Ist 3 23

1.
(a) 6 marks (Bookwork)
(b) 6 marks (Application)
(c) 6 marks (Bookwork/Understanding)
(d) 2 marks (Bookwork/Understanding

(a)
strategy:
uniform cost: 'optimal' node based on lowest actual cost from start-node as computed by cost function g
best first: 'optimal' node based on lowest estimated cost from node-goal as computed by heuristic function h
A*: 'optimal' node based on lowest estimated cost of path from start-goal through node as computed by estimated cost function f = g + h

Completeness
Uniform cost: yes
Best first: no, infinite paths and oscillations
A* search: yes

Optimality
Uniform cost: yes, on condition that cost of successor node is greater than node
Best first: no, first node found
A* search: yes

Complexity (branching factor b and depth of solution d)
Uniform cost: time and space complexity both O(b^d)
Best first: O(b^m) where m is maximum depth of tree
A* search: exponential in length of the path

(b)
Heuristic 1: count the number of tiles out of place.
Heuristic 2: count distance out of place divided by 2

Heuristic needs to be admissible.
Needs to be admissible to ensure that h does not overestimate.

Heuristic 1: admissible because have to move at least this many tiles, but maybe more
Heuristic 2: admissible because have to make at least this many moves, and maybe more.

(c)
Let f* be cost of optimal node.

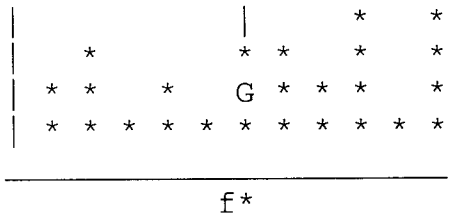
A* expands all nodes with f-cost less than f*.

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)

With the two heuristics given, the second is more informed.

(d)

- **making heuristics more accurate or efficient, i.e. more informed**
  - try to reduce the *effective branching factor b**
    - *b**: the branching factor that a uniform tree of depth *d* needs for *N* nodes

      $N = 1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$
    - the best heuristics will minimize *b** (search is a straight line if *b** is *1*...)
  - make *h(n)* as large as possible *without over-estimating*
    - e.g. by combining heuristics
    - but don't forget to consider the cost of computing the heuristic(s)...
    - ➢ **e.g. by inventing heuristics**
      - relax the problem
        - express logically and drop the conditions

2.
(a) 8 marks (Application)
(b) 8 marks (Application)
(c) 4 marks (Bookwork/Understanding)

(a)

```
sublist( L, SL ) :-
     append( Front, Back, L ),
     append( Mid, Rest, Back ).

monotonic( L ) :-
     L = [H|T],
     monotonic_ascending( H, T, a ).
monotonic( L ) :-
     L = [H|T],
     monotonic_descending( H, T, d ).

monotonic_ascending( _, [], a ).
monotonic_ascending( I, [H|T], a ) :-
     I < H,
     monotonic_ascending( H, T, a ).
monotonic_descending( _, [], d ).
monotonic_descending( I, [H|T], d ) :-
     I > H,
     monotonic_descending( H, T, d ).

length( [], 0 ).
length( [H|T], N1 ) :-
     length( T, N ),
     N1 is N + 1.

reverse( [], [] ).
reverse( [H|T], RevH ) :-
     reverse( T, Rev ),
     append( Rev, [H], RevH ).
```

(b)
Representation: list of 5 integers, 1 indicating smallest, 2 next smallest and so on.

Initial state: [1,5,2,3,4]

Goal state: [1,2,3,4,5]

```
State_change( flip, OldState, NewState ) :-
        Sublist( OldState, SL ),
        Monotonic( SL ),
        Length( SL, L ), L > 1,
        Append( Front, Back, OldState ),
        Append( SL, Rest, Back ),
        Reverse( SL, RevSL ),
        Append( RevSL, Rest, NewBack )
        Append( Front, NewBack, NewState ).
```

Note sublist will generate all sublists on backtracking. Could have used findall.

So the way this works:

Generate (for each) sublist SL
Is it montonic
Is it longer than one (flipping has no effect otherwise)
Break down oldstate
Oldstate = front ++ SL ++ Rest Back = SL ++ Rest
Reverse the sublist
Build Newstate
Newstate = front ++ RevSL ++ Rest Newback = RevSL ++ Rest

(c)
G' = <n0, Op>

This implicitly defines, inductively, an infinite set of paths, where

P0 = { <n0> }

Pi+1 = { p ++ <n> | p is a path in Pi, op is an operator in Op, n = op(frontier(p)) }

Where frontier(p) returns the frontier node, or last element, of the path p.

Then graph search algorithm, with breadth first, takes each member of Pi, computes the members of Pi+1, and so on; depth first search, takes one member of Pi, computes derived members of Pi+1, takes one element of Pi+1, computes derived members of Pi+2, and so on.

3.
(a) 10 marks (Bookwork, Application)
(b) 10 marks (Bookwork, Applciation)

(a)
data structure

- **information required about each grid location**
  - co-ordinate of current location in the robot's own (relative) system
  - what is to the 'north', 'south', 'east' and 'west' (robot's own system)
  - whether the robot has visited this location or not
  - the reward associated with this location

steps in procedure

- **adopt basic strategy**
  - e.g. random walk, follow left wall, ..., with some mechanism for loop checking
- **initialise** (starting grid location)
  - assign a coordinate value (say (0,0))
  - set reward to 0.0 and visited to true
  - perceive and record the result of performing action A (go up, down, left or right)
    - update pointers, set rewards to 0.0, visited to false, assign relative co-ordinate
- **explore**
  - choose next direction to go in and move one grid location in that direction
  - set visited for to 'true' for that grid location
  - perceive and record the result of performing action A (go up, down, left or right)
    - update pointers, set rewards to 0.0, visited to false, assign relative co-ordinate
  - repeat until exit location is found
- **assign** (reward or credit assignment)
  - set reward of exit location, when found, to 10
  - set reward of each visited grid location to 0.9 * maximum reward of any grid location that can be moved to by doing action A in that location
  - propagate values back until all visited grid locations are assigned a reward
    (with loop checking...)

assumptions

- it has no a priori knowledge about its environment (size and contents)
- it can move forward exactly one grid square in direction currently facing
- it can turn through exactly 90 degrees and knows which way it is facing

- four sensors detect if a wall is in front, behind, to its left, and/or to its right
- all movements and perceptions are carried out 100% accurately

Example
Could be anything

(b)
global map of environment

- **calculating potential fields**
  - constants required:
  - η, a scaling factor on the repulsive potential force at any point
  - ξ, a scaling factor on the attractive potential force at any point
  - r0, the distance of influence factor, beyond which an obstacle has no repulsive effect
- **procedure**
  - derive C-regions (one for each obstacle)
  - for each co-ordinate q,
    calculate the repulsive potential at q,
    by appying the following function to each C-region C:

$$U_{rep}(q) = 0.5\,\eta\,(1/r(q) - 1/r0)2, \quad \text{if } r(q) \leq r0$$
$$= 0, \quad \text{otherwise}$$

where $r(q)$ is the distance from q to the nearest coordinate in the C-region given by:
$r(q) = min\ ||q - q'||$ for all q' in C

moving to goal: use attractive potential

- **for a robot moving from a current location q to a goal location q_{goal}, calculate the next best location to move in by adding in the attractive potential of the goal location**
  - for each co-ordinate q' adjacent to q, calculate the attractive potential by:

$$U_{att}(q') = 0.5\xi r_{goal}(q')2$$

where $r_{goal}(q') = ||q' - q_{goal}||$ for all q' in C

adding in moving object
treat as local C-region

choosing next move:

- **the best co-ordinate to move to is given by:**
  - for each co-ordinate q' adjacent to q (the current location), sum the values obtained from the previous calculations:
    - $U_{sum}(q') = U_{rep}(q') + U_{att}(q') + U_{loc}(q')$
  - move to the co-ordinate q' with the lowest value of Usum

4.
(a) 6 marks (Application)
(b) 12 marks (Application)
(c) 2 marks (Understanding)

(a)

(Pr) $\neg ((p \rightarrow q) \leftrightarrow (\neg p \vee q))$

branch 1			*branch* 2		
1	$\neg(p \rightarrow q)$	*PB*	6	$p \rightarrow q$	*PB*
2	$\neg p \vee q$	*e*, *pr*, 1	7	$\neg(\neg p \vee q)$	*e*, *pr*, 6
3	p	*a*, 1	8	$\neg\neg p$	*a*, 7
4	$\neg q$	*a*, 1	9	$\neg q$	*a*, 7
5	q	*b*,2,3	10	p	$\neg\neg$, 8
close		4,5	11	q	*b*,6,10
			close		9,11

(Pr) $\neg ((p \vee q) \leftrightarrow \neg(\neg p \wedge \neg q))$

branch 1			*branch* 2		
1	$p \vee q$	*PB*	7	$\neg(p \vee q)$	*PB*
2	$\neg(\neg(\neg p \wedge \neg q))$	*e*, *pr*, 1	8	$\neg(\neg p \wedge \neg q)$	*e*, *pr*, 6
3	$\neg p \wedge \neg q$	$\neg\neg$, 2	9	$\neg p$	*a*, 7
4	$\neg p$	*a*, 3	10	$\neg q$	*a*, 7
5	$\neg q$	*a*, 3	11	$\neg\neg q$	*b*, 8, 10
6	q	*b*, 1, 4	12	q	$\neg\neg$, 10
close		5, 6	*close*		10,12

(Pr) $\neg ((p \wedge q) \rightarrow r) \leftrightarrow (p \rightarrow (q \rightarrow r))$

branch 1				*branch* 2			
1	$((p \wedge q) \rightarrow r)$	*PB*		11	$\neg((p \wedge q) \rightarrow r)$	*PB*	
2	$\neg(p \rightarrow (q \rightarrow r))$	*e*, *pr*, 1		12	$p \rightarrow (q \rightarrow r)$	*e*, *pr*	
3	p	*a*, 2		13	$p \wedge q$	*a*, 11	
4	$\neg(q \rightarrow r)$	*a*, 2		14	$\neg r$	*a*, 11	
5	q	*a*, 4		15	p	*a*, 13	
6	$\neg r$	*a*, 4		16	q	*a*, 14	
branch 1.1				17	$q \rightarrow r$	*b*, 12, 15	
7	$(p \wedge q)$	*PB*		18	r	*b*, 17, 16	
8	r	*b*, 1, 7		*close*		14, 18	
close		6, 8					
branch 1.2							
9	$\neg(p \wedge q)$	*PB*					
10	$\neg q$	*b*,3,9					
close		5, 10					

(b)
$\forall x,y.\ general(x) \wedge captain(y) \rightarrow order(x,y)$
$\forall x,y.\ captain(x) \wedge private(y) \rightarrow order(x,y)$
$\forall x,y,z.\ order(x,y) \wedge order(y,z) \rightarrow order(x,z)$
$general(chaos)$
$captain(scarlet)$
$private(benjamin)$

substituting $g = general$, $c = captain$, $p = private$, $o = order$

1 $\neg g(x) \vee \neg c(y) \vee o(x,y)$
2 $\neg c(x) \vee \neg p(y) \vee o(x,y)$
3 $\neg o(x,y) \vee \neg o(y,z) \vee o(x,z)$
4 $g(chaos)$
5 $c(scarlet)$
6 $p(benjamin)$

query $\neg o(chaos,benjamin)$

resolve with 3	$\neg o(chaos,y) \vee \neg o(y,benjamin)$	$[x = chaos, z = benjamin]$
resolve with 1	$\neg g(chaos) \vee \neg c(y1) \vee \neg o(y,benjamin)$	$[x = chaos, y = y1]$
resolve with 4	$\neg c(y1) \vee \neg o(y,benjamin)$	
resolve with 5	$\neg o(scarlet,benjamin)$	$[y = y1 = scarlet]$
resolve with 2	$\neg c(scarlet) \vee \neg p(benjamin)$	$[x = scarlet, y = benjamin]$
resolve with 5	$\neg p(benjamin)$	
resolve with 6	*contradiction*	

conclude $o(chaos,benjamin)$, *as required*

(c)
by solving the sub-goals in left to right order, and attempting to unify each sub-goal with each head of every rule.

For every rule that unifies, this is a 'state change': the proof procedure is a kind of search.

Algorithm is depth first and generate the alternatives on back-tracking.

5
(a) 4 marks, Bookwork
(b) 7 marks, Application/Understanding
(c) 6 marks, Application
(d) 3 marks, Application

(a)
Soundness, every proof is a theorem
Completeness, every theorem can be proved
Decidable, stops with yes or no
Efficient, coping with exponential complexity

(b)
e
$e \rightarrow c$
$e \rightarrow d$
$e \wedge d$

$p1$	e	
$p2$	$e \rightarrow c$	
$p3$	$e \rightarrow d$	
$\neg conc$	$\neg(c \wedge d)$	
1	c	$b, p1, p2$
2	d	$b, p1, p3$
3	$\neg d$	$b, 1, \neg conc$
$close$		2, 3

Using a resource twice when "in reality" can only use it once, the and is really an or

As in modal logics, add labels to the formulas, change the proof rules e.g new beta (b) rule, closure conditions check the labels are OK according to the logic

$\alpha : p \rightarrow q$
$\beta : p$

$f(\alpha, \beta) : q$

(c)
Let M = <W,R,P> with W = { α, β }, R = { (α, β) } and ||p|| = { α}

So $\models M,a\ p$ *and* $\models M,b\ p$

If we have axioms schema B then we also have $\models M,a\ \Box\Diamond p$

Since aRb then $\models M,b\ \Diamond p$

But $\models M,b \Diamond p$ *if and only if there is a world w st* bRw *and* $\models M,w\ p$

Since there is no such world w this is a contradiction.

if $\models M,a\ p$ *then we need to show* $\models M,a \Box\Diamond p$

Let b be any world such that aRb.

Then by symmetry bRa.

So by semantics of $\Diamond$ $\models M,b \Diamond p$

Since b was any world accessible from a, then it must be true of every world accessible from a

So by semantics of $\Box$, $\models M,a \Box\Diamond p$ *as required*

(d)

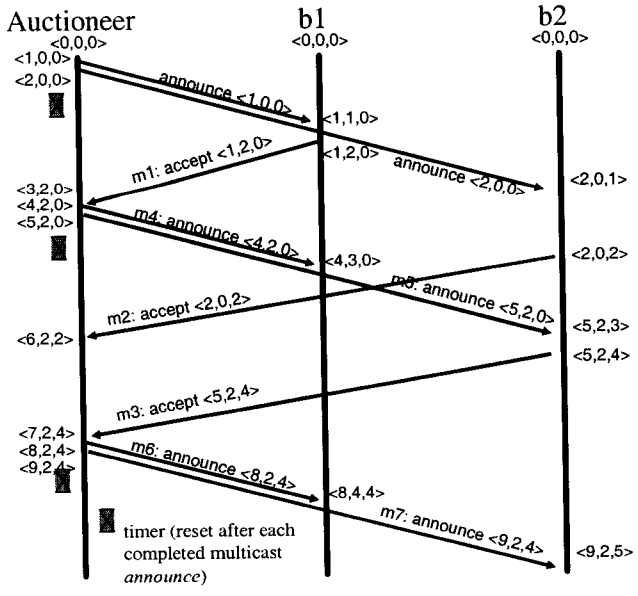
$\neg conc$	$1 : \neg(p \rightarrow \Box\Diamond p)$	
1	$1 : p$	$a, \neg conc$
2	$1 : \neg\Box\Diamond p$	$a, \neg conc$
3	$2 : \neg\Diamond p$	$\Diamond$ *rule*
4	$1 : \neg p$	$\Box$ *rule*
close		1,4

6.
(a) 6 marks, Bookwork
(b) 8 marks, Application
(c) 6 marks, Bookwork/Understanding

(a)

- distributed systems without global clocks
  - is that the ordering of event can only be determined locally
  - If an event e precedes an event f, then e *potentially caused* f.
- A *vector clock* is defined over an auctioneer agent A and a set of B buying agents with cardinality n
  - (n+1)-ary vector of natural numbers v = <A, b1, ..., bn>.
  - The starting clock of every agent is <0, 0, ..., 0>.
  - Each agent increments its entry in the vector when it performs a local event
  - It attaches its entire vector clock as a timestamp to every message it sends out.
  - When it receives a message, it takes the element-wise max of its vector clock and the timestamp on the incoming message, increments its entry in the vector by 1 (for the local event), and sets this value to be its new vector clock.
- One vector is defined as *later* than another if the value of at least one entry in the first vector is greater than the corresponding entry in the second, and no value in the second is greater than the first, i.e. v is later than u if and only if firstly $\forall i.0<=i<=n$: $ui <= vi$ and secondly $\exists i.0<=i<=n$: $ui < vi$. We write $v4u$ if v is later than u.

(b)
Message Sequence Chart



Problems
Auctioneer does not know which bids are in response to which announces
Bidders do not know which announces are in response to their bids

Rules for resolving message sequences via potential causality, with indication with reference to MSC

R1 For the auctioneer, if the timestamp on an incoming *accept* message is later than the auctioneer's vector clock after the last accepted bid (the last *accept* to cause an *announce*), then it was caused by the most recent multi-cast *announce*;

R2 For a bidding agent, if the timestamp on an incoming *announce* is later than their vector clock, then the *announce* was caused by its *accept*.

m1	120 ‣ 000	*accept* is caused by most recent (multi-cast) *announce*
m2	202 *not* ‣ 320	*accept* not caused by most recent (multi-cast) *announce*
m3	524 ‣ 320	*accept* is caused by most recent (broadcast) *announce*
m4	420 ‣ 120	(multi-cast) *announce* caused by *b*1's *accept* (m1)
m5	520 *not*‣ 202	(multi-cast) *announce* not caused by *b*2's *accept* (m2)
m6	824 *not* ‣ 430	(multi-cast) *announce* not caused by *b*1's *accept*
m7	924 ‣ 524	(multi-cast) *announce* caused by *b*2's *accept* (m3)

(c)
Use an agent communication language
Performative plus parameters
Performative includes bid and announce
One parameter is a conversation identifier
Using conversation identifiers:
Requirement here, each side attached unique integer id to message in a conversation.
Pair of identifiers is enough to uniquely identify a conversation (cf syn and ack bits in TCP)
Parameterize conversation identifier
with local vector clock timestamp