + V_{\text{opt}}((D_2, t+1))) + (1-q)V_{\text{opt}}((D_0, t+1)) \\
V_{\text{opt}}((D_1, t)) &= q(2 + V_{\text{opt}}((D_3, t+1))) + (1-q)V_{\text{opt}}((D_1, t+1)) \\
V_{\text{opt}}((D_2, t)) &= p(1 + V_{\text{opt}}((D_3, t+1))) + (1-p)V_{\text{opt}}((D_2, t+1)) \\
V_{\text{opt}}((D_3, t)) &= 0 \\
V_{\text{opt}}((D, T)) &= 0
\end{aligned}
$$

c. (*10 points*)

Suppose the exam is $T = 2$ minutes long (don't panic!), and that $p = 1$ (you are sure to solve problem 1 if you attempt it). Compute V_{opt} for the various arguments below as a function of q. Your expressions should only contain numbers, q, max, and arithmetic operations.

$D\backslash t$	0	1	2
D_0	_____	_____	_____
D_1	n/a	_____	_____
D_2	n/a	_____	_____
D_3	n/a	n/a	_____

Solution

$D\backslash t$	0	1	2
D_0	$\max(1+2q, q(2+1)+(1-q)\max(1,2q))$	$\max(1,2q)$	0
D_1	n/a	$2q$	0
D_2	n/a	1	0
D_3	n/a	n/a	0

Describe the optimal action and value from the starting state for every value of $q \in [0, 1]$. Fill in the ranges and circle actions 1, 2, or both if either achieves the same value. If a range turns out to contain only one number, you should write $0 \leq q \leq 0$ or $1 \leq q \leq 1$.

Range of q	optimal initial action(s)	optimal value
$0 \leq q \leq$ _____	1 / 2	_____
_____ $< q <$ _____	1 / 2	_____
_____ $\leq q \leq 1$	1 / 2	_____

Finally, write one or two sentences describing how the optimal policy varies as q changes.

Solution Let us first consider $t = 1$. If $q \leq \frac{1}{2}$, then the optimal action in state D_0 is attempting problem 1, which yields value 1. Otherwise, if $q \geq \frac{1}{2}$, then the optimal action is attempting problem 2, which yields value $2q$. Now consider $t = 0$. If $q \leq \frac{1}{2}$, then both actions yield value $1 + 2q$, so either action is optimal. Otherwise, for $q \geq \frac{1}{2}$, we are taking the max of $1 + 2q$ (attempt problem 1) and $3q + (1 - q)2q$ (attempt problem 2). By doing some algebra (using the quadratic formula), we find that the two are equal exactly when $q = \frac{1}{2}$ and $q = 1$; and for $\frac{1}{2} < q < 1$, problem 2 is the optimal action. Summarizing:

Range of q	optimal initial action(s)	optimal value
$0 \leq q \leq \frac{1}{2}$	1 and 2	$1 + 2q$
$\frac{1}{2} < q < 1$	2	$5q - 2q^2$
$1 \leq q \leq 1$	1 and 2	3

Intuitively, when q is small (problem 2 is hard), then we might as well try to solve it first, and if it doesn't work, then fall back on the easy win (problem 1). If we have a good chance of solving problem 2, then we want to make sure we have two tries at solving it. The case of $q = 1$ is special; we can solve the two problems deterministically in any order because we're just that good.

d. (*10 points*)

Now let us go back to the general exam case with k problems and n parts per problem. In reality, you don't know the success probabilities $p_{i,j}$.[1] Suppose each problem-part is associated with a feature vector $\phi_{i,j} \in \mathbb{R}^d$ (e.g., features might include what the topic of the problem is, whether it involves proofs, etc.) Suppose for each minute in the first hour ($t = 0, \ldots, 59$), you attempt a problem $(I[t], J[t])$ and observe whether it succeeded or not $y_t \in \{0, 1\}$.

You want to now fit a linear model with weights $\mathbf{w}$ so that $p_{i,j} \approx \hat{p}_{i,j}(\mathbf{w}) \stackrel{\text{def}}{=} \sigma(\mathbf{w} \cdot \phi_{i,j})$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function. Every weight vector $\mathbf{w}$ defines some likelihood (probability) of the data. For example, if $\hat{p}_{(1,1)}(\mathbf{w}) = 0.7$, $\hat{p}_{(3,1)}(\mathbf{w}) = 0.2$, $\hat{p}_{(3,2)}(\mathbf{w}) = 0.1$, then the likelihood of the following data

$$\text{(1,1):fail} \quad \text{(1,1):succeed} \quad \text{(3,1):succeed} \quad \text{(3,2):fail} \quad \text{(3,2):fail}, \tag{3}$$

would be $(1 - 0.7) \cdot 0.7 \cdot 0.2 \cdot (1 - 0.1) \cdot (1 - 0.1) = 0.03402$.

Define a training loss function TrainLoss($\mathbf{w}$) that corresponds to the negative log likelihood of the training data:

TrainLoss($\mathbf{w}$) = _____

Solution

$$\text{TrainLoss}(\mathbf{w}) = \sum_{t=0}^{59} \log(1 + \exp(-(2y_t - 1)\mathbf{w} \cdot \phi_{I[t],J[t]})). \tag{4}$$

Now, with a trained model, you can cast the problem at hand as the MDP in (a)! From a reinforcement learning perspective, which of the following algorithms best characterizes this approach at a high-level? Circle the one that applies.

- model-based Monte Carlo
- model-free Monte Carlo
- bootstrapping

Solution Here, we are explicitly estimating $p_{i,j}$ which governs the transitions in the MDP. Therefore, this is most similar to a model-based Monte Carlo method.

[1] Assume you do know all point values $x_{i,j}$, or else it would be a mean exam!

e. (*10 points*)

The previous method required us to stop and solve an entire optimization problem in the middle of an exam. Seriously, who has time for that? Let us use reinforcement learning instead. Now the weight vector $\mathbf{w}$ is used to parametrize the Q function corresponding to the expected utility under the optimal policy:

$$Q((D,t),(i,j);\mathbf{w}) \stackrel{\text{def}}{=} \mathbf{w} \cdot \phi_{i,j}. \tag{5}$$

Suppose we are in a state (D,t) and attempt problem-part (i,j). Let $y \in \{0,1\}$ denote whether we managed to solve it or not, where $y = 0$ denotes failure and $y = 1$ denotes success. This defines a little piece of experience that we should be able to use to update $\mathbf{w}$. Write the Q-learning update in terms of the specific quantities $\mathbf{w}, \phi, y, x$, not general notation (e.g., s, a, r). You might find it useful to write down what the states, actions and rewards are explicitly and then match that with the general Q-learning update formula. Be precise (especially about what actions are being considered)!

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}. \tag{6}$$

Solution The current state is $s = (D,t)$, action is (i,j), the reward is $yx_{i,j}$. The new state is $(D', t+1)$, where $D' = D$ if $y = 0$ (failed) and $D' = D \cup \{(i,j)\}$ if $y = 1$ (succeeded). The set of new actions from the new state is all (i',j') that are not already solved (in D').

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\mathbf{w} \cdot \phi_{i,j} - (yx_{i,j} + \max_{(i',j') \in \text{Actions}((D',0))} \mathbf{w} \cdot \phi_{i',j'}))\phi_{i,j}. \tag{7}$$

4. Bayesian Lights (*50 points*)

This holiday season, you decide to put your knowledge of Bayesian networks to good use. You decide to create Bayesian Lights™, an arrangement of lights that turn on and off randomly according to the joint distribution of a Bayesian network.

Figure **??** shows the Bayesian network corresponding to the lights. Each light is associated with a variable which takes on values in $\{0, 1\}$ (off: 0, on: 1). For example, A_1 is the light in the upper-left corner and E_1 is the light at the very bottom. A light in the top row is on with probability α. The status of a light in subsequent rows is governed by the two parent lights directly above it, and it is on with probability β when the two parent lights above it have different statuses (on-off or off-on), and off with probability β when the two parent lights above it have the same status. In other words, each light is the result of applying a noisy XOR function.

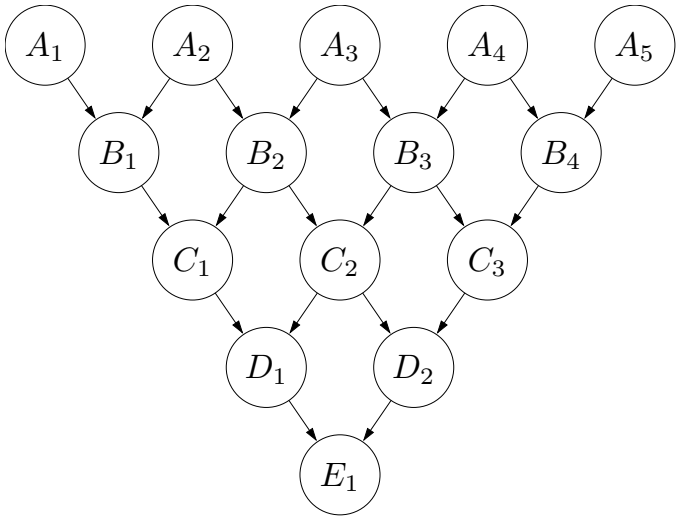

Figure 1: Bayesian Lights™ has 15 lights arranged in a triangle, whose on-off statuses are governed by this Bayesian network.

Formally, the local conditional distributions are given according to:

- Top row lights:

$$p(a_i) = \alpha.$$

- Second row lights:

$$p(b_i \mid a_i, a_{i+1}) = \beta \cdot [b_i = a_i \oplus a_{i+1}] + (1 - \beta) \cdot [b_i \neq a_i \oplus a_{i+1}].$$

The distribution of the lights in the remaining rows are analogously defined.

a. (*10 points*)

After setting up the lights, you are so mesmerized by the flickering that you suddenly feel like you can compute probabilistic inference queries in your head. Wow! But just to check that you're right, let's work it out by hand.

Write the answer to the following probabilistic inference queries in terms of α, assuming that $\beta = 1$.

1. $\mathbb{P}(A_3 = 1) =$
2. $\mathbb{P}(A_3 = 1 \mid B_1 = 1, B_4 = 1) =$
3. $\mathbb{P}(A_3 = 1 \mid B_2 = 1, B_3 = 1) =$

Solution

1. We can marginalize out all other variables away, so we're only left with A_3. Therefore

$$\mathbb{P}(A_3 = 1) = p(a_3) = \alpha.$$

2. When we marginalize out B_2, B_3 and all their descendants, we see that A_3 is independent of B_1 and B_4. Therefore, the probability is simply

$$\mathbb{P}(A_3 = 1 \mid B_1 = 1, B_4 = 1) = p(a_3) = \alpha.$$

3. We can marginalize out all other variables except A_3, A_2, A_4, B_2, B_3. When we condition on $B_2 = 1, B_3 = 1$, we get the following factors:

$$p(a_3), p(a_2), p(b_2 = 1 \mid a_2, a_3), p(a_4), p(b_3 = 1 \mid a_3, a_4).$$

Let us marginalize out A_2: this produces a new factor

$$f_2(a_3) = \sum_{a_2} p(a_2)p(b_2 = 1 \mid a_2, a_3).$$

Plugging in concrete values for a_3, we get: $f_2(1) = 1 - \alpha$ and $f_2(0) = \alpha$. Marginalizing out A_4 is exactly the same by symmetry and yields a factor $f_4 = f_2$.
Now, let us multiply all the factors together:

$$g(1) = p(a_3 = 1)f_2(1)f_4(1) = \alpha(1 - \alpha)^2 \tag{8}$$
$$g(0) = p(a_3 = 0)f_2(0)f_4(0) = (1 - \alpha)\alpha^2. \tag{9}$$

Now, we normalize g to get the desired conditional probability:

$$\mathbb{P}(A_3 = 1 \mid B_2 = 1, B_3 = 1) = \frac{g(1)}{g(0) + g(1)} = 1 - \alpha.$$

b. (*10 points*)

Assume $\beta = 1$ again. After staring at the lights some more, one particular query is particularly intriguing. What happens to the top-right light if the entire left side is on? Compute this quantity and justify your answer:

$$\mathbb{P}(A_5 = 1 \mid A_1 = B_1 = C_1 = D_1 = E_1 = 1) = \_\_\_\_\_ \tag{10}$$

Solution We can simply go through all variables with subscript 2, then 3, then 4. Conditioning on $A_1 = B_1 = 1$ forces $A_2 = 0$; $B_1 = C_1 = 1$ forces $B_2 = 0$; $C_1 = D_1 = 1$ forces $C_2 = 0$; $D_1 = E_1 = 1$ forces $D_2 = 0$. For the next column, $A_2 = B_2 = 0$ forces $A_3 = 0$, and so on. The remaining variables are all forced to be zero. Therefore:

$$\mathbb{P}(A_5 = 1 \mid A_1 = B_1 = C_1 = D_1 = E_1 = 1) = 0. \tag{11}$$

c. (*10 points*)

Now let us assume $\beta < 1$. The first version of Bayesian Lights$^{\text{TM}}$ would change to generate a completely independent setting of all the lights every second (by sampling from the local conditional distributions starting at the top and going down). However, this led to a lot of flickering. So you decide to use Gibbs sampling instead, where every 100 milliseconds, you choose a random light and sample it conditioned on everything else.

Let's work out the conditional distribution for C_2 given that all other lights are off. Your expression on the LHS should specify the Markov blanket of C_2 and the RHS should only depend on α and β.

$$\mathbb{P}(C_2 = 1 \mid \_\_\_\_\_) = \_\_\_\_\_$$

Solution First define the unnormalized distribution:

$$f(c_2) = p(c_2 \mid b_2 = 0, b_3 = 0)p(d_1 = 0 \mid c_1 = 0, c_2)p(d_2 = 0 \mid c_2, c_3 = 0)$$
$$f(1) = (1-\beta)^3$$
$$f(0) = \beta^3.$$

Normalizing yields the conditional distribution:

$$\mathbb{P}(C_2 = 1 \mid B_2 = B_3 = C_1 = D_1 = C_3 = D_2 = 0) = \frac{(1-\beta)^3}{(1-\beta)^3 + \beta^3}.$$

If we run Gibbs sampling for a very long time and record all the assignments to A_1, approximately what fraction of the time will $A_1 = 1$?

Solution Gibbs sampling generates samples over joint assignments that eventually match the true distribution, so since $\mathbb{P}(A_1 = 1) = \alpha$, approximately an α fraction of the samples will have $A_1 = 1$.

d. (*10 points*)

You lost the code to Bayesian Lights™, so you don't remember what values of α and β you used. Fortunately, you can turn on the lights for a while and collect some data, and try to reverse engineer α and β.

Suppose you observed the following:

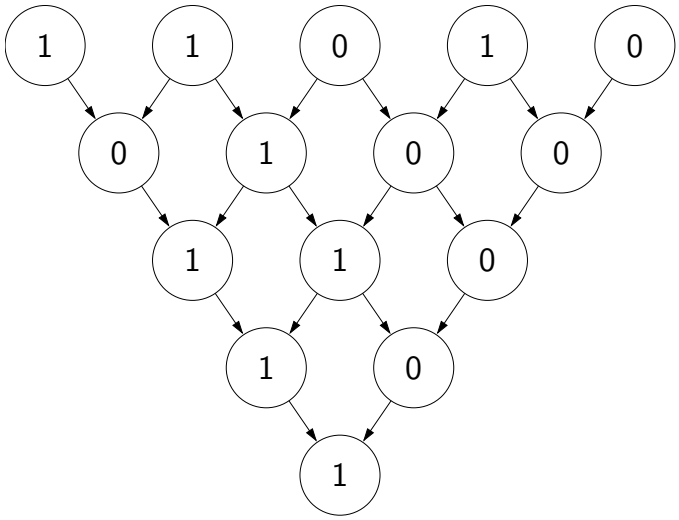

Figure 2: The status of Bayesian Lights™ at one point in time.

Compute the maximum likelihood estimate of α and β.

$\alpha =$ _____

$\beta =$ _____

Solution There are 3 1's in the top row, which are generated from α. For the rest of the variables, there are 6 variables which respect the XOR (generated from β). Therefore, $\alpha = \frac{3}{5}$ and $\beta = \frac{6}{10}$.

e. (*10 points*)

While staring at the statuses represented by Figure **??**, you realize that A_1 and E_1 are broken; their statuses are not reliable, though the underlying circuitry is still working. So you need to re-estimate α and β, but now without knowing the values of A_1 and E_1. Initialize with $\alpha = \frac{1}{2}$ and $\beta = \frac{3}{4}$, and run one step of the EM algorithm.

E-step:

$$\mathbb{P}(A_1 = 1 \mid \cdots) = \_\_\_\_\_$$

$$\mathbb{P}(E_1 = 1 \mid \cdots) = \_\_\_\_\_$$

Solution In the E-step, we use our current setting of the parameters to guess the distribution over the missing variables.

$$\mathbb{P}(A_1 = 1 \mid B_1 = 0, A_2 = 1) = \frac{\alpha\beta}{\alpha\beta + (1-\alpha)(1-\beta)} = \frac{3}{4}, \tag{12}$$

$$\mathbb{P}(E_1 = 1 \mid D_1 = 1, D_2 = 0) = \beta = \frac{3}{4}. \tag{13}$$

M-step:

$$\alpha = \_\_\_\_\_$$

$$\beta = \_\_\_\_\_$$

Solution Now we just use these probabilities as fractional counts in the computation of the maximum likelihood estimate ($\frac{3}{4}$ instead of the 1 from before):

$$\alpha = \frac{2 + \frac{3}{4}}{5} = \frac{11}{20}, \tag{14}$$

$$\beta = \frac{5 + \frac{3}{4}}{10} = \frac{23}{40}. \tag{15}$$

$$\tag{16}$$