Informed search: best-first search

informed: use a heuristic estimate of the distance from a node to a goal given by predicate eval/2

best-first: children of node are added according to heuristic (lowest value first)

Agenda is sorted

add_best(A,B,C): C contains the elements of A and B (B and C sorted according to eval/2)

search_best([Goal|RestAgenda],Goal):-
    goal(Goal).
search_best([CurrentNode|RestAgenda],Goal):-
    children(CurrentNode,Children),
    add_best(Children,RestAgenda,NewAgenda),
    search_best(NewAgenda,Goal).
add_best([],Agenda,Agenda).
add_best([Node|Nodes],Agenda,NewAgenda):-
    insert(Node,Agenda,TmpAgenda),
    add_best(Nodes,TmpAgenda,NewAgenda).
insert(Node,Agenda,NewAgenda):-
    eval(Node,Value),
    insert(Value,Node,Agenda,NewAgenda).
insert(Value,Node,[]],[Node]).
insert(Value,Node,[FirstNode|RestOfAgenda],[Node,FirstNode|RestOfAgenda]):-
    eval(FirstNode, FirstNodeValue),
    Value < FirstNodeValue.
insert(Value,Node,[FirstNode|RestOfAgenda],[FirstNode|NewRestOfAgenda]):-
    eval(FirstNode, FirstNodeValue),
    Value >= FirstNodeValue,
    insert(Value,Node,RestOfAgenda,NewRestOfAgenda).
Informed search:  
*best-first search on a puzzle*

A tile may be moved to the empty spot if there are at most 2 tiles between it and the empty spot.

Find a series of moves that bring all the black tiles to the right of all the white tiles.

Cost of a move: 1 if no tiles were in between, otherwise amount of tiles jumped over.
Informed search: best-first search on a puzzle - encoding

Board: 

get_tile(Position,N,Tile) :-
   get_tile(Position,1,N,Tile).
get_tile([Tile|Tiles],N,N,Tile).
get_tile([Tile|Tiles],N0,N,FoundTile) :-
   N1 is N0+1,
   get_tile(Tiles, N1, N, FoundTile).

replace([Tile|Tiles],1,ReplacementTile,[ReplacementTile|Tiles]).
replace([Tile|Tiles],N,ReplacementTile,[Tile|RestOfTiles]) :-
   N>1,
   N1 is N-1,
   replace(Tiles,N1,ReplacementTile,RestOfTiles).

Moves:

start_move(move(noparent, [b,b,b,e,w,w,w],0))

Agenda items:

move_value(Move, Value)

heuristic evaluation of position reached by Move
Informed search:
best-first search on a puzzle - algorithm

```
tiles(ListOfPositions, TotalCost):-
  start_move(StartMove),
  eval(StartMove, Value),
  tiles([move_value(StartMove, Value)], FinalMove, [], VisitedMoves),
  order_moves(FinalMove, VisitedMoves, [], ListOfPositions, 0, TotalCost).

finds sorted list of children with their evaluation
```
Informed search:
best-first search on a puzzle - encoding’

next_move(move(Position,LastPosition,LastCost),
move(LastPosition,NewPosition,Cost)) :-
  get_tile(LastPosition, Ne, e),
  get_tile(LastPosition, Nbw, BW),
  not(BW=e),
  Diff is abs(Ne-Nbw),
  Diff<4,
  replace(LastPosition,Ne,BW,IntermediatePosition),
  replace(IntermediatePosition,Nbw,e,NewPosition),
  (Diff=1 -> Cost=1
  ; otherwise -> Cost is Diff-1).

NewPosition is reached in one move from LastPosition with cost Cost

goal(Move):-
  eval(Move,0).

eval(move(OldPosition,Position,C),Value):-
  bLeftOfw(Position,Value).

bLeftOfw(Pos,Val):-
  findall((Nb,Nw),
  (get_tile(Pos,Nb,b),get_tile(Pos,Nw,w), Nb<Nw),L),
  length(L,Val).

sum of the number of black tiles to the left of each white tile
Informed search: best-first search on a puzzle - auxiliaries

order_moves(FinalMove, VisitedMoves, Positions, FinalPositions, TotalCost, FinalTotalCost):
  FinalPositions = Positions + connecting sequence of target positions from VisitedMoves ending in
                   FinalMove’s target position.
  FinalTotalCost = TotalCost + total cost of moves added to Positions to obtain FinalPositions.

order_moves(move(noparent, StartPosition, 0),
            VisitedMoves, Positions,
            [StartPosition|Positions], TotalCost, TotalCost).

order_moves(move(FromPosition, ToPosition, Cost),
            VisitedMoves, Positions,
            FinalPositions, TotalCost, FinalTotalCost):-
  element(PreviousMove, VisitedMoves),
  PreviousMove = move(PreviousPosition, FromPosition, CostOfPreviousMove),
  NewTotalCost is TotalCost + Cost,
  order_moves(PreviousMove, VisitedMoves,
              [ToPosition|Positions], FinalPositions, NewTotalCost, FinalTotalCost).
Informed search: best-first search on a puzzle - example run

?- tiles(M, C).
[b,b,b,e,w,w,w] - 9
[b,b,b,w,e,w,w] - 9
[b,b,e,w,b,w,w] - 8
[b,b,w,w,b,e,w] - 7
[b,b,w,w,b,w,e] - 7
[b,b,w,w,e,w,b] - 6
[b,e,w,w,b,w,b] - 4
[b,w,e,w,b,w,b] - 4
[e,w,b,w,b,w,b] - 3
[w,w,b,e,b,w,b] - 2
[w,w,b,w,b,e,b] - 1

M = [[b,b,b,e,w,w,w], [b,b,b,w,e,w,w], [b,b,e,w,b,w,w], [b,b,w,w,b,e,w], [b,b,w,w,b,w,e], [b,b,w,w,e,w,b], [b,e,w,w,b,w,b], [b,w,e,w,b,w,b], [e,w,b,w,b,w,b], [w,w,b,e,b,w,b], [w,w,b,w,b,e,b], [w,w,e,w,b,b,b]]

C = 15
Informed search: optimal best search

Best-first search is not complete by itself: a heuristic might consistently assign lower values to the nodes on an infinite path.

An A algorithm is a complete best-first search algorithm that aims at minimizing the total cost along a path from start to goal.

\[ f(n) = g(n) + h(n) \]

actual cost so far: adds breadth-first flavor
estimate on further cost to reach goal: if optimistic (underestimating the cost), an optimal path will always be found. Such an algorithm is called A*.

\[ h(n)=0 : \text{degenerates to breadth-first} \]