



## 1. (10 points) True or False :

BN: Bayesian Network, HMM: Hidden Markov-Model, PF: Particle Filter, ML: Machine learning, MDP: Markov Decision Process, TD: Temporal Difference

False Inference in BNs: Worst case complexity of variable elimination is better than enumeration.

False In likelihood weighting for approximate inference in BNs, we do not need to use the conditional probability tables of the non-evidence variables

False Emission models in HMMs denote the probability of a state given the observation.

True In the PF version we have seen for inference in HMMs, effective number of particles may change over time.

True Collecting more data can solve over-fitting issues in ML.

False Training data performance is always a good indicator of the general performance of an ML model.

True Solving a problem modeled by an MDP results in a reflex-agent.

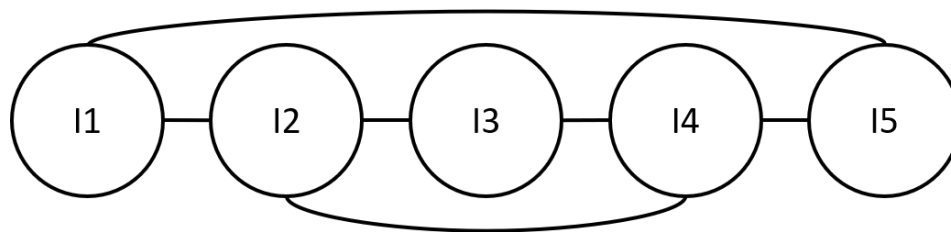
False In an MDP, the values of the states must be calculated exactly to extract an optimal policy

False Knowing the reward function is enough to extract a policy from TD learning.

False The discount factor in an MDP is not needed for approximate Q-learning

## 2. (5 points) Constraint Satisfaction Problems:

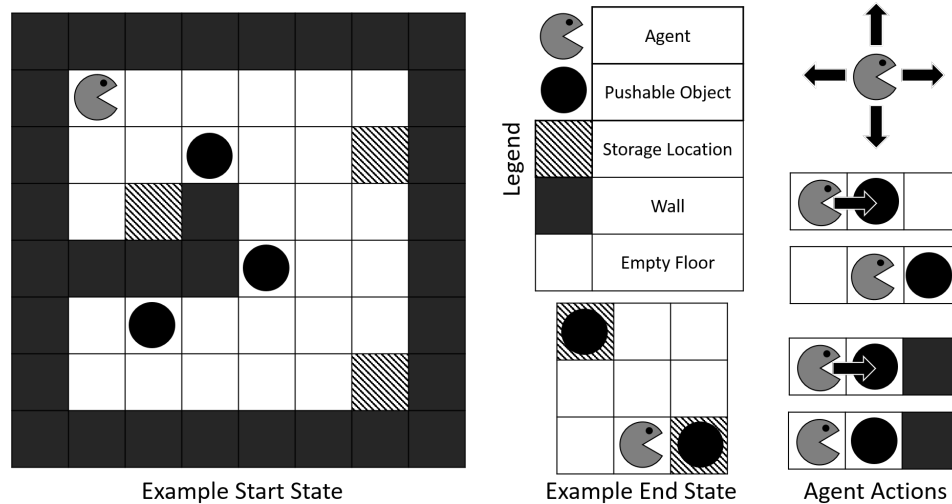
Apply the constraint propagation algorithm (AC3) on the following variables with binary inequality constraints given by the graph.



	I1	I2	I3	I4	I5
Init	R2, R3	R1	R1, R2, R3	R1, R3	R1, R2, R3
Answer	R2, R3	R1	R2	R3	R1, R2

1 point for I2 eliminations, 1.5 for I4

3. (12 points) Sokoban is a puzzle game in which an agent pushes around objects in a maze, trying to get the objects to storage positions. The game is played on a grid. Each grid location can either be a wall or a floor. Some floor locations contain objects and some storage locations. The agent can be on a floor unless it contains an object. The agent can move in 4 cardinal directions on the grid (*up, down, left, right*). If there is an object on a grid location that the agent is trying to move to, then it pushes the object as long as the next grid location is not a wall. If there is a wall, then nothing happens. The agent cannot pull the objects. The game ends when all the objects are on the storage locations. See the below figure for a visual description:



Formulate Sokoban as a search problem. Which search algorithm would you choose to use and why? What would be your cost and/or heuristics if your algorithm requires them?

One way to formulate it is given below.

**State Space ( $S$ ):** Each element in the state space is a tuple  $(F, O, L, A)$  where (2 points):

$F$  is a boolean matrix representing the grid. Each entry represents wall or floor

$O$  is a list of object locations. Each location is 2D.

$L$  is a list of target locations. Each location is 2D.

$A$  is the agent location in 2D.

**Actions ( $A$ ):** As given above *up, down, left, right*

**Transition Model ( $S \times A \rightarrow S$ )** (2 points including the actions):

Most of it is written in the text. The main point is to point out that this is part of the formulation and it is about the states and actions. The actions only change the subset of the state space ( $A$  and  $O$ ).

**Cost ( $S \times A \rightarrow c$ ):** Each action can be taken as unit cost. (0.5 points)

**Start State ( $s_0$ ):** The given start space (0.5 points)

**Goal Test ( $g(s)$ ):** Returns true if the objects ( $O$ ) are on the targets ( $L$ ), else returns false (1 point)

**Bonus:** Add an extra function to detect when the state would never lead to a solution (e.g. when an object is pushed to a corner) (1 point). This can be used to prune the state space to be searched.

**Algorithm:** The only choice is  $A^*$  (based on the class). Sokoban definitely requires a good heuristic to be solved due to its state space complexity. You should be able to come to this conclusion. Note that even with  $A^*$ , Sokoban is not easy to solve for complex and/or medium sized mazes (1 point for algorithm, 2 points for reasoning).

**Heuristics:** Cost of moving the agent to all un-occupied target locations. Variations include ignoring objects and/or ignoring walls. Additional heuristics would be the distance of each object to its closest target location. Other heuristics can be built by combining ideas (move agent to objects, then objects to locations) (3 points for admissible, 2 points for anything that makes sense)

4. (10 points) **Propositional Logic:** Show whether  $C$  is entailed by the following knowledge base using **resolution**. Any other method will not receive any credit. You can make smart choices. Clearly indicate the pair of sentences that you resolve. Make sure to give identifiers to the newly created resolvents and use these identifiers in the further steps.

$$S1: (\neg A \wedge B \wedge F) \Rightarrow C \rightarrow A \vee \neg B \vee C \vee \neg F$$

$$S2: (I \wedge J) \Rightarrow (C \vee H) \rightarrow C \vee H \vee \neg I \vee \neg J$$

$$S3: \neg H \Rightarrow \neg D \rightarrow \neg D \vee H$$

$$S4: A \vee \neg D$$

$$S5: B \wedge \neg D \Rightarrow F \rightarrow \neg B \vee D \vee F$$

$$S6: J \vee H$$

$$S7: \neg D \wedge F \Rightarrow I \rightarrow D \vee \neg F \vee I$$

$$S8: \neg A \vee D$$

$$S9: B$$

$$S10: \neg H$$

$$S11: E$$

$$S12: J$$

First step is to convert everything to CNF (2 points). Then add  $\neg C$  to the knowledge base and do resolution. One way to do it:

$$S13: \neg C$$

$$S14: R(S3, S10) \rightarrow \neg D$$

$$S15: R(S5, S14) \rightarrow \neg B \vee F$$

$$S16: R(S9, S15) \rightarrow F$$

$$S17: R(S1, S16) \rightarrow A \vee \neg B \vee C$$

$$S18: R(S9, S17) \rightarrow A$$

$$S19: R(S8, S18) \rightarrow D$$

$$S20: R(S14, S19) \rightarrow \emptyset$$

Contradiction. We can conclude that the KB entails  $C$

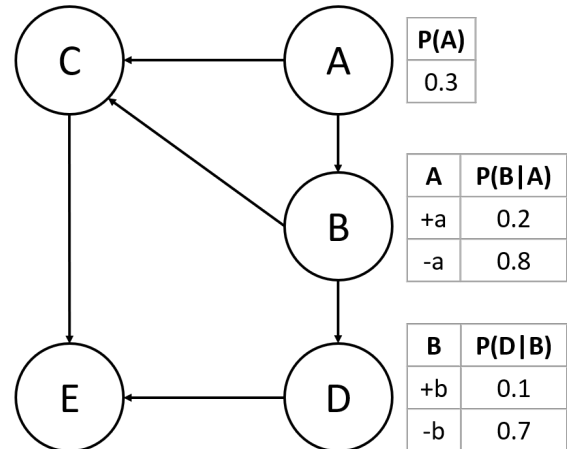
5. (18 points) Consider the Bayesian Network given below. All variables are binary and *True* versions are denoted by a plus sign followed by a lower case letter. The CPTs are given for only the *True* values. You subtract this from 1 to get the probabilities for false. E.g.  $P(-a) = 1 - 0.3 = 0.7$ ,  $P(-b|-a) = 1 - 0.8 = 0.2$ ,  $P(-e|+c, -d) = 1 - 0.6 = 0.4$ . Answer the following questions and **show your work**.

- (a) (2 points) Write the formula for the joint distribution,  $P(A, B, C, D, E)$ .

$$P(A, B, C, D, E) = P(A) \cdot P(B|A) \cdot P(C|A, B) \cdot P(D|B) \cdot P(E|C, D)$$

A	B	P(C A,B)
+a	+b	0.4
+a	-b	0.7
-a	+b	0.8
-a	-b	0.9

C	D	P(E C,D)
+c	+d	0.5
+c	-d	0.6
-c	+d	0.3
-c	-d	0.1



P(A)
0.3

A	P(B A)
+a	0.2
-a	0.8

B	P(D B)
+b	0.1
-b	0.7

- (b) (6 points) Calculate  $P(A|+b, -e)$  using variable elimination. Clearly indicate your factors and your operations (join, sum out, normalize). Use the back of this page if you run out of space. **Initial factors:**

$$f_1(A), f_2(A, +b), f_3(A, +b, C), f_4(+b, D), f_5(C, D, -e)$$

Step 1: Join  $f_4$  and  $f_5$  to get  $f_6$

C	D	$f_6(+b, C, D, -e)$
+c	+d	$0.1 \cdot (1 - 0.5) = 0.05$
+c	-d	$0.9 \cdot (1 - 0.6) = 0.36$
-c	+d	$0.1 \cdot (1 - 0.3) = 0.07$
-c	-d	$0.9 \cdot (1 - 0.1) = 0.81$

Step 2: Sum out  $D$  from  $f_6$  to get  $f_7$

C	$f_7(+b, C, -e)$
+c	$0.05 + 0.36 = 0.41$
-c	$0.07 + 0.81 = 0.88$

Step 3: Join  $f_3$  and  $f_7$  to get  $f_8$

A	C	$f_8(A, +b, C, -e)$
+a	+c	$0.4 \cdot 0.41 = 0.164$
+a	-c	$0.6 \cdot 0.88 = 0.528$
-a	+c	$0.8 \cdot 0.41 = 0.328$
-a	-c	$0.2 \cdot 0.88 = 0.176$

Step 4: Sum out  $C$  from  $f_8$  to get  $f_9$

A	$f_9(A, +b, -e)$
+a	$0.164 + 0.528 = 0.692$
-a	$0.328 + 0.176 = 0.504$

Step 5: Join  $f_1$ ,  $f_2$  and  $f_9$  to get  $f_{10}$

A	$f_{10}(A, +b, -e)$
+a	$0.3 \cdot 0.2 \cdot 0.692 = 0.04152$
-a	$0.7 \cdot 0.8 \cdot 0.504 = 0.28224$

Step 6: Normalize  $f_{10}$  to get the answer!

A	$P(A +b, -e)$
+a	$\frac{0.04152}{0.04152 + 0.28224} = 0.1282$
-a	$\frac{0.28224}{0.04152 + 0.28224} = 0.8718$

- (c) (2 points) Suppose you want to calculate  $P(A|+b, -e)$  using rejection sampling. Give an example of a sample that would be rejected and another example that would not.

Rejected examples include anything that does not have  $(+b, -e)$ , e.g.  $(+a, -b, -c, +d, -e)$

Not rejected examples include everything with  $(+b, -e)$ , e.g.  $(+a, +b, -c, +d, -e)$

- (d) (6 points) Suppose you want to calculate  $P(A|+b, -e)$  using likelihood weighting. Write down all the possible combinations of samples and corresponding weights. (Hint: There are 8 unique samples.)

For this problem, all the non-evidence variables  $(A, C, D)$  affect the probabilities calculated in weights. Since they are binary, we have  $2^3 = 8$  unique cases as follows (0.75 points per item) :

- Sample  $(+a, +b, +c, +d, -e)$  with weight  $0.2 \cdot (1 - 0.5) = 0.1$
- Sample  $(+a, +b, +c, -d, -e)$  with weight  $0.2 \cdot (1 - 0.6) = 0.08$
- Sample  $(+a, +b, -c, +d, -e)$  with weight  $0.2 \cdot (1 - 0.3) = 0.14$
- Sample  $(+a, +b, -c, -d, -e)$  with weight  $0.2 \cdot (1 - 0.1) = 0.18$
- Sample  $(-a, +b, +c, +d, -e)$  with weight  $0.8 \cdot (1 - 0.5) = 0.4$
- Sample  $(-a, +b, +c, -d, -e)$  with weight  $0.8 \cdot (1 - 0.6) = 0.32$
- Sample  $(-a, +b, -c, +d, -e)$  with weight  $0.8 \cdot (1 - 0.3) = 0.56$
- Sample  $(-a, +b, -c, -d, -e)$  with weight  $0.8 \cdot (1 - 0.1) = 0.72$

- (e) (2 points) You have run sampling for likelihood weighting 1000 times given the evidence  $(+b, -e)$ . You have a total of 276 samples that include  $+a$ . Is this information enough to calculate the probability  $P(+a|+b, -e)$ ? If so, what is the value? If not, why not?

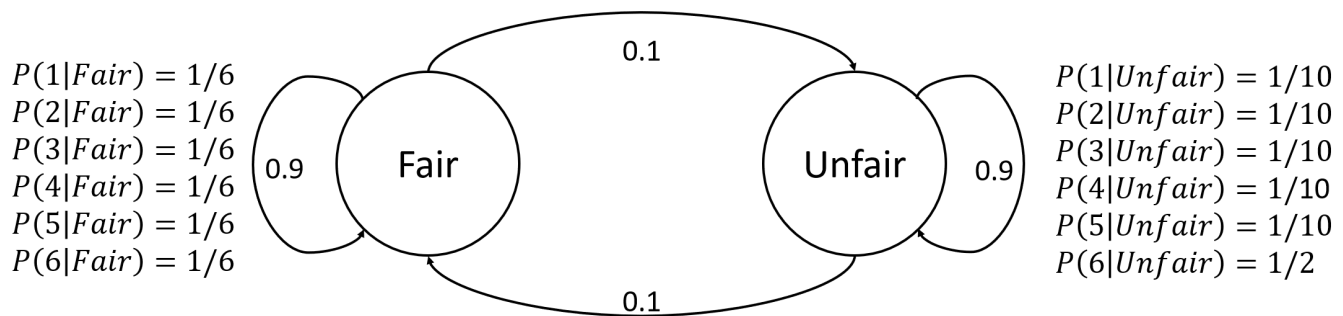
It is not enough since the sampling procedure for likelihood weighting not consistent (see the definition in the slides). That is why we calculate the weights.

## 6. (10 points) Reasoning over Time: Occasionally Dishonest Casino

Your employer sends you to Las Vegas for an exposition. You are done with all your work and have 30 minutes to kill before your shuttle takes you to the airport. You see a simple dice game and decide to play even though you know gambling is bad (Shame on you!). The rules of the game is as follows:

- You bet some money
- You roll a 6 sided die
- Casino player rolls a 6 sided die
- If your roll is higher, you get double your money. If not, you lose the money you bet

You suddenly remember two people talking about the casino switching back and forth between a fair die and an unfair die every 10 rolls and that the unfair die has a 50% chance of coming up 6, while all other numbers are uniform. Luckily, you went to the best CS department in Turkey and took the AI class. You formulate the problem in your head as a Hidden Markov Model with the following structure.



As you play you see the following sequence of die rolls  $\{1, 1, 6, 2, 6, 3, 4, 3, 6, 2, 6, 6\}$ . You have no idea which die the casino player started with.

- (a) (5 points) What is the likelihood of this sequence if the casino player used fair dice all the way based on the HMM? You do not need to calculate the final expression.

From the text we can infer the prior probabilities as:  $P(x_0 = Fair) = P(x_0 = Unfair) = 0.5$

Let the observation sequence be:  $e = \{e_1, \dots, e_{12}\} = \{1, 1, 6, 2, 6, 3, 4, 3, 6, 2, 6, 6\}$

Let the state sequence be:  $x = \{x_1, \dots, x_{12}\} = \{Fair, \dots, Fair\}$

There is some ambiguity in the text. The desired probability can be interpreted in two ways:

$$\text{Joint: } P(x, e) = P(x_0) \prod_{i=1}^{12} P(x_i | x_{i-1}) P(e_i | x_i) = 0.5 \cdot (1/6)^{12} \cdot (9/10)^{12}$$

$$\text{Conditional: } P(e|x) = \prod_{i=1}^{12} P(e_i | x_i) = (1/6)^{12}$$

I will accept both but not  $P(x|e)$ .

- (b) (5 points) Would you say that the casino player changed the die immediately after rolling two 1s?

There are two ways to approach this question:

1. Calculate the probability of each state for the second and the third die roll. See if the highest probability in each step is different.
2. Assume that the dice is Fair at roll 2. Calculate the state probabilities after the next roll given 6. See if Unfair probability is higher. (The opposite can also be done, starting from an Unfair dice assumption.)

Let  $F$  represent Fair and  $U$ , Unfair

The first approach:

After First Roll:

$$P(x_1 = F) = \alpha P(e_1 = 1 | x_1 = F) (P(x_0 = F)P(x_1 = F | x_0 = F) + P(x_0 = U)P(x_1 = F | x_0 = U)) \\ = (0.5 \cdot 0.9 + 0.5 \cdot 0.1) / 6 = \alpha / 12$$

$$P(x_1 = U) = \alpha P(e_1 = 1 | x_1 = U) (P(x_0 = F)P(x_1 = U | x_0 = F) + P(x_0 = U)P(x_1 = U | x_0 = U)) \\ = (0.5 \cdot 0.1 + 0.5 \cdot 0.9) / 10 = \alpha / 20$$

After we normalize:  $P(x_1 = F) = 5/8$  and  $P(x_1 = U) = 3/8$

After Second Roll:

Follow the same procedure to get  $P(x_2 = F) = 5/7$  and  $P(x_2 = U) = 2/7$

After Third Roll:

Follow the same procedure to get  $P(x_3 = F) = 47/116$  and  $P(x_3 = U) = 69/116$ . Note that the die roll was 6.

After the second roll, most likely state is Fair. After the third roll, the most likely state is Unfair. We can say that the casino player probably changed the dice! (Note that we did not need to normalize at any step.)

Second approach:

Assuming  $P(x_2 = F) = 1.0$

$$P(x_3 = F) = \alpha P(e_3 = 6 | x_3 = F) P(x_3 = F | x_2 = F) \\ = 1/6 \cdot 0.9 = 3 \cdot \alpha / 20$$

$$P(x_3 = U) = \alpha P(e_3 = 6 | x_3 = U) P(x_3 = U | x_2 = F) \\ = 1/2 \cdot 0.1 = \alpha / 20$$

Based on this approach, the casino player did not change the dice!

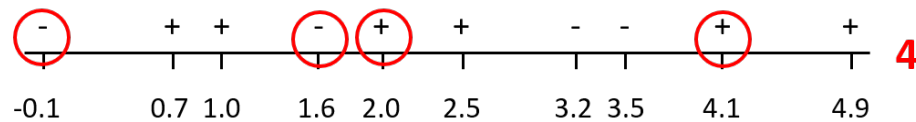
Since we have no idea which dice he started with, the first approach makes more sense but I will accept both approaches.



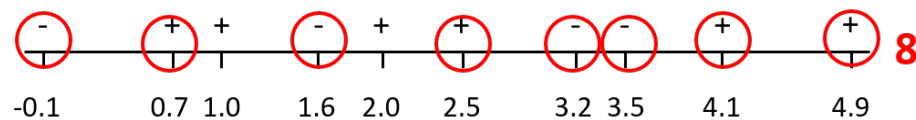
## 7. (10 points) Machine Learning:

- (a) (5 points) You are given a dataset (see below figures) with a one dimensional real valued input and a binary output (+ or -). You are going to use kNN with Euclidean distance to predict  $y$  for an input  $x$ .

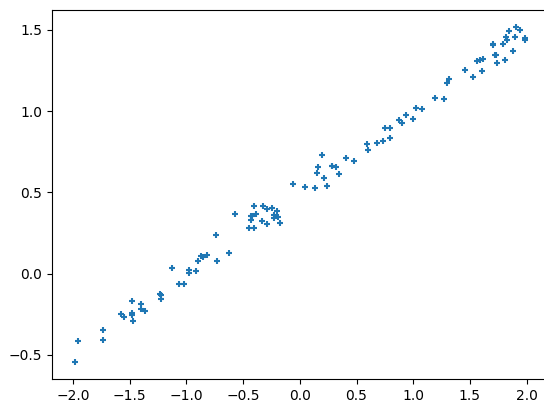
What is the leave-one-out cross-validation (Pick 1 point as a test, remaining as training data, do it for all points) error of 1-NN on this dataset. Give your answer as the number of misclassifications. Circle the misclassifications in the figure below.



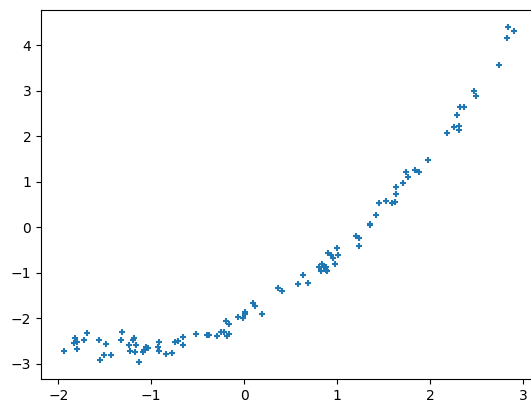
What is the leave-one-out cross-validation error of 3-NN on this dataset. Give your answer as the number of misclassifications. Circle the misclassifications in the figure below.



- (b) (5 points) In the given plots below, the horizontal axis corresponds to variable  $x$  and the vertical axis corresponds to variable  $y$ . You want to fit a function  $y = f(x)$  using basis function regression. Under each figure, write the form of  $f(x)$ . Underline the parameters you should learn from the data. Remember, you want to pick the simplest function that explains the data. Try to stick with polynomial basis functions.

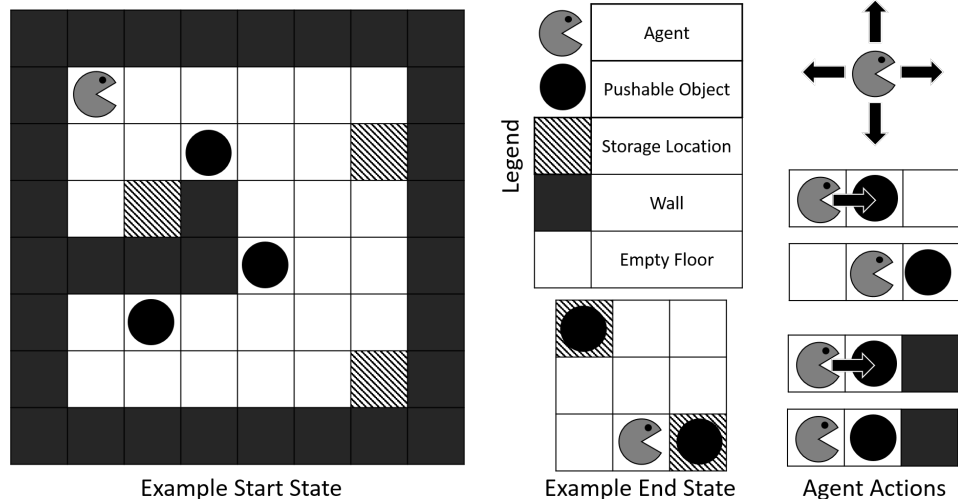


$$y = f(x) = w_0 + w_1 x$$



$$y = f(x) = w_0 + w_1 x + w_2 x^2$$

8. (12 points) Recall the sokoban from question 3 (displayed below for your convenience). Formulate sokoban as an MDP,  $\{S, A, T, \gamma, R\}$ . Feel free to directly copy parts of your answer to question 3 here if applicable. Which algorithm(s) would you use to come up with a policy for sokoban and **why**? Provide as much detail as you can about your answer. This does not imply that you should write a lot! The answer is not that long. Note that your answer must be realistic, i.e., if the algorithm you suggest is not practically applicable to sokoban, you will not get any credit for that part.



Copy paste  $S$ ,  $A$  and  $T$  from question 3. We need to decide on  $\gamma$  and  $R$ . Let's start with  $R$ . There are multiple options as given below. For certain conditions, the reward function should depend on the current-state, the action and the new state. The first two are mandatory, remaining are optional but recommended.


- Taking any action/any state transition: -1
- Transitioning to the goal state (satisfying the goal test in Q3): 1000 (or any sufficiently high positive number)
- Pushing an object to a target location: 50 (or any positive number that is appropriately selected)
- Pushing an object off of a target location: -50 (or any negative number that is appropriately selected)
- Transitioning to a dead-end state (see Q3) or any dead-end state: -1000 (or any sufficiently high negative number)

For the discount factor,  $\gamma$ , we need to think about the problem. Sokoban is closer to chess than Pacman in terms of time-horizon. The actions taken early can have a huge impact on the solution. Furthermore, reversing actions are costly and sometimes not possible so the agent should decide wisely. Thus, Sokoban would need a large  $\gamma$ . I would chose it as  $\gamma = 0.999$ .

Only relatively feasible method we have seen for Sokoban is approximate q-learning for anything but small puzzles due to the complexity of its state space. If we restrict ourselves to small puzzles, then value iteration, policy iteration or Q-learning would work. An approximate q-learning agent would not be able generalize well. Overall, the techniques we have seen in the class is not really suitable for Sokoban, especially for generalizing to multiple mazes.

For this problem, we need search-based (or more specifically planning) approaches. An idea is to use TD-learning to learn the state values and use this in a heuristic for A\*. If you are interested in doing learning for planning (e.g. learning heuristics, learning features for state representation), contact me! This is a hot topic.

9. (13 points) Consider the grid-world given below. The agent is trying to learn the optimal policy. If an action results in landing into one of the numbered states the corresponding reward is awarded during that transition. The states with numbers are terminal states. The other states have the *Up*, *Down*, *Left*, *Right* actions available which deterministically move the agent to the corresponding neighboring state or stay in place in case there is no cell to move to. Assume the discount factor  $\gamma = 0.5$ , the Q-learning rate  $\alpha = 0.5$  for all calculations and that the agent always starts in state (1,3) as shown below.

3		-80	+100
2			
1	+25	-100	+80
	1	2	3

- (a) (3 points) What is the value of the optimal value function  $V^*$  at the following states? Remember that actions are deterministic!

$$V^*(3, 2) = \underline{100}$$

$$V^*(2, 2) = \underline{50}$$

$$V^*(1, 3) = \underline{12.5}$$

- (b) (3 points) You are given the following episodes from runs of the agent in this environment. Each line of an episode contains a tuple of  $(s, a, s', r)$  format.

Episode 1	Episode 2	Episode 3
(1,3), <i>D</i> , (1,2), 0	(1,3), <i>D</i> , (1,2), 0	(1,3), <i>D</i> , (1,2), 0
(1,2), <i>R</i> , (2,2), 0	(1,2), <i>R</i> , (2,2), 0	(1,2), <i>R</i> , (2,2), 0
(2,2), <i>D</i> , (2,1), -100	(2,2), <i>R</i> , (3,2), 0	(2,2), <i>R</i> , (3,2), 0
	(3,2), <i>U</i> , (3,3), +100	(3,2), <i>D</i> , (3,1), +80

What are the following Q-value after the above three episodes as calculated by Q-learning? Assume that all the Q-values are initialized to 0 and that the Q-values are updated based on the order seen in the episode.

$$Q((3, 2), U) = \underline{50}$$

$$Q((1, 2), D) = \underline{0}$$

$$Q((2, 2), R) = \underline{12.5}$$

- (c) (4 points) Consider a feature based representation of the Q-value function:

$$Q_f(s, a) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(a)$$

where  $f_1(s)$  is the x coordinate of the state,  $f_2(s)$  is the y coordinate of the state and  $f_3(a)$  is specified as  $f_3(U) = 1, f_3(D) = 2, f_3(R) = 3, f_3(L) = 4$

- (i) (3 points) Using the episode 1 of part (b) with zero initial  $w_i$ 's, calculate:

$$w_1 = \underline{-100}$$

$$w_2 = \underline{-100}$$

$$w_3 = \underline{-100}$$

- (ii) (1 point) Assuming that the weight vector is equal to (1,1,1), what would be the action taken using the above Q-function i state (2,2)?

Calculate the Q-values for all actions at state (2, 2). *Left* has the highest value

- (d) (5 points) The environment changes and the actions are no longer deterministic. You want to calculate the new transition function. The agent goes through the following episodes (reward not shown since it is not a part of this problem).

Episode 1	Episode 2	Episode 3	Episode 4	Episode 5
(1,3), D, (1,3)	(1,3), U, (1,3)	(1,3), D, (1,2)	(1,2), R, (1,3)	(1,2), R, (2,2)
(1,3), D, (1,2)	(1,3), U, (2,3)	(1,2), R, (2,2)	(1,3), D, (1,2)	(2,2), D, (1,2)
(1,2), R, (2,2)		(2,2), R, (3,2)	(1,2), R, (2,2)	(1,2), R, (2,2)
(2,2), D, (3,2)		(3,2), U, (3,2)	(2,2), D, (2,1)	(2,2), D, (2,1)
(3,2), U, (3,3)		(3,2), D, (3,1)		

You do not think that the agent has seen enough number of episodes. To avoid overfitting your transition model, you decide to apply extended Laplace smoothing with  $k = 2$ . Fill in the values of the following conditional probabilities. You can use the free space or back of this page for calculations

$s$	$a$	$s'$	$P_1(s' s, a)$	$P_2(s' s, a)$
(1,3)	U	(2,3)	$\frac{1+2}{2+2.9} = 0.150$	$\frac{1+2}{2+2.2} = 0.500$
(1,3)	D	(1,2)	$\frac{3+2}{4+2.9} = 0.227$	$\frac{3+2}{4+2.2} = 0.625$
(1,2)	R	(2,2)	$\frac{5+2}{6+2.9} = 0.292$	$\frac{5+2}{6+2.3} = 0.583$
(2,2)	D	(2,1)	$\frac{2+2}{4+2.9} = 0.182$	$\frac{2+2}{4+2.4} = 0.333$
(3,2)	D	(3,1)	$\frac{1+2}{1+2.9} = 0.158$	$\frac{1+2}{1+2.3} = 0.429$

I did not specify anything about restricting potential next states. However, it is possible to assume that the agent can only transition to its 4-neighbors. I will accept both solutions (potential to transition to all states and only 4-states). I will also accept the cases without/with self-transitions.