Artificial Intelligence 2005 EJ-16 Master copy 31.3.05
Ite 3.23

SOLUTIONS

1

(a) bookwork

(i) breadth first, expand a node at deepest level, otherwise back up and try node at next deepest level
depth first, expand all nodes at level n, then all nodes at level n+1, then …

(ii) pick a path to expand (the frontier node thereof), note other-paths
find all the places to go from frontier node, make new-paths
for
    breadth first, form a queue (append new-paths to other-paths)
    depth first, form a stack (append other-paths to new-paths)

(iii) let b be branching factor, d be depth of solution, m maximum depth of tree
breadth first: optimal yes, complete yes, time complexity O(b^d), space O(b^d)
depth first: optimal no, complete no, time complexity O(b^m), space O(b*m)

(iv) ID search uses limited depth-first search to successive depths.
Thus it completes exhaustive search to d=0, d=1, d=2, etc
Because of the branching factor, the need to expand a few nodes at low depths several time is a relatively small overhead compared to expanding many nodes a few times at high depths.
It thus combines completeness/optimality of breadth first with time/space complexity of depth first

(b) application

(i) grid list of facts
2-tuple (Location, Colour)
location is 2-tuple (Integer,Integer) giving (X,Y) coordinate
Colour is atom, one of {red, green, blue, yellow, magenta, etc}

(ii) 2 tuple (Location, ChipList)
Location is 2 tuple (Integer,Integer) giving (X,Y) coordinate
ChipList is list of 2 tuple (Colour,Integer)

(iii)
start state e.g. ((2,3), [(red,5), (blue,2), (green,3), …])
goal state e.g. ((27,54), _)

(iv)

```
state_change( east, (Loc,Chips), (NewLoc, NewChips) ) :-
        Loc = (X,Y),
        X1 is X + 1,
        grid( (X1,Y), Colour ),
        append( Front, [ (Colour,N1) | Back ], Chips ),
        N1 > 0,
        N is N1 – 1,
        append( Front, [ (Colour,N) | Back ], NewChips ),
        NewLoc = (X1,Y).
```

Same for west, north and south.

2
(a) [bookwork]
admissible heuristic: never overestimates actual cost of getting to goal from frontier node, important because used in ensuring optimality
pathmax equation restores optimality in heuristics f(succ(n)) = max(f(n), g(succ(n)+h(succ(n)), important as it ensures path cost along any path never decreasing, used to ensure optimality

(b) [bookwork/application]
G' = < start_node, Op > where
start_node is the root node
op is set of state transformers op: node -> (edge,node)

Op has to compute every element of the incidence relation.

What we then need to store with each path we create is f, the path cost function g plus the estimated path cost to the goal given by the heuristic function h.

Then our inductive definition of the graph is given by:

$N'_G = \bigcup_{i=0}^{\infty} N_i$
where
$N_0 = \{ \langle start \rangle \}$
$N_{i+1} = \{ n_{i+1} \mid \exists\, op \in Op \,.\, \exists\, n_i \in N_i \,.\, n_{i+1} = op(n_i,e) \}$

$R'_G = \bigcup_{i=1}^{\infty} R_i$
where
$R_1 = \{ (n_0, n_1) \mid \exists\, op \in Op \,.\, n_1 = op(n_0) \}$
$R_{i+1} = \{ (n_i, n_{i+1}) \mid \exists\, op \in Op \,.\, \exists\, n_i \in N_i \,.\, n_{i+1} = op(n_i,e \}$

$P'_G = \bigcup_{i=0}^{\infty} P'_i$
where
$P'_0 = \{ (\langle start \rangle, 0) \}$
$P'_{i+1} = \{ (p_i +\!+ \langle n_{i+1} \rangle, f) \mid \exists\, op \in Op \,.\, \exists\, (p_i,g) \in P'_i \,.\, (n_{i+1},e) = op(frontier(p_i))$ AND $f = g + e + h(n_{i+1}) \}$

Let f* be the actual cost of getting from the start state to the optimal goal state. Then the A* algorithm constructs:

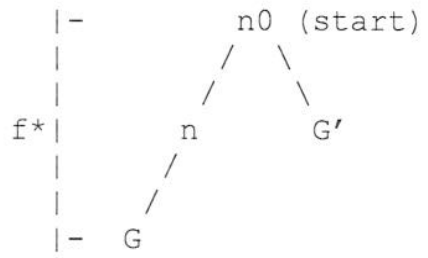
All the paths (p,f) in whichever P'_i such that $f < f^*$
Some of the paths (p,f) in whichever P'_i such that $f = f^*$
None of the paths (p,f) in any P'_i such that $f > f^*$
At each step, A* selects the path in (p,f) from whichever P'_i such that this f is least so far, and expands that.

(c) [bookwork/application]
Optimality:
Optimal solution has cost f* to get to optimal goal G

Suppose A* search returns path to sub-optimal goal G'
We show that this is impossible

f(G')	=	g(G') + h(G')	
	=	g(G') + 0	G' is a goal state, we require h to be 0
	=	g(G')	

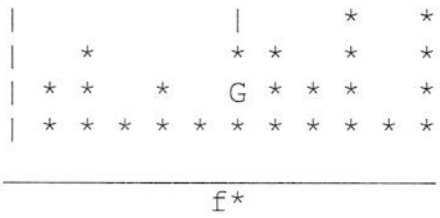
If G' is sub-optimal then g(G') > f*
Now consider a node n on path to optimal solution G



Then:	f*	≥	f(n)	monotonicity
	f(n)	≥	f(G')	otherwise A* expands n first
	f*	≥	f(G')	transitivity of [3]
	f*	≥	g(G')	a contradiction

So either G' was optimal or A* does not return a sub-optimal solution.

Let f* be cost of optimal node.
A* expands all nodes with f-cost less than f*.
A* expands some nodes with f-cost = f*.
A* expands no nodes with f-cost > f*.
Since f(n) = g(n) + h(n), this means that A* expands all those nodes such that h(n) < f* - g(n).
In other words, the more nodes for which this relation holds, the more nodes will be expanded by A* using this heuristic, and the less efficiently will the search space be explored.
Alternatively, consider histogram of nodes according to actual f-cost, whereby f-actual(n) = g(n) + h-actual(n).



Only those nodes to the left of the f* bar will be expanded. In practice of course, we don't have h-actual we just have h. Thus we could have a redistribution of the histogram which pushes more of the nodes to the left of the f* bar.
In other words, we want to ensure f* < f(n) + h(n) < f-actual(n)

(d) [understanding]
use happy and gloomy to compute an interval rather than a point
then use ordering on interval size to decide order in which to expand nodes

3
(a) [bookwork]
max is player trying to win, or MAXimize advantage
min is opponent who attempts to MINimize max's score. Assume that min uses the same information as max and attempts always to move to a state that is worst for max
each leaf node is given a score of 1 or 0, depending on whether the state is a win for max or min respectively
exhaustively generate the graph
propagate leaf values up the graph according to the rules:
if the parent is a MAX, give it the minimum value of its children
if the parent is a MIN, give it the maximum value of its children

(b)[bookwork/application]
exhaustive search is not always possible
therefore use alpha-beta search

➠Associate one of two values with each node

—Alpha value, associated with MAX nodes, which can never decrease
•Alpha is the 'least' MAX can get, given MIN will do its best to minimise MAX's value
-- Beta value, associated with MIN nodes, which can never increase
• Beta is the 'most' MAX can get, given MIN will do its best to minimise MAX's value

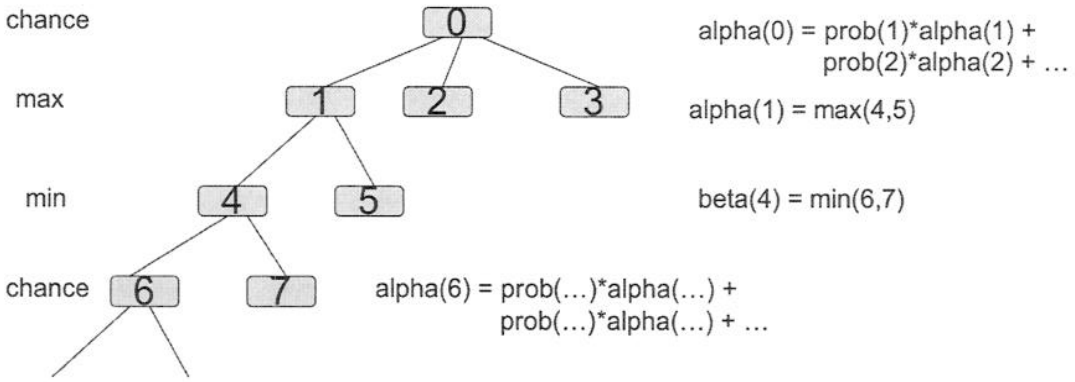
➢**Algorithm**
➠Search to full ply using depth first
➠Apply heuristic evaluation to all siblings at ply

—Assume these are MIN nodes
➠Propagate value of siblings to parent using Minimax rules

—If MIN nodes, back up the maximum value
➠Offer this value to ***grandparent*** MIN node as possible beta cutoff
➠Descend to other grandchilren
➠Terminate (prune) exploration of parent if any of their values is greater than or equal to the beta cutoff
➠Do the same for MAX nodes
➠Two rules for terminating search

—Search stopped below any MIN node having a beta value
less than or equal to alpha value of any of its MAX ancestors
Search stopped below any MAX node having an alpha value
greater than or equal to beta value of any of its MIN ancestors

(c) [application/understanding]
have three rounds one for chance, one for max and one for min
max returns as before
min returns as before
chance returns sum of chance * alpha score for each child



(d) [understanding]
ID search can be used to order nodes in estimate of utility
Effectiveness of pruning is greater when nodes are searched in order of their estimated value.

4
(a) [bookwork]
eliminate implication and equivalence
reduce scope of negation
rename variables
move quantifiers left without changing order
skolemize
drop universal quantifiers
convernt to conjunctive normal form
make each conjunct a separate clause
give variables with same name in different clauses, different names

(b) [understanding]
(i) fact
(ii) rule
(iii) goal

(*c*) [*application*]
$\forall a \forall b.\ parent(a, b) \rightarrow ancestor(a, b)$
$\forall a \forall b \forall c.\ parent(a, c) \wedge ancestor(c, b) \rightarrow ancestor(a, b)$
parent(*Bob*, *Alice*)
parent(*Alice Eve*)

$f1$ $\neg parent(a, b) \vee ancestor(a, b)$
$f2$ $\neg parent(a1, c) \vee \neg ancestor(c, b1) \vee ancestor(a1, b1)$
$f3$ *parent*(*Bob*, *Alice*)
$f4$ *parent*(*Alice Eve*)

$\neg ancestor(Bob, Eve)$

resolve with $f2$, *unifier* $\{a1 = Bob, b1 = Eve\}$
$\neg parent(Bob, c) \vee \neg ancestor(c, Eve)$
resolve with $f3$, *unifier* $\{c = Alice\}$
$\neg\neg ancestor(Alice, Eve)$
resolve with $f1$, *unifier* $\{ a = Alice, b = Eve\}$
$\neg parent(Alice, Eve\}$
resolve with $f4$,
contradiction

(c) [understanding]
Some of the models which make the premises true do not necessarily make the conclusion true, specifically those models where it is a different 'x' or person that makes each of the existential statements true.

So, for example, a model with Sko1 and Sko2 will satisfy the two statements

$\exists x.\ parent(Bob, x1)$
$\exists x.\ parent(x2, Eve)$

with $parent(Bob, Sko1)$ and $parent(Sko2, Eve)$
but $ancestor(Bob, Eve)$ is not a true statement in this model.

This is why the inference rule for existential elimination must introduce a new constant:

$\exists x.\ parent(Bob, x)$
$\exists x.\ parent(x, Eve)$
$parent(Bob, Sko1)$
$parent(Sko2, Eve)$

not

$\exists x.\ parent(Bob, x)$
$\exists x.\ parent\ (x, Eve)$
$parent\ (Bob, Sko1)$
$parent\ (Sko1, Eve)$

5
(a) [bookwork]
sound: |- implies |=
a proof in the system is an entailment in the language
essential

complete |= implies |-
an entailment in the language can be proved by the system
desirable

decidable
theorem algorithm terminates with yes
non-theorem algorithm terminates with no
desirable but can't do much about the language (e.g. predicate logic semi-decidable)

efficient
deal with exponential complexity

(b) [bookwork/understanding]
- *KE proof procedure*
- *to prove Q, begin with ¬Q and search for a refutation*
- *expand ¬Q according to, but clearing away, logical structure of Q*
- *expansion takes form of tree, called a tableau, with formulas labelling nodes*
- *tableau is representation of disjunctive normal form*
- *each branch represents the conjunction of formulas on the branch*
- *the tableau is a disjunction of its branches*

c) [application]

query = $(a \wedge b \wedge \ldots) \wedge (c \vee d \vee \ldots) \wedge (\neg e \wedge \neg f \wedge \ldots)$

d) [application/understanding]

All the search terms are premises
The query is the conclusion
To kick off a KE tableau, at the root we can add all the premises, and the negation of the conclusion

|- $(a \wedge b \wedge \ldots) \wedge (c \vee d \vee \ldots) \wedge (\neg e \wedge \neg f \wedge \ldots)$

$\neg((a \wedge b \wedge \ldots) \wedge (c \vee d \vee \ldots) \wedge (\neg e \wedge \neg f \wedge \ldots))$ ¬ *conc*

branch 1		
2	$\neg(a \wedge b \wedge \ldots)$	*PB* 1/1
branch 1.1		
3	$\neg a$	*PB* 2/1
4	*closes if search term a is in the premises*	
branch 1.2		
5	*a*	*PB* 2/2
6	$\neg(b \wedge \ldots)$	*b* 2, 5
branch 1.2.1		
7	$\neg b$	*PB* 3/1
8	*closes if search term a is in the premises*	
9	*b*	*PB* 3/2 *etc*

Branch 2

11	$(a \wedge b \wedge ...)$	*PB* 1/2
12	$\neg((c \vee d \vee \dots) \wedge (\neg e \wedge \neg f \wedge \dots))$	*b* 11, *conc*

Branch 2.1

13	$\neg(c \vee d \vee \dots)$	*PB* 4/1
14	$\neg c$	*a* 13
15	$\neg d$	*a* 13
16	...	*a* 13

closes if either c or d or ... is in the premises

Branch 2.2

17	$(c \vee d \vee \dots)$	*PB* 4/2
18	$\neg(\neg e \wedge \neg f \wedge \dots)$	b 12 17

Branch 2.2.1

19	$\neg\neg e$	*PB* 5/1
20	*e*	--

closes if e is not in the premises

Branch 2.2.2

21	$\neg e$	PB5/2
22	$\neg(\neg f \wedge \dots)$	*b*, 18, 21

Branch 2.2.2.1

23	$\neg\neg f$	*PB* 6/1
24	*f*	--

closes if f is not in the premises

Branch 2.2.2.2

25	$\neg f$	PB 6/2
26	$\neg(\dots)$	*b*, 22, 25

etc

Assumption is closed world assumption used to close branches, 2.2.2.x, i.e. if the search term is not present in the premises, then its negation must also be present.

6
(a)
wff ::= □ *wff* | ◊ *wff*

(b) [application]

M = <W, R, ||>

W = {α,β,γ,δ,ζ}
R = { α R β, β R δ, γ R α, γ R β, γ R δ, γ R γ, δ R ζ }
|p| = { α, β, γ }
|q| = { β, γ, δ }

(c) [application]
(i) true
p and q is true everywhere from γ

(ii) false
box q is true since q is true in every (the only) world accessible from b (ie d)
but dia p is false since there is no world accessible from d in which p is true

(iii) true
p is true in a
dia q is true in b, since q is true in d and d is accessible from b
so dia dia q is true in a since dia q is true in b and b is accessible from a

(iv) true
anything is necessarily true in a world with no successor

(c) [application]
M = <W,R,P>
W = {a,b}
R = {aRb}
||p|| = {a}
p is true in a
by MP box dia p true
so dia p true in b
but dia p false in b

assume p, show box dia p
suppose p is true in some a
suppose a R b, any b
then b R a by symmetry
so dia p at b
since dia p is true at any (every) b accessible from a
then box dia p is true at a, as required

(d) [application]
1 -(dia p -> box dia p)
1 dia p
1 -box dia p
2 p
3 -dia p
2 -p
close