Three algorithms are explained

* A\*
* AO\*
* MinMax
* MinMax with alpha beta cutoff

1. **Algorithm A\***

A\* algorithm finds the optimum solution by considering total of two cost in the state space representation

g cost of moving from initial state to current state

h’ cost of moving from current state to goal state (an estimate)

hence total cost f’=g+h’

Two list are maintained OPEN and CLOSED

OPEN contains all the nodes that are yet to be processed

CLOSE contain all processed nodes

If multiple paths from node X to node Y is obtained then A\* retains the optimum path by comparing existing cost (in OPEN or CLOSE list) and new path.

1. Start with OPEN containing only initial node. Set that node’s g value to 0, its h’ value to whatever it is, and its f’ value to h’+0 or h’. Set CLOSED to empty list.

***Initialize*** *open = {s} closed = { } g(s)=0 f’(s)=h’(s)*

2. Until a goal node is found, repeat the following procedure:

(a) If there are no nodes on OPEN, report failure.

***fail***  *if open={ } terminate with failure*

(b) Otherwise pick the node on OPEN with the lowest f’ value.

Call it BESTNODE. Remove it from OPEN. Place it in CLOSED.

***Select***  *Select minimum cost state n from OPEN Save n in CLOSE*

(c ) See if the BESTNODE is a goal state. If so exit and report a solution.

***terminate*** *if n € G terminate with success and f’(n)*

(d) Otherwise, generate the successors of BESTNODE

For each of the SUCCESSOR, do the following:

***expand:***  *˅SUCC m of n*

*if m ~€ {open U close }*

*set g(m) = g(n),C(n,m)*

*set f’(m)=g(m)+h’(m)*

*Insert m in open*

*set the link to parent node*

*if m € {open U close }*

*set g(m)=min[g(m),g(n)+C(n,m)]*

*set f’(m)=g(m)+h’(m)*

*if f’(m) has decreased and m € {close }*

*move it to open*

*if f’(m) has decreased and m € {open}*

*update the cost and change the link to parent node*

1. **AO\* algorithm**

Problem Reduction Algorithm.

When problem is complex it is solved by breaking it into smaller parts and solving each part independently. Total solution is obtained by combining solution of each part.

Each sub problem(sub goal) should be independent and should not overlap

State space is represented by AND and OR arc; hence the name AO\*

AND arc OR arc

AND arc: To solve m; all r1,r2 and r3 sub goals must be solved

OR arc: To solve m any one of optimum sub goal r1 or r2 or r3 can be solved

* GRAPH consist of node representing initial state – call this state as INIT
  + *Let G={s}*
* Until INIT is labelled SOLVED or h’ > FUTILITY repeat the following
  + Trace the labelled arc from INIT and select for expansion one of unexpanded node from marked subtree and call it NODE n
  + Expand For every successor m of n
    - Add succ m to graph
    - If succ m is SOLVED , mark it as SOLVED
    - If not SOLVED compute h’(m)
  + Back propagation of the newly discovered information
    - Call cost revision(n)
* Cost Revision(n) :Let S be set of nodes that have been SOLVED or whose value has been changed. Repeat following steps till S is empty
  + Initialize S ={n}
  + Select from S a node m (select a node possibly whose descendent occurs in G).Remove m from S
  + If s={ } return
  + Compute the cost of each arc emerging from m
    - If m is AND node with succ r1, r2, r3 ........... rk
      * Set h’(m)= ∑ h(ri)+C(m,ri)
      * Mark each succ of m
      * If every succ is SOLVED , mark m as SOLVED
    - If m is OR node with succ r1, r2 , r3 ............. rk
      * Set h(m) = min( h(ri)+c(m,ri))
      * Mark the best succ
      * If best succ is SOLVED mark m as SOLVED
  + If cost of m has changed then change must be propagated , hence insert all ancestor of m in S for which m is marked successor
* Main feature of AO\*
  + Every time a new node is expanded, change in cost is propagated to all parent nodes
  + Current best path is always marked and maintained
  + If current path becomes expensive , while back propagating the cost, better cost path is chosen and it is marked as best current path
  + Problem is solved if all the sub goals contributing in solution is solved and start nodes is marked as solved

1. **Algorithm MinMax**

* Game playing algorithm
* Two players game …..Maximizer and Minimizer
* Cost of node is scaled between +10 and -10
* +10 defines winning score for maximize and -10 defines winning score for minimizer
* Maximizer will try to maximize its chance of winning and minimize opponent chance of winning. Minimizer will try to achieve same in its favour

Minmax (position, depth, player )

{

Step1:If DEEP-ENOUGH(*position,player*) then return structure

VALUE=STATIC(*position , player)*

PATH=nil

Step2:Else

{

SUCCESSORS=MOVE\_GEN(*position,player)*

Step3:if (SUCCESSOR= empty) return structure // same as DEEP\_ENOUGH

Step4: if (! Empty(SUCCESSORSvalue ))

{ BEST\_SCORE=min that STATIC can generate

For every SUCC of SUCCESSOR

4(a) { RESULT\_SUCC= MINMAX(SUCC, *depth+1,*OPPOSITE(player)

4(b) NEW-VALUE= - VALUE(RESULT\_SUCC)

4(c) if NEW-VALUE > BEST\_SCORE

(i) { BSET\_SCORE=NEW-VALUE

(ii) BEST-PATH= SUCC + PATH(RESULT\_SUCC)

*path from CURRENT to SUCC .*

}// end if

}// end for

Step5: value=BEST\_SCORE

path=BEST\_PATH

return structure

}// end if

}// else

}// minmax

**Minmax with alpha beta cut off**

Minmax \_A\_B(position, depth, player ,UT,PT)

{

Step1:If DEEP-ENOUGH(*position,player*) then return structure

VALUE=STATIC(*position , player)*

PATH=nil

Step2:Else

{

SUCCESSORS=MOVE\_GEN(*position,player)*

Step3:if (SUCCESSOR= empty) return structure // same as DEEP\_ENOUGH

Step4: if (! Empty(SUCCESSORSvalue ))

{ BEST\_SCORE=min that STATIC can generate

For every SUCC of SUCCESSOR

4(a) { RESULT\_SUCC= MINMAX\_A\_B(SUCC, *depth+1,*OPPOSITE(player),-PT,-UT)

4(b) NEW-VALUE= - VALUE(RESULT\_SUCC)

4(c) if NEW-VALUE > PT

(i) { PT=NEW-VALUE

(ii) BEST-PATH= SUCC + PATH(RESULT\_SUCC)

*path from CURRENT to SUCC .*

}// end if

(d) if PT >= UT then stop examining this branch and return value=PT path=BEST-PATH

}// end for

Step5: value=PT

path=BEST\_PATH

return structure

}// end if

}// else

}// minmax