Artificial Intelligence: Search, CSPs and Logic Bayesian Networks, Probabilistic Inference

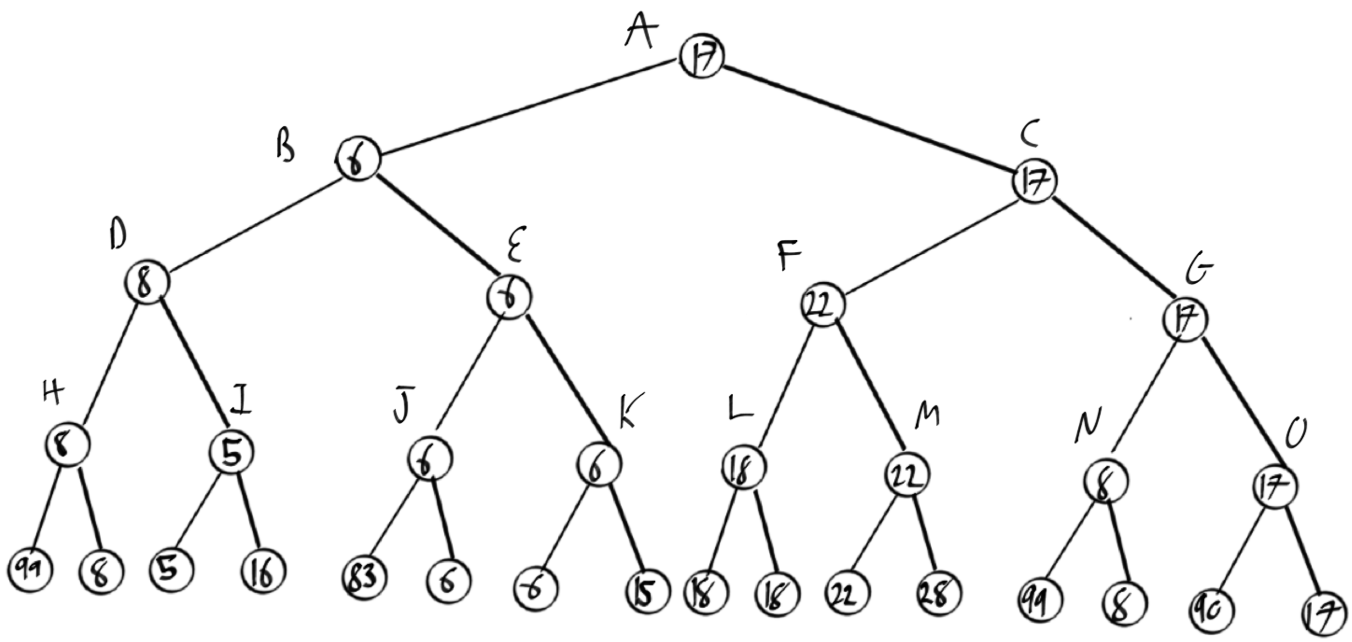
Nathan Morgenstern, Seo Bo Shim

December 10, 2017

Contents

Question 1

part a.



H: 8, I: 5, J: 6, K: 6, L: 18, M: 22, N: 8, O: 17

D: 8, E: 6, F: 22, G: 17

B: 6, C: 17

A: 17

Figure 1: Question 1: Part a

part b.

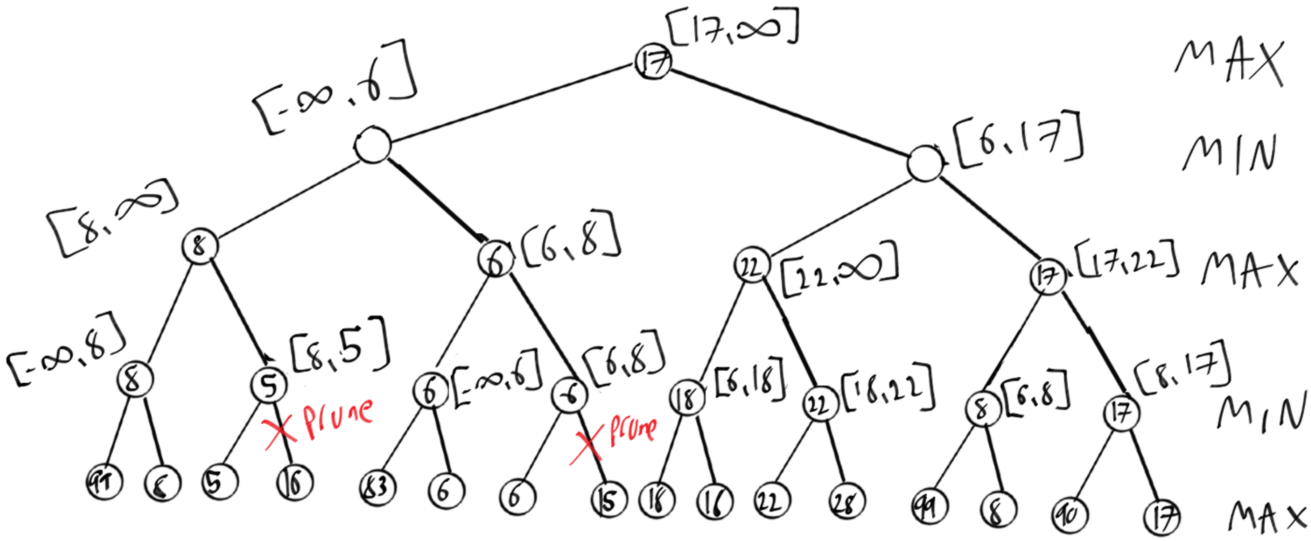


Figure 2: Question 1: Part b

part c.

NOT DONE

part d.

NOT DONE

part e.

NOT DONE

Question 2

part a.

In this section we define the set of variables, domain, and the constraints.

The set of variables is the combination of the row and column of the grid.

X = [1,1], [1,2], ... , [i,j-1], [i, j]

The domain is the possible values that each variable can hold: 1 to 9.

D = 1, 2, 3, 4, 5, 6, 7, 8, 9

The constraint is that each row, column, and neighboring box must hold unique values in each of the variables. The following rule ensures each value in a row is unique.

i, j = fixed random variable

I, J = random variable

C1 = (X(i,1), X(i,2), X(i,3), X(i,4), X(i,5), X(i,6), X(i,7), X(i,8), X(i,9)),

X(i,1) != X(i,2) != X(i,3) != X(i,4) != X(i,5) != X(i,6) != X(i,7) != X(i,8) != X(i,9)

This rule ensures each value in a column is unique.

C2 = (X(1,j), X(2,j), X(3,j), X(4,j), X(5,j), X(6,j), X(7,j), X(8,j), X(9,j)),

X(1,j) != X(2,j) != X(3,j) != X(4,j) != X(5,j) != X(6,j) != X(7,j) != X(8,j) != X(9,j)

This rule ensures each value in a 3x3 box region are unique. a = 2, 5, 8

C3 = (X(a-1,a-1), X(a-1,a), X(a-1,a+1), X(a,a-1), X(a,a), X(a,a+1), X(a+1,a-1), X(a+1,a), X(a+1,a+1)),

X(a-1,a-1) != X(a-1,a) != X(a-1,a+1) != X(a,a-1) != X(a,a) != X(a,a+1) != X(a+1,a-1) != X(a+1,a) !=

X(a+1,a+1)

part b.

We define the start state, successor function, goal test, path cost, branching factor, solution depth, maximum depth, size of the state space, and the ideal value heuristic to use. **Start state:** initial arrangement of values 1-9 on the Sudoku board in random locations and such that the constraints are not broken. No additional values on the board

Successor function: to return the set of all successive states, try a value from 1-9 at each coordinate, and exclude the Sudoku configurations that do not meet the constraints, e.g. two 9's in the same row and 3x3 region.

Goal test: all locations on the board has a value between 1 - 9 and meet the constraint. The 3x3 region, row, or column all have unique values.

Path cost:

Take for example an almost filled box in Sudoku, represented in matrix form:

$$\begin{Bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{Bmatrix}$$

The path cost to a state where 9 goes in the corresponding slot should be very low. We also note that this slot can only have a 9 as a possible value. Whereas in this case:

$$\left\{\begin{matrix} & 3 \\ 4 & 6 \\ & \end{matrix}\right\}$$

The path cost to enter a value in these slots should be very high as this action does not lead to any definite resolutions. Each of the slots can possibly be values 1, 2, 5, 7, 8, or 9. We seek paths that to states that are more constrained, or solved.

We can consider the number of possible values that can be placed on an empty slot E_p.

The path cost can be defined as the change in the number of possible values that can be placed in empty slots, or ΔE_p. If an action results in a larger difference in E_p, then the path cost is high. If an action results in a small change in E_p, then these paths are ideal. This is because if there is a small change, then this means that the action taken solves a currently existing constraint.

Minimum Remaining Value

The minimum remaining value heuristic is ideal because it finds the most constrained variable and removes it from the search tree. This prunes the search tree and reduces the possible size of the solution set.

If the degree heuristic is chosen, it will work to affect the constraints of the other values. We do want to eventually constrain the other variables more, however this is not a measure of success. Arbitrarily choosing a random cell that affects the maximum number of other cells would be favorable under the degree heuristic.

To be reliable, we should choose the heuristic that removes the most constrained variable, irrespective of it's effect on other cells.

Branching factor: Once the first value is written to a cell, there are: $9 * 51 = 459$ different options to choose from.

Eventually as the board gets filled and there are only 2 cells remaining there will be $9 * 2 = 18$ options remaining because there are two cells left to be filled and each have 9 options.

Solution depth: The solution will always be found at a depth of 52. This is when the board will be full of values.

Maximum depth: 52. When 52 values are written into each cell, then the board will be completely filled.

Size of state space:

Each possible value for each cell can be repeated for each cell on the board. 9 possible values for each cell

81 - M = 52 total cells in board

State space $=9^{52} \approx 10^{49}$;

part c.

Easy problems can be solved by choosing actions in the coordinates that are most constrained, and continue choosing actions this way until the board is filled. The clues are given in such a way that every action the user takes can be deterministic and the user does not have to make any choices.

Difficult problems most likely will require backtracking, because the solution can require actions that aren't intuitive and may require the user to make approximations. The puzzle may reach an equilibrium point where the next action is not obvious, as multiple actions can be taken and still be valid until a much later state.

part d.

function Search():
boards = PriorityQueue of boards;
i[num cells] = 0;
j = 0;
if AdvanceBoard(board, cell, 0) = failure // if random value changes required
j = 0;
while boards is not empty
get board ¡= boards
if AdvanceState($board, conflict_index[j], probability$)= success then return success
end while
else return success
end if;

function AdvanceBoard(board, cell, p):
while available successors greater than 0
with probability p, select random value for board[cell]
with probability 1-p, select best value based on path costs
num constraints unmet ¡- evaluate board
if (board = full) return success
if constraints unmet:
conflictIndex[j] = store board's index of conflict
boards[j] = store board
increment j
board[cell] = 0 //reset the board
else cell = empty cell on board
end if
end while

This algorithm will advance the states of the Sudoku board as normal. If it reaches a state where a conflict exists and the constraints are not met, this state is saved in a priority queue data structure. Once the first iteration passes through all the actions and gets stuck (no backtracking) as a solution could not be found, then we go back to the boards with the unmet constraints. The boards should be ordered in a priority queue, and the boards with the least amount of empty cells should be chosen first. We randomly change the value of the board at these conflict cells and continue to advance the state of the board if there are any successors. We iterate over the saved boards until we iterate through all of them and a solution is found.

The incremental formulation will likely do better on easier puzzles. This local search algorithm will likely do better on more difficult puzzles as it will randomly switch these variables. It won't have to waste time with back-propagation and will randomly switch a variable to proceed to the optimal state.

We argue that randomly switching a value in a cell will have better results than back-propagating as this will take time and it may have to search the whole state space to find the correct.

Question 3

part a.

NOT DON3

part b.

NOT DONE

part c.

NOT DONE

Question 4

part a.

$$(\neg P_1 \vee ... \vee \neg P_m \vee Q)$$

Using DeMorgan's theorem, this term is equivalent to:

$$\neg(P_1 \wedge ... \wedge P_m \wedge \neg Q)$$

Using the Associative property

$$\neg((P_1 \wedge ... \wedge P_m) \wedge \neg Q)$$

Using the conditional equivalence: $\neg(P \rightarrow Q) = (P \wedge \neg Q)$

$$\neg(\neg(P_1 \wedge ... \wedge P_m \rightarrow Q))$$

Finally,

$$(P_1 \wedge ... \wedge P_m \rightarrow Q)$$

part b.

We can repeat the above proof by replacing Q with an expression $(Q_1 \vee ... \vee Q_n)$. A literal can be replaced with an expression.

$$(\neg P_1 \vee ... \vee \neg P_m \vee \neg(Q_1 \vee ... \vee Q_n))$$

Using DeMorgan's theorem, this term is equivalent to:

$$\neg(P_1 \wedge ... \wedge P_m \wedge \neg(Q_1 \vee ... \vee Q_n))$$

Using the Associative property

$$\neg((P_1 \wedge ... \wedge P_m) \wedge \neg(Q_1 \vee ... \vee Q_n))$$

Using the conditional equivalence: $\neg(P \rightarrow Q) = (P \wedge \neg Q)$

$$\neg(\neg((P_1 \wedge ... \wedge P_m) \rightarrow (Q_1 \vee ... \vee Q_n)))$$

Finally,

$$(P_1 \wedge ... \wedge P_m) \rightarrow (Q_1 \vee ... \vee Q_n)$$

part c.

To complete the full resolution and find the resolvent, start with the two clauses:

$$(l_1 \vee ... \vee l_k \vee m_1 \vee ... \vee m_n)$$

We can split this expression up using the associative property:

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

Using the Conditional equivalence property, similar to what we derived in part A, this term is equivalent to:

$$\neg(l_i \vee m_j) \rightarrow (l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k) \vee (m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)$$

Because l_i and m_j are complimentary: $(l_i \vee m_j) = True$

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

Therefore it is inferred that

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

Question 5

part a.

NOT DON3

part b.

NOT DONE

part c.

NOT DONE

1

Question 6

part a.

B = burglary

E = earthquake

A = alarm

J = John calls

M = Mary calls

$$P(B) = 0.001$$

$$P(E) = 0.002$$

$$P(B|J,M) = \alpha * P(B) * \sum_{e} P(e) * \sum_{a} P(a|B,e) * P(J|a) * P(M|a)$$

Compute the sum over A, which results in summing two terms as alarm is either true or false.

$$\sum_{a} P(a|B,e) * P(J|a) * P(M|a) = P(a|B,E) * P(j|a) * P(m|a) + P(\neg a|B,E) * P(j|\neg a) * P(m|\neg a)$$

$$P(b,e) = .95 * .9 * .7 + .05 * .05 * .01 = .598525$$

For the next summation we will also need the case where the earthquake event does not occur:

$$P(b, \neg e) = .94 * .9 * .7 + .06 * .05 * .01 = .592230$$

Then the rest of the calculations:

$$P(\neg b, e) = .29 * .9 * .7 + .71 * .05 * .01 = .183055$$

$$P(\neg b, \neg e) = .001 * .9 * .7 + .999 * .05 * .01 = .001130$$

Including these calculations into the matrix form:

$$f_6(B, E) = \begin{Bmatrix} P(b, e) & P(b, \neg e) \\ P(\neg b, e) & P(\neg b, \neg e) \end{Bmatrix}$$

$$f_6(B, E) = \begin{Bmatrix} .598525 & .592230 \\ .183055 & .001130 \end{Bmatrix}$$

Then sum over E, whether the event of an earthquake is true or false.

$$P(B|J, M) = \alpha * P(B) * \sum P(e) * f_6(B, e)$$

$$\sum_e P(e) * P(B, e) = P(e) * f_6(B, e) + P(\neg e) * f_6(B, \neg e)$$

$$\begin{Bmatrix} P(b, e) \\ P(\neg b, e) \end{Bmatrix} = .002 * \begin{Bmatrix} .598525 \\ .183055 \end{Bmatrix} + 0.998 * \begin{Bmatrix} .592230 \\ .001130 \end{Bmatrix} = \begin{Bmatrix} .590466 \\ .001494 \end{Bmatrix}$$

$$f_7(B) = \begin{Bmatrix} .590466 \\ \\ .001494 \end{Bmatrix}$$

$$f_7(b) = .590466$$

$$f_7(\neg b) = .001494$$

Finally

$$P(B|J, M) = \alpha * P(B) * f_7(B))$$

$$P(b|J, M) = \alpha * .001 * .590466$$

$$P(\neg b|J, M) = \alpha * .999 * .001494$$

We sum up the two probabilities to find alpha

$$P(b|J, M) + P(\neg b|J, M) = 0.0020830$$

$$\alpha = 1/0.0020830 = 480.$$

$$\boldsymbol{P(b|J, M) = 480 * .001 * .590466 = 0.28347}$$

$$\boldsymbol{P(\neg b|J, M) = 480 * .999 * .001494 = 0.71653}$$

part b.

Operations in Variable Elimination

4*(4 mult 1 add) + 2*(2 mult 1 add) + 2*(2 mult):

+ 1 division and 1 addition for finding the α value:

7 additions

24 multiplications

1 division

32 total operations

Operations in tree enumeration algorithm

For each probability in the final term $P(B|j,m)$ we have three additions, 14 multiplications, and 1 division. The division is from the normalization factor.

There are 8 different cases in $P(B|j,m)$, so in total:

24 additions

112 multiplications

8 divisions

144 total operations

Variable elimination uses much less operations than the tree enumeration algorithm. It uses 22 % of the operations used by the tree enumeration algorithm.

part c.

where n is the number of boolean variables

Using variable elimination: The time and space complexity are dominated by the highest factor constructed. The largest factor is comprised of the number of a children a node in a Bayesian Network may have. If there are n boolean variables, then it is possible for n-1 boolean variables to all have the same parent node. So the largest factor is O(n-1), but this is in fact the worst case scenario. On average it will be less than n as the Bayesian Network will not always be in this configuration.

The space complexity is $0(2^{(n-1)})$ The time complexity is $O(2^{(n-1)})$

the worst case complexity is worse than the tree enumeration, however on average the time complexity is better. There is higher potential space complexity to save states so they do not need to be calculated again (reduce time complexity).

Using enumeration: The space complexity is $O(n)$ as the only requirement is to store each boolean variable The time complexity is $O(2^(n))$, as each combination of the boolean variable must be calculated.

Question 7

part a.

NOT DON3

part b.

NOT DONE

part c.

NOT DONE

Question 8

part a.

Cost of good quality car:

$$C(q^+(c_1)) = \$4000 - (3000) = \$1000. \tag{1}$$

Cost of bad quality car:

$$C(q^-(c_1)) = \$4000 - (3000 + 1400) = -\$400. \tag{2}$$

Assuming that repairs can be made without taking it to the mechanic - perhaps the $1400 is a separate cost than the $100 needed to check the quality of the car.

Probability of good quality car:

$$P(q^+(c_1)) = 0.7 \tag{3}$$

Probability of bad quality car:

$$P(q^-(c_1)) = 0.3 \tag{4}$$

Expected net gain:

$$E(c_1) = C(q^+(c_1)) * P(q^+(c_1)) + C(q^-(c_1)) * P(q^-(c_1))$$

$$\mathbf{E(c_1) = \$580} \tag{5}$$

part b.

$$P(Pass|q^+) = 0.8 \tag{6}$$

$$P(Pass|q^-) = 0.35 \tag{7}$$

Using 6 and 7:

$$P(\neg Pass|q^+) = 1 - P(Pass|q^+) = 0.2 \tag{8}$$

$$P(\neg Pass|q^-) = 1 - P(Pass|q^-) = 0.65 \tag{9}$$

Using 3, 4, 6, 7:

$$P(Pass) = P(Pass|q^+) * P(q^+) + P(Pass|q^-) * P(q^-) = 0.665$$

$$\mathbf{P(Pass) = 0.665} \tag{10}$$

Using 10:

$$P(\neg Pass) = 1 - P(Pass) = 0.335$$

$$\mathbf{P(\neg Pass) = 0.335} \tag{11}$$

Using 6, 3, and 10:

$$\mathbf{P(q^+|Pass) = \frac{P(Pass|q^+) * P(q^+)}{P(Pass)} = 0.842} \tag{12}$$

Using 7, 4, and 10:

$$\mathbf{P(q^-|Pass) = \frac{P(Pass|q^-) * P(q^-)}{P(Pass)} = 0.158} \tag{13}$$

Using 8, 3, and 11:

$$\mathbf{P(q^+|\neg Pass) = \frac{P(\neg Pass|q^+) * P(q^+)}{P(\neg Pass)} = 0.418} \tag{14}$$

Using 9, 4, and 11:

$$\mathbf{P(q^-|\neg Pass) = \frac{P(\neg Pass|q^-) * P(q^-)}{P(\neg Pass)} = 0.582} \tag{15}$$

part c.

Paying for the test with the mechanic, the new costs are:

$$C'(q^+(c_1)) = C(q^+(c_1)) - \$100 = \$900$$

$$C'(q^-(c_1)) = C(q^-(c_1)) - \$100 = -\$500$$

Given a pass:

$$E(c_1|Pass) = C'(q^+(c_1)) * P(q^+(c_1)|Pass) + C'(q^-(c_1)) * P(q^-(c_1)|Pass)$$

$$\mathbf{E(c_1|Pass) = \$678.8}$$

Given a failure:

$$E(c_1|\neg Pass) = C'(q^+(c_1)) * P(q^+(c_1)|\neg Pass) + C'(q^-(c_1)) * P(q^-(c_1)|\neg Pass)$$

$$\mathbf{E(c_1|\neg Pass) = \$85.2}$$

Regardless of a pass or a failure, the best decision is the sell the car as there will be a net gain.

part d.

Without the mechanic's test, the expected gain from selling the car will be $\mathbf{E(c_1) = \$580}$

With the test, the expected gain is $\mathbf{\$678.80}$. The value of the optimal information is the difference between the expected gain with the information and the expected gain without the information. The optimal information value is $**98.80**. I should take C1 to the mechanic.