EE3-16

The Answers

The Answers

1

a)

[bookwork]

depth first: choose node on search frontier furthest from start
depth-limited depth first: stop depth first search if path depth d=limit
id depth first: depth limited depth limited first with limit=0, limit=1, limit=2

df: optimal no, complete no, time complexity b^m (m=max depth of tree), space complexity b*m
dl-df: optimal no, complete no, time complexity b^l (l=limit), space complexity b*l
id df: optimal yes, complete yes, time complexity b^l (l=limit), space complexity b*d

b)

[application]

```
select2( [H|T], (H,Other) ) :-
        select1( T, Other ).
select2( [_|T], Pair ) :-
        select2( T, Pair ).

select1( [H|T], H ).
select1( [_|T], H ) :-
        select1( T, H ).
```

c)

[application]

([(a,A), (b,B), (c,C), (d,D)], T) where
A, B, C, D are l or r to indicate which side of the river, T is the time elapsed

([(a,l), (b,l), (c,l), (d,l)], 0)

([(a,r), (b,r), (c,r), (d,r)], T), T < 16

```
state_change( 2lr, (List, Time), (Newlist, NewTime) ) :-
        select2( [a,b,c,d], Pair ),
        both_left( Pair, List ),
        change_banks( Pair, r, List, NewList ),
        slowest( Pair, Elapsed ),
        NewTime is Time + Elapsed.

state_change( 1lr, (List, Time), (Newlist, NewTime) ) :-
        select1( [a,b,c,d], One ),
        member( (One,l), List ),
        replace( One, r, List, NewList ),
        crossing_time( One, Elapsed ),
        NewTime is Time + Elapsed.
```

Similar rules for one or two people crossing right to left

```
both_left( (X,Y), List ) :-
        member( (X,l), List ),
        member( (Y,l), List ).

change_banks( (X,Y), Bank, List, NewList ) :-
        replace( X, Bank, List, Temp ),
        replace( Y, Bank, Temp, NewList ).

replace( X, B, [(X,_)|T], [(X,B)|T] ).
replace( X, B, [H|T1], [H|T2] ) :-
        replace( X, B, T1, T2 ).

slowest( (X,Y), Slower ) :-
        crossing_time( X, Faster ),
        crossing_time( Y, Slower ),
        Faster < Slower, !.
slowest( (X,Y), Slower ) :-
        crossing_time( X, Slower ).
```

[Solution

1,2, cross
1 comes back
5,8 cross
2 comes back
1,2 cross (again)

time elapsed = 2+1+8+2+2 = 15]

2
a)
[bookwork]
uniform cost: choose node on search frontier with least actual cost from start node (cost function g)
best first: choose node on search frontier with least estimated cost to goal node (heuristic function h)
A*: choose node n on search frontier with least estimated path cost from start node to goal node through n (f = g + h)

Uni cost: optimal yes (assuming …), complete yes, time complexity b^d (m=max depth of tree), space complexity b*d
Best f: optimal no, complete no, time complexity b^d, space complexity b^d (but this is worst case, do much better than this with a good heuristic)
A*: optimal yes, complete yes, complexity depends on heuristic

b)
[understanding]
i)
P_0 & = & \{ (\langle\mathit{start}\rangle, 0) \} \\
P_{x+1} & = & \{ (p_i +\!\!\!+ \langle n_j \rangle, (g+e)) \mid \exists (p_i,g) \in P_x . (\mathit{frontier}(p_i), e, n_j) \in R \}

ii)
P'_0 & = &\{ (\langle\mathit{start}\rangle, 0) \} \\
P'_{x+1} & = & \{ (p_i +\!\!\!+ \langle n_j \rangle, (g+e)) \mid \exists \mathit{op} \in \mathit{Op} . \exists (p_i,g) \in P'_x . \mathit{op}(\mathit{frontier}(p_i)) = (n_j,e) \}

iii)
\forall n_1, e, n_2, (n_1, e, n_2) \in R \iff \exists o \in op. op(n_1) = (n_2, e)

c)
[understanding]
P'_G = \bigcup_{x=0}^{\infty}P'_x

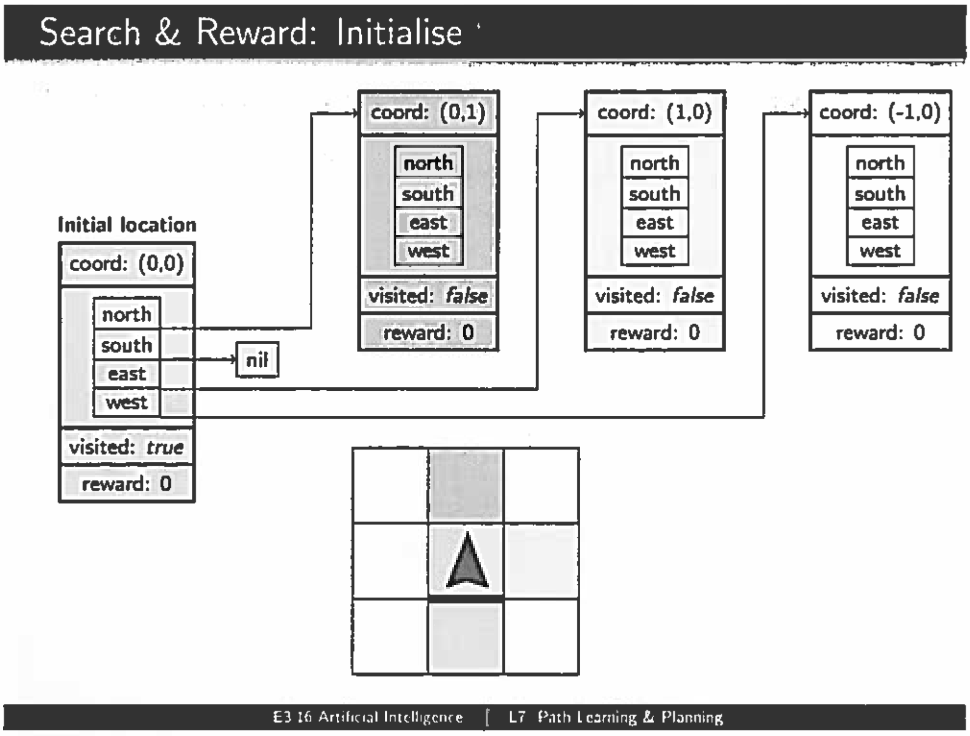
P'_0 = { (<s>, 0) }

P'_1 + { (<s,n_1>, e_1), … } for all n_i such that op(s) = (n_i,e)

So uniform cost picks the only path in P'_0, the cheapest e_i in P'_1, the cheapes in the union of P'_1 and the subset of P'_2 generated, and so on

d)
[understanding]
admissible, monotonic
actual cost of goal node is f*
node expanded if g(n)+h(n) < f*
maximize h(n) without over-estimating

3
a)
[Bookwork]



Path Finding

adopt a basic exploration strategy (random, follow left wall,...) search until exit (goal state) found – receive rewardpropagate reward backwards from exit state through states which lead to such exit state (credit assignment)

Path Following

from each state, move to next state with the highest reward – optimal move

Path Learning (by "reinforcement")

repeat

path following, using a (sub)-strategy to visit unexplored part of the griddo credit assignment on-line (exploration and change detection)

propagate rewards backwards

until: done enough training runs, good enough solution, whole maze explored, run out of time/money...

b)
[Understanding]

exploit until find change then re-search, updating map.

Rate of change of environment faster than replanning

Do not replan – might make plan and be out of date

Do replan – might be constantly interrupting to replan and make no progress

c)
[Understanding]
search to fixed ply rather than exhaustive

use depth first search

apply heuristic function rather than assign 1 or 0

use propagation and cut-off rules to prune search space

4

a)

[bookwork]

resolution: a rule of inference used for automated theorem proving

rule:

p   –p / contradiction

p + q   –p / q

p1 +p2 + ... + pi + ... pm   -pi + q1 + q2 + ... + qn / p1 +p2 + ... + q1 + q2 + ... + qn + ... pm

in logic programming: query and horn clauses

b)

[bookwork]

unification: solving the problem of equating symbolic expressions

algorithm

uninstantiated variable unifies with atom, term or other uninstantiated variable

atom unifies with atom

term unifies with term if functors same, arity same, pairwise arguments unify

in logic programming: way of binding values to variables in resolution

c)

-longnose(X1) V seebetter(X1,Y1)

-bigeyes(X2) V smellbetter(X2,Y2)

-seebetter(X3,Y3) V -smellbetter(X3,Y3) V chase(X3,Y3)

-chase(X4,Y4) V -faster(X4,Y4) V catch(X4,Y4)

-wolf(X5) V -littlegirl(Y5) V -catch(X5,Y5) V eat(X5,Y5)

d)

wolf(bbw)

littlegirl(rrh)

bigeyes(bbw)

longnose(bbw)

faster(bbw,rrh).

-eat(bbw,rrh)

-wolf(X5,Y5) V -littlegirl(X5,Y5) V -catch(X5,Y5)   {X5->bbw, Y5->rrh}

-littlegirl(X5,Y5) V -catch(X5,Y5)

-catch(X5,Y5)

-chase(X4,Y4) V -faster(X4,Y4)   {X4->X5, Y4->Y5}

-seebetter(X3,Y3) V -smellbetter(X3,Y3) V -faster(X4,Y4)   {X3->X4, Y3->Y4}

-longnose(X1) V -smellbetter(X3,Y3) V -faster(X4,Y4)   {X1->X3, Y1->Y3}

-smellbetter(X3,Y3) V -faster(X4,Y4)

-bigeyes(X2) V -faster(X4,Y4) {X2->X3, Y2->Y3}
-faster(X4,Y4)
contradiction
5)
a)
[bookwork]

p & q / p, q

-(p & q), p / -q

-(p + q) / -p, -q

p + q, -p / q

-(p -> q) / p, -q

p -> q, p / q

p -> q, -q / -p

p <-> q, p / q etc.

PB bivalence branch p | -p every formula is either true or false

closure p, -p / close trying to make every formula true on a branch, can't make p and –p true on the same branch

if you have one component of a beta formula on a branch, it can safely be analysed as there is no use in applying pb as it will not add any information, ie

suppose
p + q
p

try branching on subformula p:
branch 1 –p … close immediately
branch 2 p … we already knew that

try branching on sub-formula q:
branch 1 –q
add p … we already knew that but now add
branch 2 q
we have to prove –p on two branches

we can close on literals of we can close on sub-formulas, but any complex formula, trying to F and –F true then we must have at least one literal different in the row of the truth table

if it is sound to close on contradictory sub-formulas it is sound to close on complementary literals
As we are guaranteed to end up with literals the algorithm is complete

b)
[Application]

1	premise	(-A & -B) -> (out <-> -C)
2	premise	(-A & B) -> (out <-> -C)
3	premise	(A & -B) -> -out
4	premise	(A & B) -> out
5	neg conc	-(out <-> ((A & B) + (-A & -C)))
		Branch 1
6	PB1	out
7	<->, 5, 6	- ((A & B) + (-A & -C))
8	a, 7	-(A & B)
9	a, 7	-(-A & -C)
10	b, 3, 6	-(A & -B)
		Branch 1.1
11	PB1	(-A & -B)
12	a, 11	-A
13	a, 11	-B
14	b, 1, 11	out <-> -C
15	b, 6, 14	-C
16	b, 9, 12	--C
17	-- 16	C
		Close (15,17)
		Branch 1.2
18	PB2	-(-A & -B)
		Branch 1.2.1
19	PB1	-A & B
20	a, 19	-A
21	a, 19	B
22	b, 2, 19	out <-> -C
23	b, 6, 14	-C
24	b, 9, 20	--C
25	--, 24	C
		Close (23,25)
		Branch 1.2.2
26	PB2	-(-A & B)

[we now have –(A & B), -(-A & B), -(-A & -B) and –(A & -B) on the branch…]

		Branch 2
27	PB2	-out
28	<->, 5, 27	(A & B) + (-A & -C)
29	b, 4, 27	-(A & B)
30	b, 28, 29	-A & -C
31	a, 30	-A
32	a, 31	-C

		Branch 2.1
33	PB1	(-A & B)
34	a, 33	-A
35	a, 33	B
36	b, 2, 33	out <-> -C
37	b, 27, 36	--C
38	--, 37	C
		Close (32,38)
		Branch 2.2
39	PB2	-(-A & B)
40	b, 39, 31	-B
		Branch 2.2.1
41	PB1	-A & -B
42	b, 1, 42	out <-> -C
43	b, 27, 42	--C
44	--, 43	C
		Close (32,44)
		Branch 2.2.2
46	PB2	-(-A & -B)
47	b, 31, 46	--B
48	--, 47	B
		Close (40,48)

6

a)

[bookwork]

wff = □*wff* | ◊*wff*

M = <W,R,||>
Where W is a non-empty set of worlds
R is binary accessibility relation on W
|| denotation function p -> 2^W maps each proposition to subset of W where it is true

|=M,a box p <-> forall b . aRb -> |=M,b p
|=M,a dia p <-> exists b . aRb ∧ |=M,b p

b)

[bookwork, understanding]

box p is true because there is no b such that aRb thus the antecedent aRb is false so does not matter that |=M,b p is false, false implies anything is true, so by <-> this means that |=M,a box p is true.

dia true is false because there is no b such that aRb thus the left-hand conjunct is false so false and anything is false, so by <-> this means |=M,a dia p is false

c)

[understanding – but for the first part, just take the model given in part b]
Model M = < {a}, {}, ||p|| = {a} >
Then

|=M,a p	is true	
|=M,a box p	is true	by semantics of box
|=M,a dia p	is true	by implication of axiom
but		
|=M,a dia p	is false	by semantics of dia
contradiction		

assume box p
we want to show dia p

	box p	
->	forall b . aRb -> |=M,b p	by semantics of box
but	forall a . exists b . aRb	by seriality
->	|=M,b p	by forall
->	aRb and |=M,b p	
->	exists b. aRb ∧ |=M,b p	
<->	dia p	

as required

d)
[application]
in **K**
1: -(box p -> dia p)
1: box p
1: - dia p
???

in **S5**
1: -(box p -> dia p)
1: box p
1: - dia p
1: p
1: -p
close

In **K,** we cannot be sure there is a world to go to make either box p or –dia p true.

e)
[application]
1: $\neg(p \leftrightarrow q) \rightarrow (\Box p \leftrightarrow \Box q)$
1: $p \leftrightarrow q$
1: $\neg(\Box p \leftrightarrow \Box q)$

Branch 1
1: $\Box p$
1: $\neg\Box q$
2: $\neg q$
2: p

Model M = < {1,2}, { (1,2) }, ||p|| = {1,2} ||q|| = {1} >