Name:_____

1	2	3	4	5	6	7	8	9	total
10	*10*	*40*	*30*	*20*	*10*	*30*	*25*	*30*	*205*

UMBC CMSC471/671 Final Exam
December 14, 2006

Please write all of your answers on this exam. The exam is closed book and consists of N problems which add up to M points. You have the two hours to work on this exam. Good luck.

0. Warm Up Exercise (0)

The department just purchased a FixBot maintenance robot (model 2.0) and plans to use it to maintain all the department's robots who do not maintain themselves. Is there a problem with this plan?

This is a direct variation on the famous barber paradox.

1. Uninformed search (10)

Briefly discuss the advantages and disadvantages of depth and breadth first search. What sort of problem is each appropriate for?

DFS can find a solution quickly and it can be done efficiently using a tack, requiring little memory. But, it may not find the best solution or even a very good one and it can fail to find a solution at all of the search space is infinite.

BFS will always find a solution if one exists and the first solution it finds will be the best one. However, it requires a lot of memory and if the search pace is large it can take a long time to find a solution.

2. Heuristic search (10)

Briefly describe all of the conditions that must hold in order for an instance of graph search (i.e., algorithm A) to be an instance of A* search. Include any relevant constraints on the size and structure of the graph, the costs associated with the arcs, the number of goal states, and the heuristic function employed.

All of the arc cost must be positive. The heuristic function must never overestimate the true cost to reach the nearest goal.

3. True/False (40 points)

Circle either T or an F in the space before each statement to indicate whether the statement is true or false. If you think the answer is simultaneously true and false, quit while you are ahead. There is no penalty for incorrect answers (but then, there are no points for incorrect answers either).

T F Every complete inference procedure is also sound. true

T F Every sound inference procedure is also complete. false

T F An incomplete inference procedure can produce an incorrect answer. false

T F An unsound inference procedure can produce an incorrect answer. true

T F The scope of a variable in Prolog is just the clause it is in. true

T F Prolog programs are sound but not complete. false

T F Hill climbing search algorithms only work for search spaces that are two-dimensional or have solution-preserving projections onto two-dimensions. false

T F Iterative deepening is essentially a technique to combine the best features of breadth-first and depth-first search. true

T F Overfitting can occur in machine learning when the data contains many irrelevant attributes. true

T F The decision tree learning algorithm covered in class finds an optimal decision tree, i.e., one that minimizes the number of questions needed to classify a case. false

T F Ockham's razor is a heuristic that says to prefer the simplest consistent explanation . true

T F Decision tree learning can not be used if the data is noisy. false

T F In a proper Bayesian network, a node is always conditionally independent of its non-descendants given its parents. true

T F *Information gain* is used to determine the network structure in a Bayesian Network. false

T F Random variables A and B are independent if p(A^B) = p(A|B)*p(B). false

T F If two variables are independent, then they are always conditionally independent given an arbitrary third variable. false

T F In searching for a plan, a partial order planner searches though a space formed by possible situations. false

T F A partial order planner can either search forward from an initial state to a goal, backward from a goal state to the initial state, or in both directions toward the middle. false

T F The situation calculus is an approach to reasoning about changes in random variables using first order logic. false

T F Resolution is refutation-complete, which means that if a set of sentences is unsatisfiable, then a contradiction can always be derived in a finite amount of time. true

4. English to logic (30)

Translate the following statements into a **single sentence** in first order logic, choosing appropriate predicates and functions.

(a) Good food is not cheap and cheap food is not good.

"x [food(x) ^ good(x) ® ~cheap(x)]

This reduces to a simple implication that I identical in meaning to [food(x) ^ cheap(x) ® ~good(x)]

(b) If a computer beats Kramnik, then that computer beats every person.

"c computer(c) ^ beats(c,kramnik) ® ("x person(x) ® beats(c,x))

(c) Every person is either good or bad but not both.

"x person(x) ® (good(x) ^ not (bad(x)) v (not(good(x)) ^ bad (x))

Of course there are other ways to do this, including the following:

"x person(x) ® [(good(x) v bad (x)) ^ (good(x) ⇔ not(bad(x)))]

Now rewrite each of these in clausal normal form, i.e. as a set of sentences without quantifiers in which each sentence is a disjunction of atomic literals. Recall that an atomic literal is either a predicate (i.e, loves(john,X)) or the negation of a predicate (i.e., not(loves(john, father(mary)))).

(a) Good food is not cheap and cheap food is not good.

~food(x) v ~good(x) v ~cheap(x)

(b) If a computer can beat Kramnik in chess, then a computer can beat anyone in chess.

~computer(c) v ~beats(c,kramnik) v ~person(x) v beats(c,x)

(c) Every person is either good or bad but not both.

~person(x) v good(x) v bad(x)
~person(x) v ~good(x) v ~bad(x)

5. Logic to English (20)

(a) Which of the following, if any, are correct translations of "No two adjacent countries have the same color"?

i. ∀x ∀y ~Country(x) Ú ~Country(y) Ú ~Adjacent(x,y) Ú ~(Color(x) = Color(y))
ii. ∀x ∀y (Country(x) ^ Country(y) ^ Adjacent(x,y) ^ ~(x = y))) ® ~(Color(x) = Color(y))
iii. ∀x ∀y Country(x) ^ Country(y) ^ Adjacent(x,y) ^ ~ (Color(x) = Color(y))
iv. ∀x ∀y (Country(x) ^ Country(y) ^ Adjacent(x,y)) ® Color(x ≠ y)

(i) and (ii) are correct. Adding a constraint that x≠y is not necessary, since a country can not be adjacent to itself. (iii) is not correct. (iv) is malformed due to Color(x ≠ y).

(b) Which of the following are semantically and syntactically correct translations of "No dog bites a child of its owner" ?

i. ∀x Dog(x) => ~Bites(x,Child(Owner(x)))
ii. ~$x ~$y Dog(x) ^ Child(y,Owner(x)) ^ Bites(x, y)
iii. ∀x Dog(x) => (∀y Child(y,Owner(x)) => ~Bites(x, y))
iv. ~$x Dog(x) => ($y Child(y,Owner(x)) ^ Bites(x, y))

(ii) and (iii) are ok. (i) is bad because it uses child as a function, rather than a relation. (iv) says that it's not the case that every dog bites some child of its owner.

6. Failure as negation (10)

(a) Describe the relationship, if any, between Prolog's *failure as negation* and *non-monotonic reasoning*.

Both involve treating the inability to prove a goal as a condition of a proof. For example, we might encode a default rule that all birds fly as: Fly(X) :- bird(X), not(flightless(X).

(b) Write a Prolog rule encoding the knowledge that "All birds fly, except for penguins, kiwis and dead birds" choosing appropriate names for your predicates.

```
fly(X) :- bird(X), not(flightless(X).
flightless(X) :- penguin(x).
flightless(X) :- kiwi(x).
flightless(X) :- dead(x).
```

7. Decision trees (30)

Note

ln(1/4) = -2.0
ln(5/12) = -1.263
ln(1/2) = -1.0
ln(7/12) = -.778
ln(3/4) = -0.415
5/12 = 0.417
7/12 = 0.583

You are constructing a decision tree and are deciding which attribute is best to ask about first using the *information gain* metric. You are given a set of 128 examples, with 64 positively labeled and 64 negatively labeled. There are three attributes, HomeOwner, InDebt, and Rich. For 64 examples, HomeOwner is true with 1/4 negative and 3/4 positive. For 96 examples, InDebt is true with half positive and half negative. For 32 examples, Rich is true with 3/4 positive and 1/4 negative. Recall that the entropy of a distribution P with probabilities $(p_1 \ldots p_n)$ is defined as $-\Sigma\ p_i * \ln(p_i)$. Information gain is the difference in entropy before asking the question and the (weighted) entropy after.

ALL	neg	pos
	64	64

HO	neg	pos
true	16	48
false		

ID	neg	pos
true	48	48
false		

RICH	neg	pos
true	8	24
false		

(a) fill out the tables to the right to help you with your computations.

(b) What's the entropy of the initial data?

entropy = I(1/2,1/2) = - (0.5*-1 + 0.5*-1) = 1

(b) What is the information gain of splitting on the HomeOwner attribute as the root node?

The entropy of the result is

0.5 * I(1/4,3/4) + 0.5 * I(1/4,3/4) = I(1/4,3/4) = 2*.25 + .415*.75 = 0.81125

So the information gain is 0.18875

(c) What is the information gain of splitting on the InDebt attribute as the root node?

The entropy of the result is

.75 * I(1/2,1/2) + .25*I(1/2,1/2) = I(1/2,1/2) = 1

So the information gain is 0.0

(d) What is the information gain of splitting on the Rich attribute as the root node?

The entropy of the result is

.25*I(1/4,3/4) + .75*I(5/12, 7/12) = .25*0.81125 + .75*(.417*1.263 + .583*.778)
= .203 + .739 = .942

So the information gain is ..058

(e) Which attribute do you split on?

HomeOwner

8. Bayesian networks (25)

You are designing a troubleshooting advisor for PCs. Let CF be a boolean random variable representing whether the computer fails (CF=true) or not. Assume there are two possible causes of failure: Electricity-Failure and Malfunction-of-the-Computer represented using the Boolean random variables EF and MC, respectively. The table below lists some of the common formulae. Assume the following probabilities

P(EF) = 0.1
P(MC) = 0.2
P(CF | ~EF ^ ~MC) = 0.0
P(CF | ~EF ^ MC) = 0.5
P(CF | EF ^ ~MC) = 1.0
P(CF | EF ^ MC) = 1.0

rule	not conditioned	conditioned on C
basic	P(A,B) = P(A|B)P(B)	P(A,B|C) = P(A|B,C)P(B|C)
Bayes	P(B|A)P(A) = P(A|B)P(B)	P(B|A,C)P(A|C) = P(A|B,C)P(B|C
independence	A&B are independent iff P(A|B) = P(A)	A&B cond. Indep. given C iff P(A|B,C) = P(A|C)
chain rule	P(A,B,C) = P(A)P(B|A)P(C|A,B)	
axiom	P(A) + P(~A) = 1	P(A|C) + P(~A|C) = 1
marginalization	P(A) = SUMb P(A,B)	P(A|C) = SUMb (P(A,B|C)

(a) Are EF and MC independent?

yes

(b) Draw the Bayesian Network for this problem.

EF and MC are parents of CF

(b) Compute P(CF ^ ~EF ^ MC). Hint: use the chain rule.

P(CF ^ ~EF ^ MC) = P(CF | MC ^ ~EF) P(MC | EF) P(~EF)
= P(CF | MC ^ ~EF) P(MC) P(~EF)
= (0.5)(0.2)(0.9) = 0. 09

(c) Compute P(MC | EF)

Since MC and EF are independent, P(MC|EF) = P(MC) = 0.2

(d) Compute P(EF | CF)

P(EF|CF) = P(EF ^ ~MC|CF) + P(EF ^ MC | CF)
= [P(CF ^ EF ^ ~MC)/P(CF)] + [P(CF ^ EF ^ MC)/P(CF)]
= (0.08+0.02)/0.19
= 0.5263

9. Planning (30 points)

In the minimal block world domain used in HW7 and in class, there are just five predicatesdescribing states (ontable/1, holding/1, handempty, on/2, clear/1) and four operators (pickup, putdown, stack, unstack). Operators are defined in terms of five elements: description, preconditions, adds, deletes, and constraints, as in this example:

stack operator
Description: stack(X,Y)
Preconditions: [holding(X), clear(Y)]
Adds: [on(X,Y), clear(X)]
Deletes: [holding(X), clear(Y)]
Constraints: [not(Y=table)]

Define the following strips operators in this format.

(a) The **clone** operation allows the robot to create a copy of an existing object that it is holding. The newly created object will be on the table and have nothing on it. If the robot clones an object named *foo*, then the new copy will be named *copyOf(foo)*.

clone operator
Description: clone(Obj)
Preconditions: [holding(Obj)]
Adds: [ontable(copyOf(Obj), clear(CopyOf(Obj))]
Deletes: [ ]
Constraints: [ ]

(b) The **destroy** operation allows the robot to destroy an object that it is holding. The object is removed from the world description.

destroy operator
Description: destroy(Obj)
Preconditions: [holding(Obj)]
Adds: [handempty]
Deletes: [holding(obj)]
Constraints:

(c) The **swap** operation can reverse the order of the top two objects in a stack. That is, if a is on b, and b is on c, and nothing is on a, then after the **swap** operation, b will be on a and a will be on c and nothing will be on b. In order initiate a swap operation, the robot's hand has to be empty. Nothing else in the state will have changed.

swap operator
Description: swap(Obj1,Obj2)
Preconditions: [on(Obj1,Obj2), clear(Obj1), handempty]
Adds: [on(Obj2,Obj1), clear(Obj2)]
Deletes: [on(Obj1,Obj2), clear(Obj1)]
Constraints: [ ]