Assignment 4

Search, CSPs and Logic
Bayesian Networks, Probabilistic Inference

Deadline: December 1, 11:59pm.
Perfect score: 100 points.
Available points: 130.

Assignment Instructions:

Teams: Assignments should be completed by pairs of students. No additional credit will be given for students working individually. You are strongly encouraged to form a team of two students. Please inform the TAs as soon as possible about the members of your team so they can update the scoring spreadsheet and no later than October 17. In order to do so, please use the sign-up form provided by the TAs. You can find the TAs' contact information under the course's website: `http://www.pracsyslab.org/cs440`). Failure to inform the TAs on time will result in a lower grade in the assignment.

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For programming questions, you need to also submit a compressed file via Sakai, which contains your results and/or code as requested by the assignment. For your reports, do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. *Each team of students should submit only a single copy of their solutions and indicate all team members on their submission*. Failure to follow these rules will result in a lower grade in the assignment.

Program Demonstrations: There will be no program demonstrations for this assignment.

Late Submissions: No late submissions are allowed. You will be awarded 0 points for late assignments!

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (using LaTeX, in which case you should also submit electronically your source code). Resources on how to use LaTeX are available on the course's website (`http://www.pracsyslab.org/cs440`). There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise in your description and thorough in your evaluation. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common code, notes or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university (the standards are available through the course's website: `http://www.pracsyslab.org/cs440`). Failure to follow these rules may result in failure in the course.

Problem 1: [15 points] Two players, the MAX and the MIN are playing a game where there are only two possible actions, “left” and “right”. The search tree for the game is shown in Figure 1. The leaf/terminals nodes, which are all at depth 4, contain the evaluation function at the corresponding state of the game. The MAX player is trying to maximize this evaluation function, while the MIN player is trying to minimize this evaluation function.

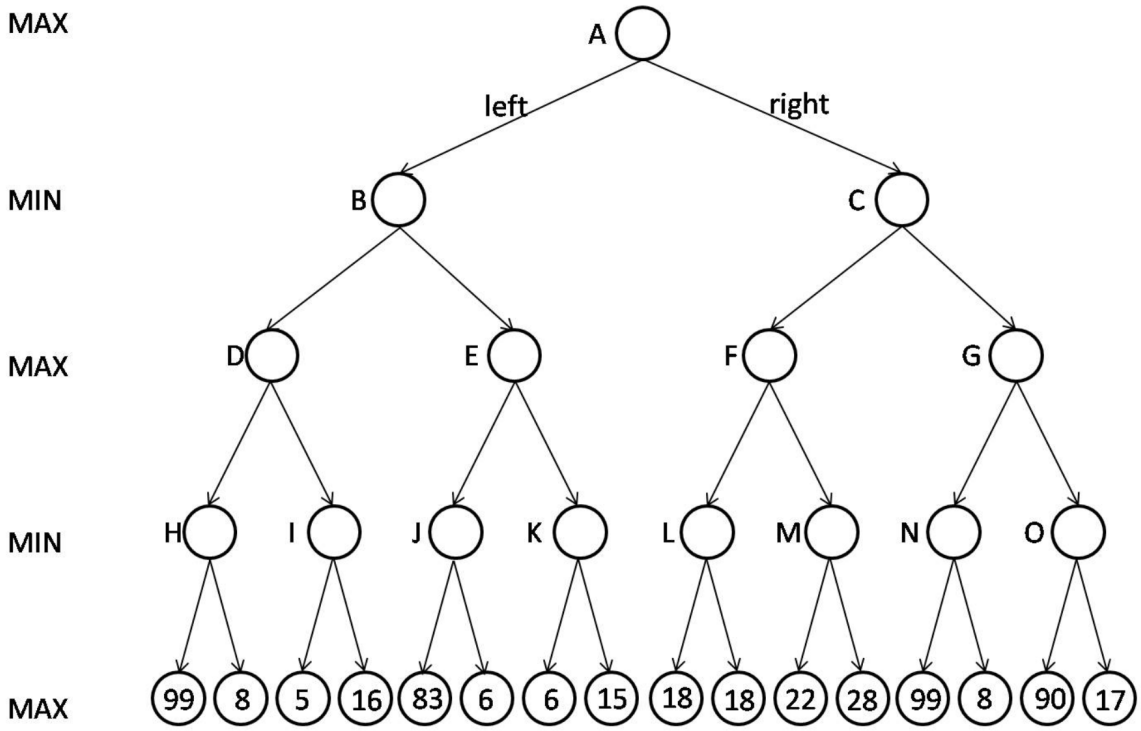


Figure 1: The adversarial search tree.

a. Consider the operation of the Minimax algorithm over the above search tree and indicate the value of each intermediate node.

b. Consider the operation of the Minimax algorithm with alpha-beta pruning over the above search tree. Indicate which nodes will not be consider by this algorithm (you can reuse the figure and mark them visibly or mention which intermediate and terminal nodes are not visited in text). Furthermore, indicate the alpha and beta values on each intermediate node that is visited.

c. What action will the MAX player choose at the root state according to the exhaustive Minimax algorithm? What action will the MAX player choose at the root state according to the Minimax algorithm employing alpha-beta pruning? In general, is the best move computed by the two versions of the algorithm guaranteed to be the same or not?

d. Consider a version of the algorithm that heuristically evaluates the quality of nodes at depth 2 so as to guide the ordering of nodes considered during the alpha-beta pruning.

   In particular, assume that the heuristic value at the depth 2 nodes are: $h(D) = 9$, $h(E) = 7$, $h(F) = 24$ and $h(G) = 16$, and Minimax backs-up these values at nodes B and C as well.

   Now, the MAX player executes the alpha-beta pruning algorithm down to depth 4. This time, however, it uses the heuristic values at nodes B through G to reorder the nodes of the depth-4 game tree at depths 1 and 2. The objective is to maximize the number of nodes that will not be examined by the alpha-beta pruning process.

   How will the alpha-beta pruning algorithm reorder the nodes, given the information generated up to depth 2? Show the corresponding new tree considered by the alpha-beta pruning

algorithm using the labels of the nodes from the above figure. In addition, inside each leaf node give the value of the evaluation function.

How many nodes will not be examined by the alpha-beta pruning algorithm in this case?

e. Consider now a variation of the game, where the opponent is known to play randomly, i.e., instead of trying to minimize the evaluation function, it selects with 50% the left action and with 50% the right action. Provide the value of each node on the original tree in this case. What will be the choice of the MAX player in this case at the root state?

Can you apply alpha-beta pruning in this case? If yes, show how. If not, explain why.

Problem 2: [10 points] Consider the game Sudoku, where we try to fill a 9×9 grid of squares with numbers subject to some constraints: every row must contain all of the digits $1, \ldots, 9$, every column must contain all of the digits $1, \ldots, 9$, and each of the 9 different 3×3 boxes must also contain all of the digits $1, \ldots, 9$. In addition, some of the boxes are filled with numbers already, indicating that the solution to the problem must contain those assignments. Here is a sample board:
Each game is guaranteed to have a single solution. That is, there is only one assignment to the empty squares which satisfies all the constraints. For the purposes of this homework, let's use $n_{i,j}$ to refer to the number in row i, column j of the grid. Also, assume that M of the numbers have been specified in the starting problem, where $M = 29$ for the problem shown above.

	1		4	2				5
		2		7	1		3	9
							4	
2		7	1					6
				4				
6					7	4		3
	7							
1	2		7	3		5		
3				8	2		7	

Figure 2: Sample Sudoku board

a. This is an instance of a Constraint Satisfaction Problem. What is the set of variables, and what is the domain of possible values for each? How do the constrains look like?

b. One way to approach the problem, is through an incremental formulation approach and apply backtracking search. Formalize this problem using an incremental formulation. What are the start state, successor function, goal test, and path cost function?

Which heuristic for backtracking search would you expect to work better for this problem, the degree heuristic, or the minimum remaining values heuristic and why?

What is the branching factor, solution depth, and maximum depth of the search space? What is the size of the state space?

c. What, is the difference between "easy" and "hard" Sudoku problems? [Hint: There are heuristics which for easy problems will allow to quickly walk right to the solution with almost no backtracking.]

d. Another technique that might work well in solving the Sudoku game is local search. Please design a local search algorithm that is likely to solve Sudoku quickly, and write it in pseudocode. You may want to look at the WalkSAT algorithm for inspiration. Do you think it will work better or worse than the best incremental search algorithm on easy problems? On hard problems? Why?

Problem 3: [10 points] Consider the following sequence of statements, which relate to Batman's perception of Superman as a potential threat to humanity and his decision to fight against him.

For Superman to be defeated, it has to be that he is facing an opponent alone and his opponent is carrying Kryptonite. Acquiring Kryptonite, however, means that Batman has to coordinate with Lex Luthor and acquire it from him. If, however, Batman coordinates with Lex Luthor, this upsets Wonder Woman, who will intervene and fight on the side of Superman.

a. Convert the above statements into a knowledge base using the symbols of propositional logic.

b. Transform your knowledge base into 3-CNF.

c. Using your knowledge base, prove that Batman cannot defeat Superman *through an application of the resolution inference rule* (this is the required methodology for the proof).

Problem 4: [10 points] Disjunctive logical sentences are of the form: $(l_1 \vee l_2 \vee \cdots l_n)$, where l_i is a literal or clauses, and sentences that have the form $(\alpha \Rightarrow \beta)$ (i.e., implication sentences).

a) Can you show the logical equivalence of $(\neg P_1 \vee \cdots \vee \neg P_m \vee Q)$ and $(P_1 \wedge \cdots \wedge P_m) \Rightarrow Q$?

b) Show that regardless of the number of positive literals in a clause, such expressions can be written in the form $(P_1 \wedge \cdots \wedge P_m) \Rightarrow (Q_1 \vee \cdots \vee Q_n)$, where the Ps and Qs are proposition symbols.

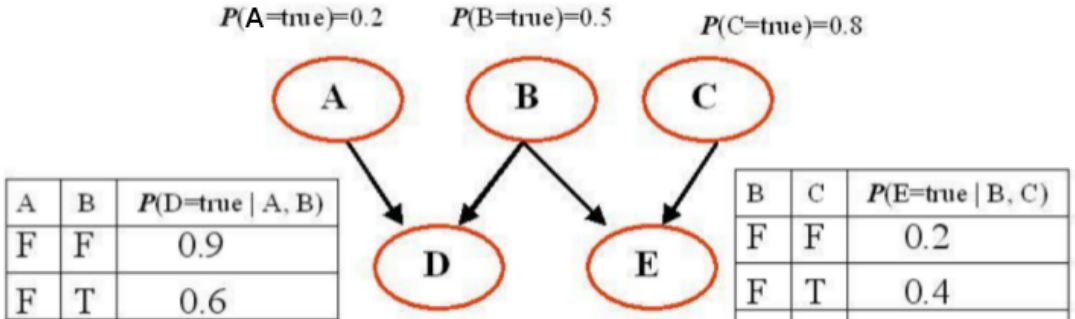
c) Write down the full resolution rule using implications. As a reminder the full resolution rule looks as follows:

- Given clause $(l_1 \vee \ldots \vee l_k)$
- and clause $(m_1 \vee \ldots \vee m_n)$
- we can infer that:

$$(l_1 \vee \ldots l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

where l_i and m_j are complementary literals.

Question 5: [15 points] Consider the following Bayesian network, where variables A through E are all Boolean valued:



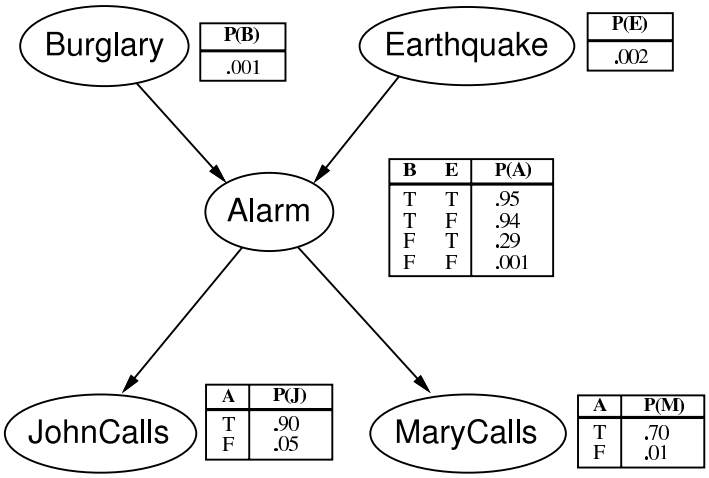
A	B	P(D=true | A, B)
F	F	0.9
F	T	0.6
T	F	0.5
T	T	0.1

B	C	P(E=true | B, C)
F	F	0.2
F	T	0.4
T	F	0.8
T	T	0.3

a) What is the probability that all five of these Boolean variables are simultaneously true?
[Hint: You have to compute the joint probability distribution (JPD). The structure of the Bayesian network suggests how the JPD is decomposed to the conditional probabilities available.]

b) What is the probability that all five of these Boolean variables are simultaneously false?
[Hint: Answer similarly to above.]

c) What is the probability that A is false given that the four other variables are all known to be true?
[Hint: Try to use the definition of the conditional probability and the structure of the network. For probabilities that can not be computed directly from the network, remember the following normalization trick: if $P(x) = \alpha \cdot f(x)$ and $P(\neg x) = \alpha \cdot f(\neg x)$, then you can compute the normalization factor as $\alpha = \frac{1.0}{f(x)+f(\neg x)}$, since $P(x) + P(\neg x) = 1.0$.]



Question 6: [10 points] For this problem, check the Variable Elimination algorithm in your book. Also consider the Bayesian network from the "burglary" example.

a) Apply variable elimination to the query:

$$P(Burglary|JohnsCalls = true, MaryCalls = true)$$

and show in detail the calculations that take place. Use your book to confirm that your answer is correct.

b) Count the number of arithmetic operations performed (additions, multiplications, divisions), and compare it against the number of operations performed by the tree enumeration algorithm.

c) Suppose a Bayesian network has the from of a *chain*: a sequence of Boolean variables $X_1, \ldots X_n$ where $Parents(X_i) = \{X_{i-1}\}$ for $i = 2, \ldots, n$. What is the complexity of computing $P(X_1|X_n = true)$ using enumeration? What is the complexity with variable elimination?

Question 7: [15 points] One method for approximate inference in Bayesian Networks is the Markov Chain Monte Carlo (MCMC) approach. This method depends on the important property that a variable in a Bayesian network is independent from any other variable in the network given its Markov Blanket.

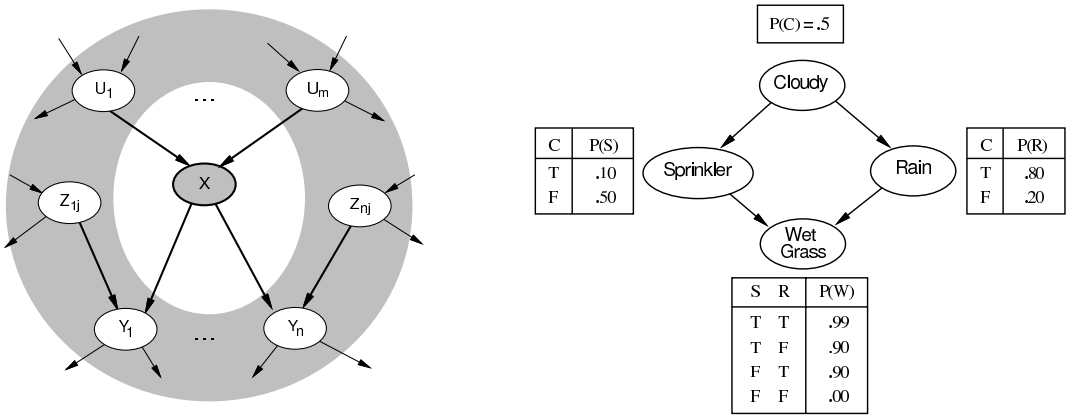


Figure 3: (left) The Markov Blanket of variable X (right) The Rain/Sprinkler network.

a) Prove that

$$P(X|MB(X)) = \alpha\ P(X|U_1, \ldots, U_m) \prod_{Y_i} P(Y_i|Z_{i1} \ldots)$$

where $MB(X)$ is the Markov Blanket of variable X.

b) Consider the query

$$P(Rain|Sprinkler = true, WetGrass = true)$$

in the Rain/Sprinkler network and how MCMC would answer it. How many possible states are there for the approach to consider given the network and the available evidence variables?

c) Calculate the transition matrix Q that stores the probabilities $P(y \rightarrow y')$ for all the states y, y'. If the Markov Chain has n states, then the transition matrix has size $n \times n$ and you should compute n^2 probabilities.
[Hint: Entries on the diagonal of the matrix correspond to self-loops, i.e., remaining in the same state. Such transitions can occur by sampling either variable. Entries where one variable is different between y and y', must sample that one variable. Entries where two or more variables change cannot occur, since in MCMC only one variable is allowed to change at each transition.]

Question 8: [15 points] Assume you are interested in buying a used vehicle C_1. You are also considering of taking it to a qualified mechanic and then decide whether to buy it or not. The cost of taking it to the mechanic is \$100. C_1 can be in good shape (quality q^+) or bad one (quality q^-). The mechanic might help to indicate what shape the vehicle is in. C_1 costs \$3,000 to buy and its market value is \$4,000 if in good shape; if not, \$1,400 in repairs will be needed to make it in good shape. Your estimate is that C_1 has a 70% chance of being in good shape. Assume that the utility function depends linearly on the vehicle's monetary value.

a. Calculate the expected net gain from buying C_1, given no test.

b. We also have the following information about whether the vehicle will pass the mechanic's test:

$$P(pass(c_1)|q^+(c_1)) = 0.8$$

$$P(pass(c_1)|q^-(c_1)) = 0.35$$

Use Bayes' theorem to calculate the probability that the car will pass/fail the test and hence the probability that it is in good/ bad shape given what the mechanic will tell you.
[Hint: Compute the four probabilities: $P(q^+|Pass)$, $P(q^-|Pass)$, $P(q^+|\neg Pass)$, $P(q^-|\neg Pass)$]

c. What is the best decision given either a pass or a fail? What is the expected utility in each case?
[Hint: Use the probabilities from the previous question.]

d. What is the value of optimal information for the mechanic's test? Will you take C_1 to the mechanic or not?
[Hint: You can easily answer this based on the answers from questions a) and c).]

Question 9: [15 points]

H	H	T
N	N	N
N	B	H

Consider that you are placed in an unknown cell of the above 3×3 map, i.e., initially there is a probability $P(x_0) = \frac{1}{8}$ to be located in any of the cells. We denote as $(1, 2)$ the coordinates of the top middle cell, and as $(2, 3)$ the coordinates of the rightmost middle cell, where:

- N is a normal cell;
- H is a highway cell;
- T is a hard to traverse cell;
- B is a blocked cell.

You are able to move inside this world by executing actions $\alpha = \{Up, Left, Down, Right\}$. You are also equipped with a sensor that informs you about the terrain type that you are occupying after every time you are trying to move inside this world. Your objective is to figure out your location inside this world given that you had no idea initially where you were located.

Lets denote as $P(x_i|x_{i-1}, \alpha)$ the transition model for moving from cell x_{i-1} at step $(i-1)$ to cell x_i at step i given an action α. If given your location x_{i-1} and action α, you would hit the boundary of this grid world or a blocked cell, then you stay in the same cell, i.e., $x_i = x_{i-1}$. The model is probabilistic because our motions are not executed perfectly inside this grid world. In particular, 90% of the time the action is executed correctly but 10% the action fails and the agent stays on the same cell. So, for instance if you execute action $\{Up\}$ from cell $(2, 2)$, with 90% probability you move to cell $(1, 2)$ and with 10% probability you stay at cell $(2, 2)$. If you are at cell $(3, 1)$ and move $Right$, then with 100% probability you stay at the same cell (because cell $(3, 2)$ is a blocked cell).

Furthermore, denote as $P(e_i|x_i)$ the observation model for detecting terrain type, where e_i is the observed terrain type for cell x_i. The terrain sensor is correct 90% of the time but with probability 5% it can return either of the other two terrain types (the sensor never returns "blocked cell" as you never occupy one). For instance, if you are located at cell $(2, 2)$, your terrain sensor returns with probability 90% the value N for "normal cell", with 5% probability it returns the value H for "highway cell" and with 5% probability it returns the value T for "hard to traverse cell".

*S*tep A: You are asked to compute the probability of where you are inside this grid world *after* you execute actions $\{Right, Right, Down, Down\}$ and the corresponding sensing readings are $\{N, N, H, H\}$ (you sense after you move). You are encouraged to do this by hand and through a program so as to have the capability to debug your solution. Submit in your report the probabilities of where you are inside the world after each action/sensor reading pair assuming that in the beginning you have no knowledge of where you are located. This means that you need to provide four 3x3 maps with sets of eight probabilities indicating where you are inside this grid world after each action/sensing pair.

*S*tep B: The next objective is to scale up the size of filtering problems like the above one that you can solve. For this question, you will use large maps (e.g., 100 by 100), where you randomly assign terrain types to each cell out of the four possible types (50% normal cells, 20% highway cells, 20% hard to traverse and 10% blocked cells). First, generate "ground truth" data, i.e., generate sequences of actions and sensor readings to test your algorithm.

For this purpose, first randomly select a non-blocked cell as the initial location of your agent in the world x_o. Then, randomly generate a sequence of 100 actions, i.e., randomly select actions from the set $\alpha = \{Up, Left, Down, Right\}$ and generate a string of length 100. For each action starting from the initial location x_0 apply:

- the transition model (i.e., 90% follow the action - when you collide stay in place, 10% stay in place), in order to get the next ground truth state x_{i+1};
- the observation model (i.e., 90% the correct terrain type and 5% one of the other two types), in order to get the "ground truth" sensor reading e_{i+1}.

Once you have generated the 100 actions, the 100 ground truth states and the 100 observations, store them in a file. Generate 10 such files per map and for 10 different maps of the world (i.e., 100 total ground truth files), as different experiments inside the large maps. The format for the file can be as follows:
$x_0 y_0$:coordinates of initial point
$x_i y_i$:100 coordinates of the consecutive points that the agent actually goes through separated by a new line
α_i :100 characters indicating the type of action executed $\{U, L, D, R\}$ separated by a new line
e_i :100 characters indicating the sensor reading $\{N, H, T\}$ collected by the agent as it moves

In your report, provide examples of ground truth paths and the corresponding action and sensor readings generated.

*S*tep C: Demonstrate the capability of estimating the position of the agent inside the world given only the actions and observations indicated in these files. In particular, your program should be able to load a "ground truth" file and a map. Then, after each action/sensor reading pair, it should be able to compute the probability that the agent is in each cell of the map by applying the filtering algorithm. Visualize the different probabilities on your map (e.g., think of a heatmap) or provide a capability to indicate the probability on any cell of the map. Visualize the ground truth location of the agent inside the world at the corresponding step. Your visualization should be updated with each new action/sensor reading pair until you consume all 100 readings.

Attach example heat maps in your report after 10, 50 and 100 iterations. Indicate the ground truth path up to this point in each case.

For each of the 100 experiments, compute the error (as distance inside the grid world) between the true location of the agent and the maximum likelihood estimation (i.e., the cell with the highest probability according to the filtering algorithm) as the number of readings increases. For the computation of the maximum likelihood estimation, ties can be broken randomly. Generate a plot of the average error over all 100 experiments as the number of readings increases. For this plot, you can ignore the first 5 iterations as many cells will have the same probability in the beginning.

For each of the 100 experiments, keep track of the probability that the cell where the agent is actually located (which changes over time) is assigned by the filtering algorithm. Generate a plot of the average probability of the ground truth cell over 100 experiments as the number of readings increases. Here you can start with the uniform probability assigned to the cell in the beginning of this process.

You *may* face computational challenges in the straightforward implementation of the above algorithms given the sizes of the map. If you find this to be the case, you are allowed to provide results for smaller maps (e.g., try first 20x20 maps, then 50x50).

Question 10: [15 points] Consider the specification of a Markov Decision Process according to the following figure. Code your own implementation of Value Iteration and compute the optimal policy as well as the optimum utilities for this challenge.

Indicate the original utilities you used in order to start the process. Provide at least 5 intermediate results (in terms of optimum utilities and policies) depending on the number of iterations needed for convergence as well as the final results. Describe your implementation and your convergence criterion. Report computation time and number of iterations.

s	$r(s)$
s_1	0
s_2	0
s_3	1
s_4	0

s_i	a	s_j	$T(s_i, a, s_j)$
s_1	a_1	s_1	0.2
s_1	a_1	s_2	0.8
s_1	a_2	s_1	0.2
s_1	a_2	s_4	0.8
s_2	a_2	s_2	0.2
s_2	a_2	s_3	0.8
s_2	a_3	s_2	0.2
s_2	a_3	s_1	0.8
s_3	a_4	s_2	1
s_3	a_3	s_4	1
s_4	a_1	s_4	0.1
s_4	a_1	s_3	0.9
s_4	a_4	s_4	0.2
s_4	a_4	s_1	0.8

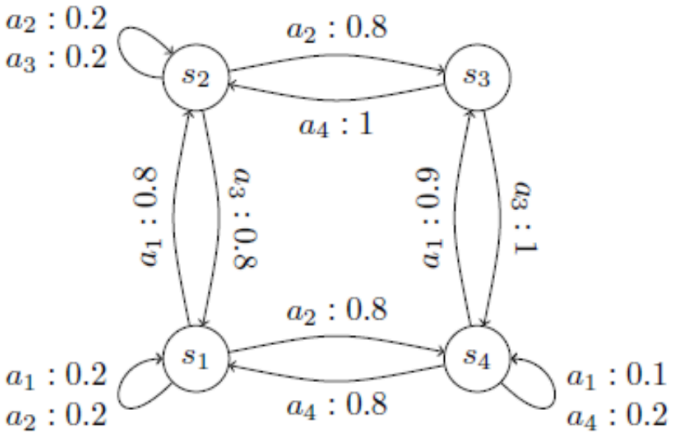


Figure 4: The specification of the MDP problem.