

COMP472/6721 – Winter 2003 Midterm Examination — Solutions

26. February 2003

Name:

ID:

The *academic code of conduct* holds. The following questions have to be answered without any help from books, notes, processors, or third persons. Please write in pen, not pencil.

The exam will be graded out of 100 points (4 questions). Please indicate clearly which question you do not answer by crossing out its number in the following table:

Question	1	2	3	4	5
Points					

Question 1 (25 points) Please answer the following questions. Give short, precise statements (2–3 sentences max.)

a. (5 points) When is a search algorithm *admissible*? Are all A* algorithms admissible?

Solution. A search algorithm is admissible if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists. All A* algorithms are admissible.

b. (5 points) Discuss: the ALPHA-BETA procedure can produce much better moves than plain MINIMAX.

Solution. The ALPHA-BETA procedure produces exactly the same moves as MINIMAX. However, to determine a move, it only needs to examine $O(b^{d/2})$ nodes in the best case ($O(b^{3d/4})$ average, $O(b^d)$ worst case), thus allowing to either find the same move with less effort or searching (on average) twice as deep into the search tree, which usually improves the overall performance of the search.

c. (5 points) What is an *Ontology*? Can you use a *Semantic Network* to build one?

Solution. An Ontology defines the concepts and relationships for a particular domain. This can be done with a Semantic Network, by defining concepts with nodes and their relationships through arcs.

d. (5 points) When is a planning algorithm *sound* and *complete*?

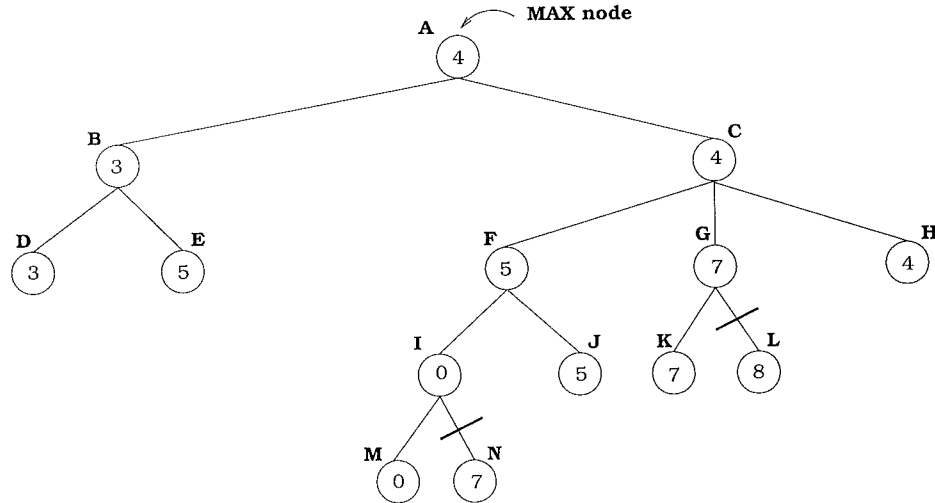
Solution. A planning algorithm is sound if all solutions found are legal plans. It is complete if a solution can be found whenever one actually exists (it is strictly complete if all solutions are included in the search space).

e. (5 points) The MINIMAX algorithm assumes that the opponent (MIN) plays optimally. What happens if he does not?

Solution. In exhaustively searched game trees (i.e., not searching to a fixed ply depth) this always improves the chances for MAX. In general, this may not be the case (though it usually does).

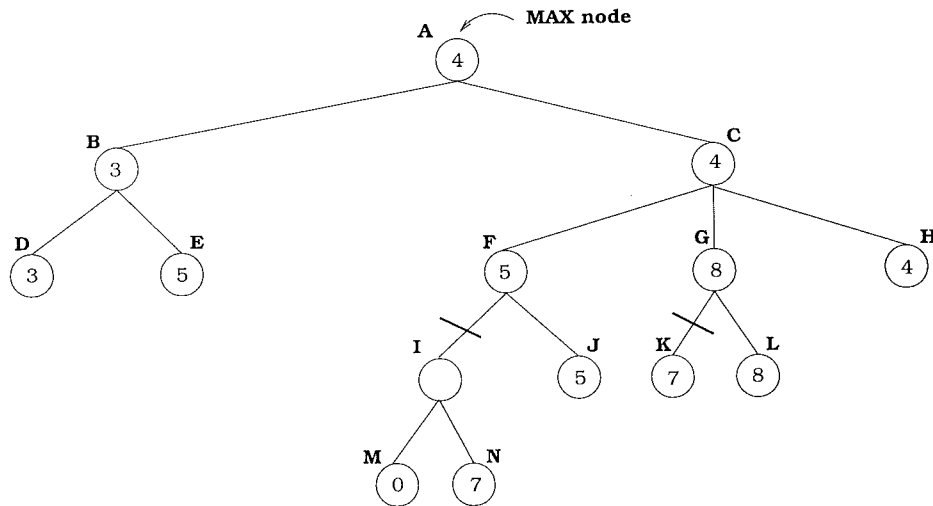
Question 2 (25 points)

a. (10 points) Perform a left-to-right ALPHA-BETA prune on the following tree:



Solution. Two prunes as shown above. MIN on left-most deepest subtree (3,5) to get α at A = 3. From next left-most deepest subtree (0,7), MIN of 0, prune. F is now 5 and β at C = 5. Subtree (7,8) is MAX, so with 7, 8 is β pruned. Seeing the right-most 4, β at C = 4, and α at A = 4, the final backed up value.

c. (10 points) Now perform a right-to-left ALPHA-BETA prune on the same tree:



Solution. Again, two prunes: Starting at the right-most deepest subtree, 4 gives β of C = 4. Seeing 8 in subtree (7,8), 7 is β -pruned. Seeing 5 next the entire subtree (I, M, N) is β -pruned. A takes on the α value of 4. 5 and then 3 are examined (MIN) in the left-most subtree. The final value of A = 4.

d. (5 points) Discuss why different pruning occurs.

Solution. The ALPHA-BETA procedure performs a depth-first search and propagates the values up the path. Thus, the result depends on the order in which the successors of a node are examined.

Question 3 (25 points) Three cannibals and three missionaries are standing on the west bank of a river. A boat is available that will hold either one or two people. If the missionaries are ever outnumbered — on either bank or in the boat — the cannibals will eat them. Your job is to determine a sequence of trips that will get everyone across the river to the east bank.

a. (5 points) Choose a suitable state representation and show how you use it to encode the start state and the goal state.

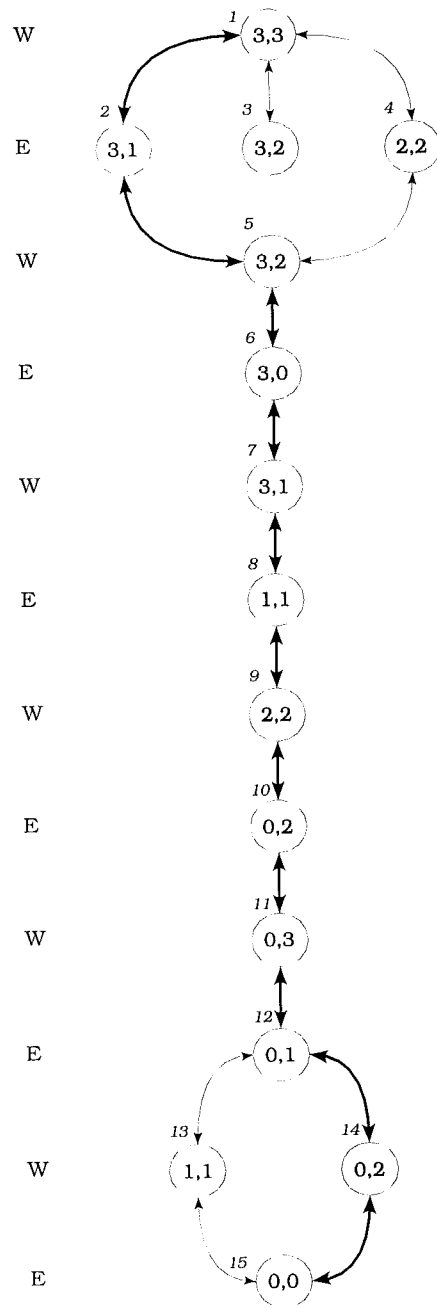
Solution. There are many possible representations, one is the tuple (M, C) with M = number of missionaries on the west bank and C = number of cannibals on the west bank. Then, start = $(4, 4)$ and goal = $(0, 0)$.

b. (5 points) How many different (distinct) states occur in this problem?

Solution. This obviously depends on the chosen representation, but with the one shown above: $M \times C = 4 \times 4 = 16$ different states. This includes the illegal states (there are 10 legal states). If you choose to encode the boat position, the number of possible states doubles (note that in this case you cannot reach all possible legal states from the start state).

c. (15 points) Show the state space search graph. Do not draw any illegal states. Number all distinct states and terminate the search at any repeated state (in any path). Clearly mark a possible path from the start state to the goal state.

Solution. The graph shows the complete search space when we include the boat position in the state description (drawn on the left for better clarity). A possible path is drawn in bold (there are 4 different paths in total).



Question 4 (25 points) Consider the following depth-first search algorithm implementation in Scheme:

```
(define (search state path exit)
  (if (goal? state)
      (exit (reverse path))
      (for-each (lambda(s)
                 (if (and (not (illegal? s))
                          (not (member s path)))
                     (search s (cons s path) exit)))
                (generate-states state))))
```

You can assume that the functions `illegal?` and `generate-states` have been implemented for a given search problem.

a. (15 points) Enhance the algorithm so that it only searches upto a given fixed depth (please indicate clearly which lines of the algorithm you change or where you add code).

Solution:

```
(define (search state depth path exit)
  (if (goal? state)
      (exit (reverse path))
      (if (> depth 0)
          (for-each (lambda(s)
                     (if (and (not (illegal? s))
                              (not (member s path)))
                         (search s (- depth 1) (cons s path) exit)))
                    (generate-states state))))))
```

b. (5 points) Show how to call your modified search function to search a problem until depth 10:

```
(define (solve-problem)
  (call-with-current-continuation
    (lambda (exit) (search start-state (list start-state) exit))))
```

Solution:

```
(define (solve-problem depth)
  (call-with-current-continuation
    (lambda (exit) (search start-state depth (list start-state) exit))))
```

Then call `(solve-problem 10)`.

c. (5 points) How can you use your modified search function to implement iterative deepening? Give a short sketch (code and explanation).

Solution: Start search with `depth=1`. Increase depth by 1 until goal has been found:

```
(define (iterative-deepening-search depth)
  (let ((result (solve-problem depth)))
    (if (list? result)
        result
        (iterative-deepening-search (+ 1 depth)))))
```

And call `(iterative-deepening-search 1)`.

Question 5 (25 points) You are the proud owner of a tasty banana cake. Your goal is to have the cake and eat it, too! Fortunately, you also have all the ingredients necessary for baking another cake, however, you only have enough ingredients for baking one cake and cannot buy more.

a. (5 points) Encode the operators *Eat* and *Bake* into the STRIPS representation. Baking a cake requires having the necessary ingredients. Eating the cake results in you having eaten the cake.

Solution:

Eat(Cake)	Bake(Cake):
Pre: Have(Cake)	Pre: Have(Ingredients)
Add: Eaten(Cake)	Add: Have(Cake)
Delete: Have(Cake)	Delete: Have(Ingredients)

b. (5 points) Give the description of the start (initial) state and the goal state.

Solution:

Start state: Have(Cake), Have(Ingredients)

Goal state: Have(Cake), Eaten(Cake)

c. (15 points) Run the STRIPS algorithm on this problem:

1. $state = initial-state; plan = []; stack = [];$
2. Push *goals* on *stack*
3. Repeat until *stack* is empty:
 - a) If top of *stack* is *goal* that matches *state*, then pop *stack*
 - b) Else if top of *stack* is a conjunctive goal *g*, then select an ordering for the subgoals of *g* and push them on *stack*
 - c) Else if top of *stack* is a simple goal *sg*, then
 - Choose an operator *o* whose add-list matches goal *sg*
 - Replace goal *sg* with operator *o*
 - Push the preconditions of *o* on the *stack*
 - d) Else if top of *stack* is an operator *o*, then
 - $state = apply(o, state)$
 - $plan = [plan; o]$

⇒ *continued on next page!*

5c. (continued) Show each step of the planning process. In the "Iter" column, also indicate which branch of the algorithm (a-d) you follow. Is it possible to construct a plan? Is it optimal?

Solution:

Iter	Stack	State	Plan
1	Have(Cake), Eaten(Cake)	Have(Cake), Have(Ingredients)	[]
2b	Have(Cake), Eaten(Cake) Have(Cake) Eaten(Cake)	Have(Cake), Have(Ingredients)	[]
3c	Have(Cake), Eaten(Cake) Have(Cake) Eat(Cake) Have(Cake)	Have(Cake), Have(Ingredients)	[]
4a	Have(Cake), Eaten(Cake) Have(Cake) Eat(Cake)	Have(Cake), Have(Ingredients)	[]
5d	Have(Cake), Eaten(Cake) Have(Cake)	Have(Ingredients), Eaten(Cake)	[Eat(Cake)]
6c	Have(Cake), Eaten(Cake) Bake(Cake) Have(Ingredients)	Have(Ingredients), Eaten(Cake)	[Eat(Cake)]
7a	Have(Cake), Eaten(Cake) Bake(Cake)	Have(Ingredients), Eaten(Cake)	[Eat(Cake)]
8d	Have(Cake), Eaten(Cake)	Eaten(Cake), Have(Cake)	[Eat(Cake), Bake(Cake)]
9a		Eaten(Cake), Have(Cake)	[Eat(Cake), Bake(Cake)]

Yes, it is possible to construct a plan. It is optimal, since the subgoal *Have(Cake)* can always be achieved without undoing *Eaten(Cake)*.