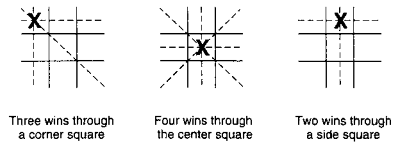
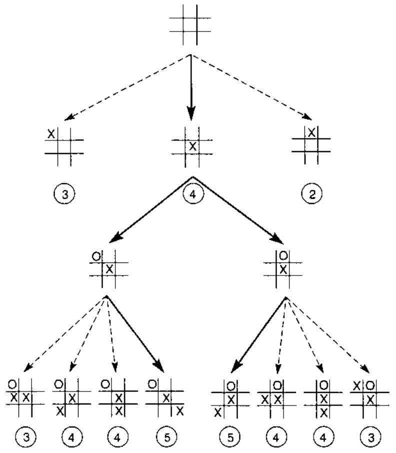
Heuristic Search

- A heuristic is a rule for choosing a branch in a state space search that will most likely lead to a problem solution
- Heuristics are used when
  - there is no exact solution to a problem, as in medical diagnosis
  - there is an exact solution but the computation is prohibitively expensive, as in the game of chess
- Heuristics are fallible
  - they may find suboptimal solutions
  - they may find no solution at all
- Heuristics are problem dependent and there may be many alternative heuristics for the same problem

Tic-Tac-Toe

- Without considering symmetry the search space is 9!; using symmetry the search space is 12 * 7!


Three wins through a corner square
Four wins through the center square
Two wins through a side square



- A simple heuristic is the number of solution paths still open when there are 8 total paths (3 rows, 3 columns, 2 diagonals)

Here is the search space using this heuristic

The total search space is now reduced to about 40, depending on the opponents play

Another Heuristic

- Try to devise any alternative heuristic for tic-tac-toe that is either entirely different than the heuristic on the previous page or that uses that heuristic plus some other measurements (such as blocking the opponent)

Hill Climbing

- The basic steps are
  - expand the current node and evaluate the children using your heuristic
  - select the child with the best heuristic value and discard the other children
- Although this algorithm is simple to implement, it has many severe problems
  - the algorithm may halt with the parent is better than all the children and where it halts may not be a solution
  - since a history is not maintained it cannot recover from this failure
  - this approach is susceptible to getting stuck in a local maxima

Best-first Search

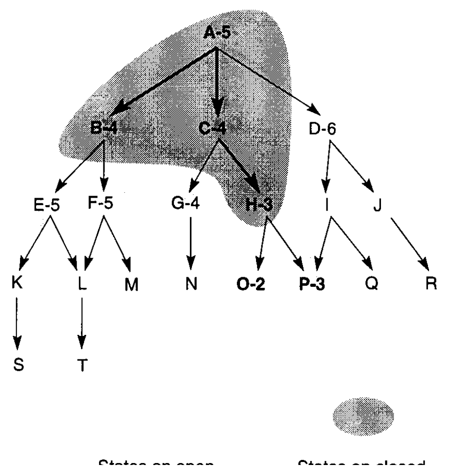
```
procedure best_first_search;

begin
  open := [Start];                                                    % initialize
  closed := [ ];
  while open ≠ [ ] do                                                 % states remain
    begin
      remove the leftmost state from open, call it X;
      if X = goal then return the path from Start to X
      else begin
            generate children of X;
            for each child of X do
            case
               the child is not on open or closed:
                  begin
                        assign the child a heuristic value;
                        add the child to open
                  end;
               the child is already on open:
                  if the child was reached by a shorter path
                  then give the state on open the shorter path
               the child is already on closed:
                  if the child was reached by a shorter path then
                     begin
                        remove the state from closed;
                        add the child to open
                     end;
            end;                                                      % case
            put X on closed;
            re-order states on open by heuristic merit (best leftmost)
         end;
return failure                                                        % open in empty
end.
```

- Since the heuristic value changes as the current state changes, what happens if a node is revisited and now has a lower value
  - if still on the open list, reduce the value
  - if on the closed list, assign the new value and put it back on the open list (may be useful when the exact form of the goal is not known)

Example search space

- Here is a hypothetical search space

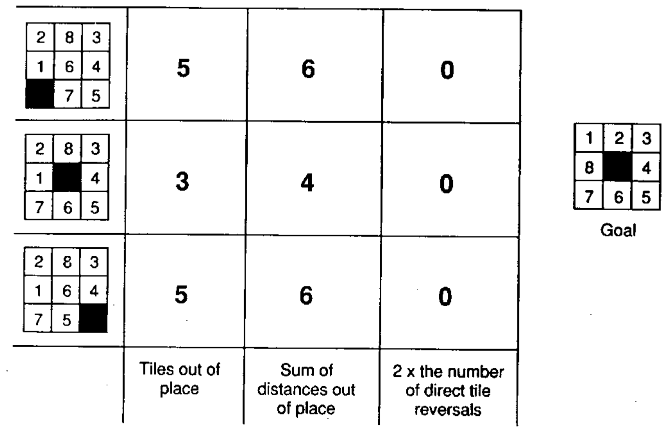


- Notice the heuristic is not perfect; O-2 is not a goal but has a lower value that P-3 which is a value

1. open = [A5]; closed = [ ]
2. evaluate A5; open = [B4,C4,D6]; closed = [A5]
3. evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]
4. evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]
5. evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]
6. evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!

Heuristics for the Eight Puzzle

- Three possible heuristics
  - count # tiles out of place
  - sum of the distances out of place
  - 2 times the number of direct reversals



	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals
2 8 3 / 1 6 4 / ■ 7 5	5	6	0
2 8 3 / 1 ■ 4 / 7 6 5	3	4	0
2 8 3 / 1 6 4 / 7 5 ■	5	6	0

- Reversals only works well if you are near a goal; a combined heuristic (sum of distances and reversals) might work better

Applying Heuristics

- Use the heuristic of adding the number of tiles out of place to two times the number of direct reversals
- Start with

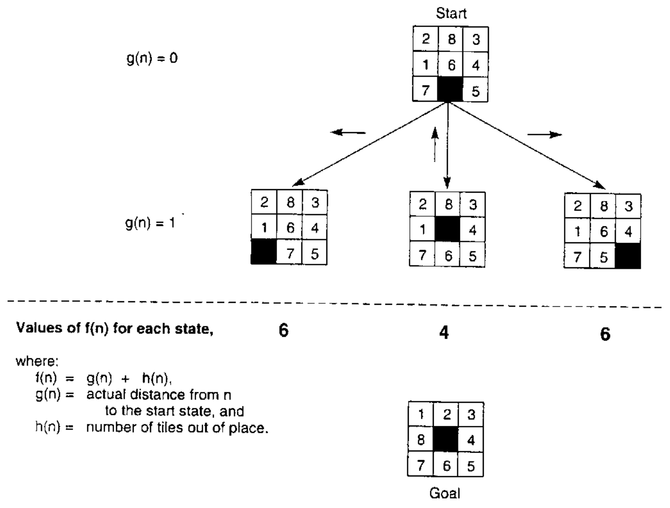
2	8	3
1	6	4
	7	5

and apply this heuristic relative to the goal shown below; find the next five moves

1	2	3
8		4
7	6	5

Avoiding the “garden path”

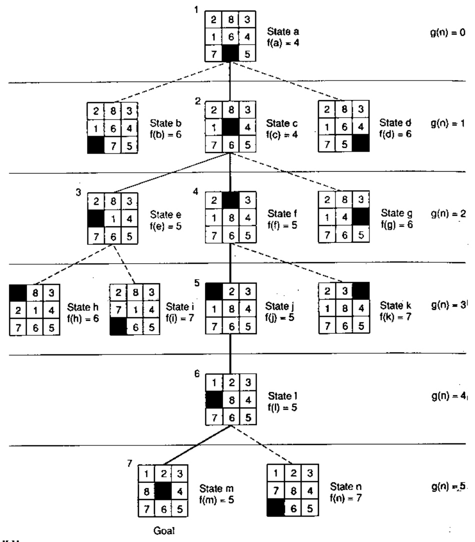
- We need to penalize long searches so we include path depth as another factor f(n) = g(n) + h(n) where g(n) is the path length and h(n) is the heuristic value
- Path length starts at 0 and is incremented by 1 for each move



Values of f(n) for each state, 6 4 6

where:
f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.

A heuristic search



1. open = [a4];
   closed = [ ]
2. open = [c4, b6, d6];
   closed = [a4]
3. open = [e5, f5, g6, b6, d6];
   closed = [a4, c4]
4. open = [f5, h6, g6, b6, d6, l7];
   closed = [a4, c4, e5]
5. open = [j5, h6, g6, b6, d6, k7, l7];
   closed = [a4, c4, e5, f5]
6. open = [l5, h6, g6, b6, d6, k7, l7];
   closed = [a4, c4, e5, f5, j5]
7. open = [m5, h6, g6, b6, d6, n7, k7, l7];
   closed = [a4, c4, e5, f5, j5, l5]
8. success, m = goal!

- The search space is drastically reduced in size by the evaluation function

Applying Another Heuristics

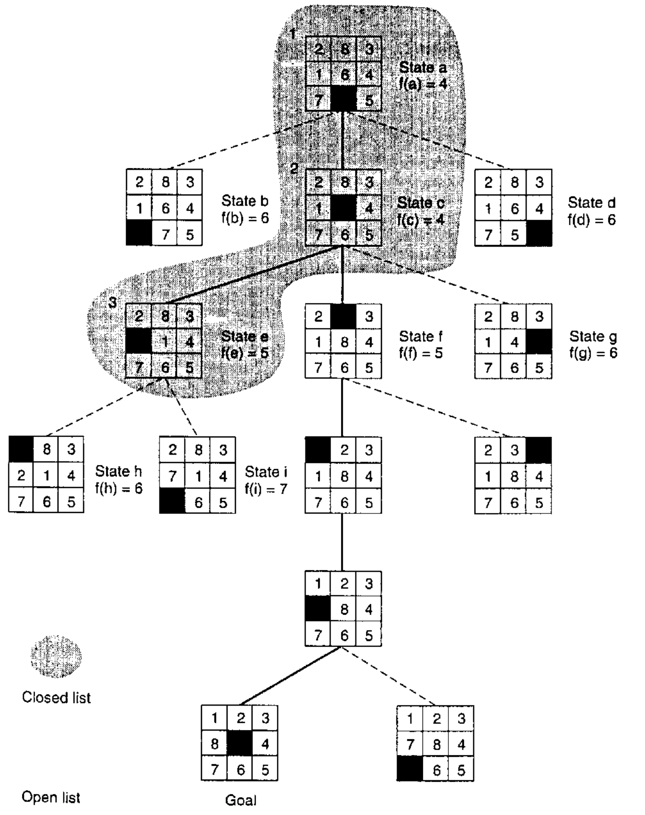
- Use the heuristic of adding the sum of distances out of place to the path length
- Start with

2	8	3
1	6	4
	7	5

and apply this heuristic relative to the goal shown below; find the next five moves

1	2	3
8		4
7	6	5

Open and closed lists

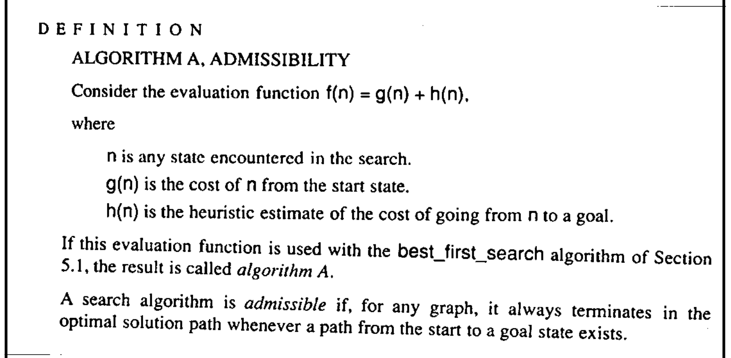


The basic approach

- When a state is examined all of its children are generated
- Any children already visited (on the open or closed lists) are discarded
- The value of f(n) is the sum of the heuristic function and the length of the search path
- The set of open states is sorted by the values for f(n)
- The algorithm can be more efficient by choosing appropriate data structures for the open and closed lists

Admissibility

- An algorithm is admissible if it is guaranteed to find a minimal path to a solution (assuming one exists)
- Breadth-first search is admissible



DEFINITION

ALGORITHM A, ADMISSIBILITY

Consider the evaluation function $f(n) = g(n) + h(n)$,

where

n is any state encountered in the search.

$g(n)$ is the cost of n from the start state.

$h(n)$ is the heuristic estimate of the cost of going from n to a goal.

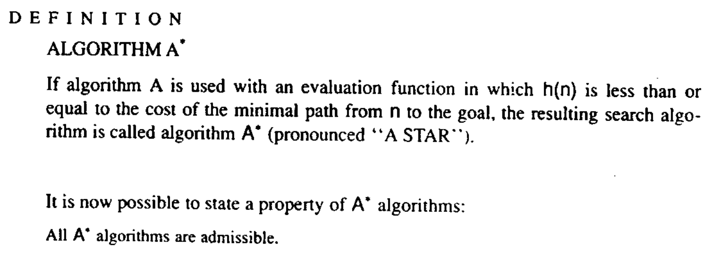
If this evaluation function is used with the best_first_search algorithm of Section 5.1, the result is called *algorithm A*.

A search algorithm is *admissible* if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

- Suppose g*(n) is the cost of the shortest path from the start node to n
- Suppose the h*(n) is the actual cost of the shortest path from node n to the goal
- if f*(n) = g*(n) + h*(n) then a best first search using f* is admissible

A* Algorithms

- In general, g(n) > g*(n)
- If h(n) ≤ h*(n) for all n, then any evaluation function f using h(n) and best first search will result in an A* algorithm



DEFINITION

ALGORITHM A*

If algorithm A is used with an evaluation function in which h(n) is less than or equal to the cost of the minimal path from n to the goal, the resulting search algorithm is called algorithm A* (pronounced "A STAR").

It is now possible to state a property of A* algorithms:

All A* algorithms are admissible.

- Breadth first search is an A* algorithm where h(n) = 0
- Examples with the eight puzzle
  - # tiles out of place ≤ # moves to the goal
  - sum of direct distances ≤ # moves to the goal

Is it A* ?

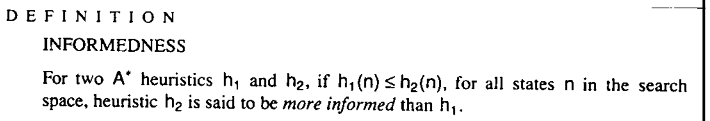
- Consider the eight puzzle
- Is the heuristics of counting the number of tiles out of place A* ?
- Is the heuristics of counting the sum of direct distances A* ?

A Blocks World Problem

- Suppose a blocks world has problems of the form "stack block X on block Y". Give a heuristic that might solve such problems.

- Is it admissible?

Informedness


DEFINITION

INFORMEDNESS

For two A* heuristics h_1 and h_2, if $h_1(n) \le h_2(n)$, for all states n in the search space, heuristic h_2 is said to be *more informed* than h_1.

- Three heuristics for the eight puzzle
  - $h_1(n) = 0$ for all states, this is a breadth first search; it is admissible but it has the problem of maintaining too many open states
  - $h_2(n)$ is the number of tiles out of place, which is admissible
  - $0 = h_1(n) \le h_2(n) \le h^*(n)$ so we say h_2 is more informed than h_1
  - being more informed means you will search fewer states to find an optimal solution

A Third Heuristic

- Let $h_3(n)$ be the sum of the distances out of place
- How does h_3 compare with h_2 and h_1 on the previous page? Which heuristic is most informed?

Comparison of three searches

- Breadth first searches every state shown
- Using the number of tiles out of place is in gray; 14 states are searched
- Optimal searches only 6 states