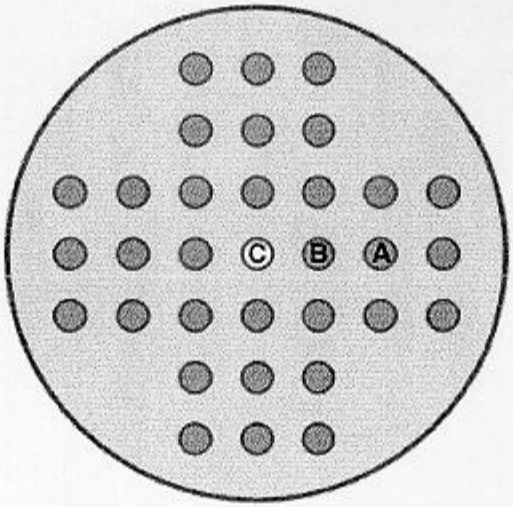
C1 Artificial Intelligence (25 points)
Planning



Consider the problem of playing *peg solitaire*, as shown in the figure above. Initially all the holes are filled with pegs except the center hole (labeled C). The goal is to remove all the pieces but one, which must be left in the center hole C. A piece is removed by hopping an adjacent piece over the piece in question into an empty hole. For example, piece A may hop over piece B into hole C, which removes piece B.

You should formulate peg solitaire as a partial-order planning problem using STRIPS operators.

(a) (6 points) Describe the START and FINISH steps for the problem (no need to list EVERY literal one by one, but say what literals are needed). You will need to choose a vocabulary. You may assume that the predicate $Line(u, v, w)$ (meaning that positions u, v, w are consecutive along a line) is available, and that the constant C refers to the center location.

(b) (10 points) Now write the STRIPS operator(s) for making a move. Be sure to correctly observe the syntactic restrictions on STRIPS operators.

(c) (9 points) Describe how a clobbering conflict can occur in the course of partial-order planning with this particular formulation, or explain why one cannot occur.

C2 Artificial Intelligence (25 Points)
Search

(a) (5 Points) Briefly explain IDA* search and discuss its advantages over other search methods.

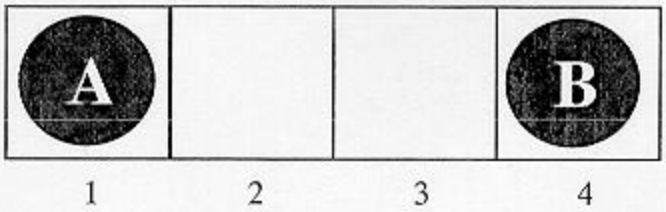
(b) (8 points) Explain why IDA* may not be able to find an optimal solution if there is insufficient memory. Your answer should discuss the difference between path length and path cost. Construct an example where a less than optimal solution is found because memory problems preclude searching some areas of the graph.

(c) (12 points) SMA* is a memory bounded search that abandons unpromising paths during a search when memory runs low. I.e., before it expands a node, it checks to see if memory is full. If it is, some unpromising nodes must be removed from the search graph (and search variables updated accordingly) before continuing.

It can easily be shown that IDA* returns an optimal solution whenever there is sufficient memory for the longest path with cost $\leq f^*$. Could the IDA* algorithm be modified (in a memory-bounded fashion similar to SMA*) to succeed in finding the optimal solution even when there is only enough memory to search the *shortest* optimal solution path?

C3 Artificial Intelligence (25 Points)
Games

Consider a two-player game featuring a board with four locations, numbered 1 through 4 and arranged in a line. Each player has a single token. Player A starts with his/her token on space 2, and player B starts with his/her token on space 4. Player A moves first.



The two players take turns moving, and each player must move his/her token to an open adjacent space *in either direction*. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space, if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game is +1; if player B reaches space 1 first, then the value of the game is -1.

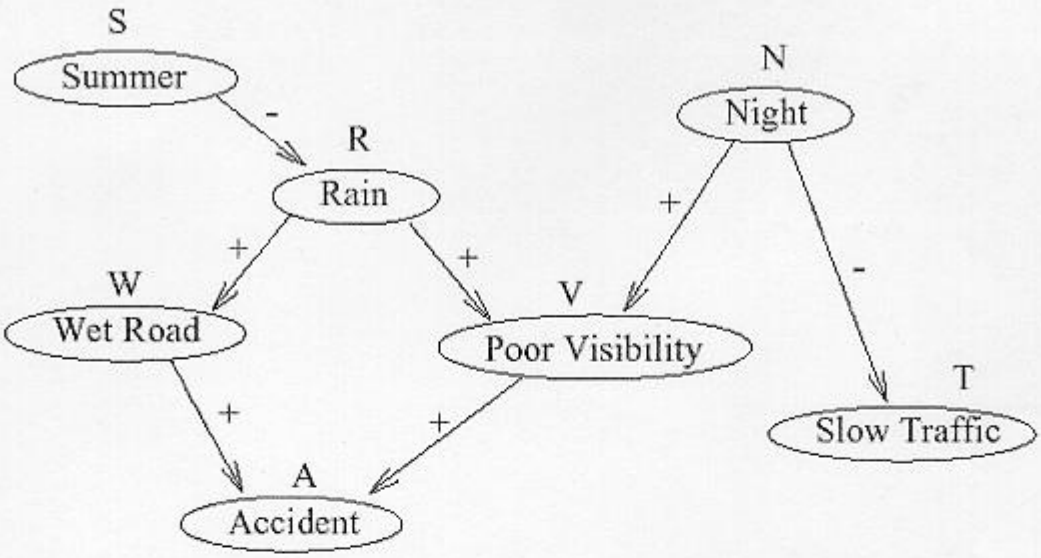
(a) (5 points) Draw the complete game tree using the following conventions:

- Write each state as (Sa, Sb) where Sa and Sb denote the token locations.
- Put the terminal states in square boxes, and annotate each with its game value in a circle.
- Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.

(b) (7 points) Mark each node with its backed-up minimax value (also in a circle). Explain in words how you handled the "?" values, and why your solution is consistent with the minimax algorithm.

(c) (13 points) Explain why the standard minimax algorithm would fail on this game tree because of the loops. Briefly sketch how you might handle loops in games, drawing on your answer to (b). Discuss whether or not your modified algorithm will give optimal decisions for all games with loops. Consider situations where your solution may have difficulty.

C4 Artificial Intelligence (25 points)
Belief Networks



Consider the above Bayesian network. The domain of each variable is $\{true, false\}$. For convenience, each variable has a one letter abbreviation. Instead of showing the conditional probability tables, we have indicated the qualitative nature of the causal influences in the network. A plus indicates a positive influence, and a minus indicates a negative influence. If there is a positive influence from X to Y, then $P(y|x) > P(y|\neg x)$, and visa versa for a negative influence. (Note: $P(y|\neg x)$ is shorthand for $P(Y = true|X = false)$, etc.) This relationship holds for *all values* of the other parents of Y. For example, in the given network we can infer, amongst other things:

$$\begin{aligned} P(t|n) &< P(t|\neg n) \\ P(v|n,r) &> P(v|n,\neg r) \\ P(v|\neg n,r) &> P(v|\neg n,\neg r) \end{aligned}$$

You should also assume that "explaining away" occurs whenever there are two edges labeled + going into a node, e,g, $P(r|v,n) < P(r|v)$ because the presence of *Night* "explains away" the *Poor Visibility*.

Your goal is to determine, for each of the pairs of conditional probability values below whether one is larger than the other, they are equal, or they are incomparable with the information we have given you.

(a) $P(s|v)$ and $P(s)$

(b) $P(v|t)$ and $P(v)$

(c) $P(s|v,\neg n)$ and $P(s|v)$

(d) $P(w|v)$ and $P(w)$

(e) $P(w|a,v)$ and $P(w|a)$

(f) $P(w|r,v)$ and $P(w|r)$

(g) $P(w|a,r,v)$ and $P(w|a,r)$