

# Artificial Intelligence: Exam 1 Solutions

March 3, 2005

This exam includes six problems. You are required to answer Problem 1. You may pick any four of the remaining five to solve. (If you solve all five you may mark one as extra credit, which will be worth up to an additional ten points on the exam.)

The length of the exam is 70 minutes.

Print your name (10 points):

---

## Problem 1 (18 Points) *Required*

- (a) For each of the following algorithms, consider an agent that uses only this algorithm to decide which action(s) to take. What type of agent would they be? Choose from among the following: a) simple reflex, b) problem-solving, c) knowledge-based, d) goal-based, and e) utility-based.

- Propositional Logic Resolution: C (logic), A, B, or D also apply depending on the problem
- $A^*$ : B, D, E
- Best-first search: B, D

List all that apply.

- (b) Poker (with betting) is a a) zero-sum game, b) deterministic search problem, c) contingency search problem, d) admissible heuristic, e) a and c, f) b and c or g) all of the above. (Circle one)

E

- (c) Under what conditions is breadth-first search complete? Optimal?

It is complete when the branching factor is finite. It is optimal when the step cost is uniform.

- (d) What is the time and space complexity of iterative-deepening search?

Time:  $b^d$ , Space:  $bd$  where  $b$  is the branching factor and  $d$  is the depth of the shallowest goal node.

- (e) If one uses simulated annealing instead of hill-climbing, what problems are they trying to avoid?

Local maxima

- (f) Define the *utility* of a node in the search tree of a game.

It is the numeric value given to end-game nodes which describes the utility of the outcome for each player. For example in Chess the utility is either +1 when the first player wins, -1 when the second player wins, or 0 if the game is a draw.

- (g) What is the difference between greedy and  $A^*$  search?

In  $A^*$  the function  $f(state)$ , which determines the order in which nodes are expanded, includes both the distance from the initial state and the heuristic (estimated distance to the goal state), i.e.  $f(state) = g(state) + h(state)$ . Greedy only considers the distance to the goal, i.e.  $f(state) = h(state)$ .

## Problem 2 (18 Points)

(a) Give an example of a search space in which breadth-first search is likely to take much more time than depth-first search. You may use an *artificial example*, that is your space may not correspond to any real problem.

Consider a search space in which all goal nodes are found deep within the search tree, and can be found on most branches from the root (i.e. it does not matter which branch DFS decides to explore first). BFS is likely to take much longer than DFS to solve problems in this search space. Examples are games (where repeated states are avoided) that require many moves before either player has won, or large mazes with the start and goal at either end.

(b) Give an example of a search space in which depth-first search is likely to take much more time than breadth-first search. Again you may use an *artificial example*.

Consider a search space in which the branching factor is much smaller than the average depth, the goal is only a few steps from the initial state, and the number of goal states is small compared to the branching factor. Since fewer of the branches have goal states, DFS is unlikely to get lucky and select a branch that leads directly to the goal. For example using a map of Tampa to find your way from downtown Tampa to Old Hyde Park.

### Problem 3 (18 Points)

Suppose that Alice, Bob, Cathy, and David need to cross a narrow bridge at night. They are on the left side of the river and they all need to get to the right side. They have one lantern, and they cannot walk without it. In addition, the bridge is narrow, at most two of them can cross at one time. Thus, after the first two people cross the bridge, one of them must return with the lantern; then two more people can cross the bridge. In general, they will be crossing the bridge in groups of one or two, carrying the lantern back and forth.

Different people walk with different speeds: Alice can cross in one minute, Bob takes two minutes, Cathy needs five minutes, and David spends ten. When two cross the bridge together their speed is determined by the slower walker. For example, if Alice and Cathy cross the bridge together, they spend five minutes. The task is to find a sequence of moves that enables these four people to cross the bridge in the *minimal* possible time.

(a) Give an *admissible* heuristic for this puzzle. Try to make your heuristic as accurate as possible.

Take the maximum travel time for each individual on the left hand side,  $max(left)$ .

If  $max(left) = 0$ , everyone is across so return 0. If  $max(left) = 2$ , only one more crossing is needed so return  $max(left)$ . Otherwise return  $max(left) + 1$ .

*This is an admissible heuristic, it may not be the most accurate.*

(b) Give the optimal solution to this puzzle, that is, describe the optimal sequence of moves and determine the resulting time. In this question, you do not need to use any specific search algorithm. Just give the answer. (Hint: the optimal solution takes less than 19 minutes.)

A = Alice

B = Bob

C = Cathy

D = David

$l$  = the lantern

Left Side	Right Side	Heuristic	Min distance to solution
A, B, C, D, $l$	none	11	17
C, D	A, B, $l$	11	15
C, D, A, $l$	B	11	14
A	C, D, B, $l$	2	4
A, B, $l$	C, D	2	2
	A, B, C, D, $l$	0	0

## Problem 4 (18 Points)

Consider the task of modifying MINIMAX for a three-player game. The players make moves in the pre-defined order (first, second, third, first, second, third,...) until one of them wins. Describe a modified MINIMAX algorithm for playing this game.

Modify the utility function to return an array which specifies the utility (or payoff) for each player, i.e.  $utility[0]$  is Player 1's utility,  $utility[1]$  is Player 2's utility, and  $utility[2]$  is Player 3's utility. In the modified algorithm each player tries to maximize their utility.

**3-Player\_Minimax\_Decision**(state, playerIndex) returns a state

$v \leftarrow$  **3-Player\_Max\_Value**(state, playerIndex)

return the successor with value  $v$

**3-Player\_Max\_Value**(state, playerIndex)

if **Terminal**(state)

return **Utility**(state)

$v[\text{playerIndex}] \leftarrow -\infty$

$\text{nextPlayer} = (\text{playerIndex} + 1) \% 3$

for all successors  $s$

$v[\text{playerIndex}] \leftarrow \max(v, \text{3-Player_Max_Value}(s, \text{nextPlayer}))$

return  $v$

## Problem 5 (18 Points)

In Monty Python's Quest for the Holy grail, King Arthur and Sir Bedevere assist a group of peasants by leading them to use logic to prove that a woman (let's call her A) is a witch with the assistance of a duck (let's call it D). The agreed upon facts used by Sir Bedevere and King Arthur are similar to these:

- All people that are made of wood are witches.
- All ducks are made of wood.
- Anything that weighs the same as a duck is made of wood.
- The woman (A) weighs the same as the duck (D).

With these facts use First-Order Logic (FOL) and resolution to prove that the woman is a witch. (You can assume that she is a person.) For full credit you must rewrite these facts in FOL, then convert them to Conjunctive Normal Form (CNF), and use *only unification and resolution* to prove that the woman is a witch. Show each step of your conversion to CNF and each step of the resolution-based proof.

### Convert to FOL gives:

1.  $\forall x Person(x) \wedge Wood(x) \rightarrow Witch(x)$
2.  $\forall y Duck(y) \rightarrow Wood(y)$
3.  $\forall x, y Duck(y) \wedge Equals(weight(x), weight(y)) \rightarrow Wood(x)$
4.  $Equals(weight(A), weight(D))$
5.  $Person(A)$  (assumed)
6.  $Duck(D)$  (assumed)

### Convert to CNF gives:

1.  $\neg Person(x) \vee \neg Wood(x) \vee Witch(x)$
2.  $\neg Duck(y) \vee Wood(y)$
3.  $\neg Duck(y) \vee \neg Equals(weight(x), weight(y)) \vee Wood(x)$
4.  $Equals(weight(A), weight(D))$
5.  $Person(A)$  (assumed)
6.  $Duck(D)$  (assumed)

### Proof:

7.  $\neg Duck(D) \vee Wood(A)$  Unify 3,4
8.  $Wood(A)$  Resolution 6,7
9.  $\neg Person(A) \vee Witch(A)$  Unify 1,8
10.  $Witch(A)$  Resolution 5,9

## Problem 6 (18 Points)

Create an algorithm for determining the truth value of a sentence in propositional logic. Your algorithm should take two inputs, a proposition, and a model; and return the truth-value of the proposition. You can assume that the proposition is grammatically correct and that it *does not* include parentheses. Assume also that a function `DECOMPOSE(sentence, operator)` exists and that it returns the two sentences on either side of the left-most occurrence of the given operator. For example:

`DECOMPOSE( $P \vee Q$ ,  $\vee$ )` returns the array  $\{P, Q\}$ .

The model can be used to look up the truth value of a simple proposition (a proposition that contains no logical operators). Hint: Try a recursive solution.

First we need to consider the order of operations to determine how a proposition should be decomposed. For example:  $\neg P \vee Q \wedge R \leftrightarrow \neg Q \rightarrow S$

So I define an array called *operators* =  $\{\leftrightarrow, \rightarrow, \vee, \wedge, \neg\}$  which contains the operators in order from last to first in the order of operations. Now I can write the algorithm:

```
Truth_Value( $S$ , model) returns TRUE or FALSE
  if there are no operators in  $S$ 
    return model( $S$ )
```

```
  for  $i \leftarrow 1$  to 4
```

```
    if ( $P$  contains operators[ $i$ ])
```

```
       $\{P, Q\} = \text{DECOMPOSE}(S, \text{operators}[i])$ 
```

```
      return (Truth_Value( $P$ , model) operators[ $i$ ] Truth_Value( $Q$ , model))
```

```
  return  $\neg$ model( $S$ )
```