Uninformed Search

Punjab University College of Information Technology, Lahore

Blind Search Algorithms

- Backtrack
- Depth First
- Breadth First
- Depth First with iterative deepening

Backtracking

- Backtracking is a technique for systematically trying all paths through the state space.
- Backtracking search begins at the start state and pursues a path until is reaches either a goal or a dead end.
- If it finds goal, it quits and returns the solution
- If it reaches the dead end, it backtracks to the most recent node on the path having unexamined siblings.

Backtracking

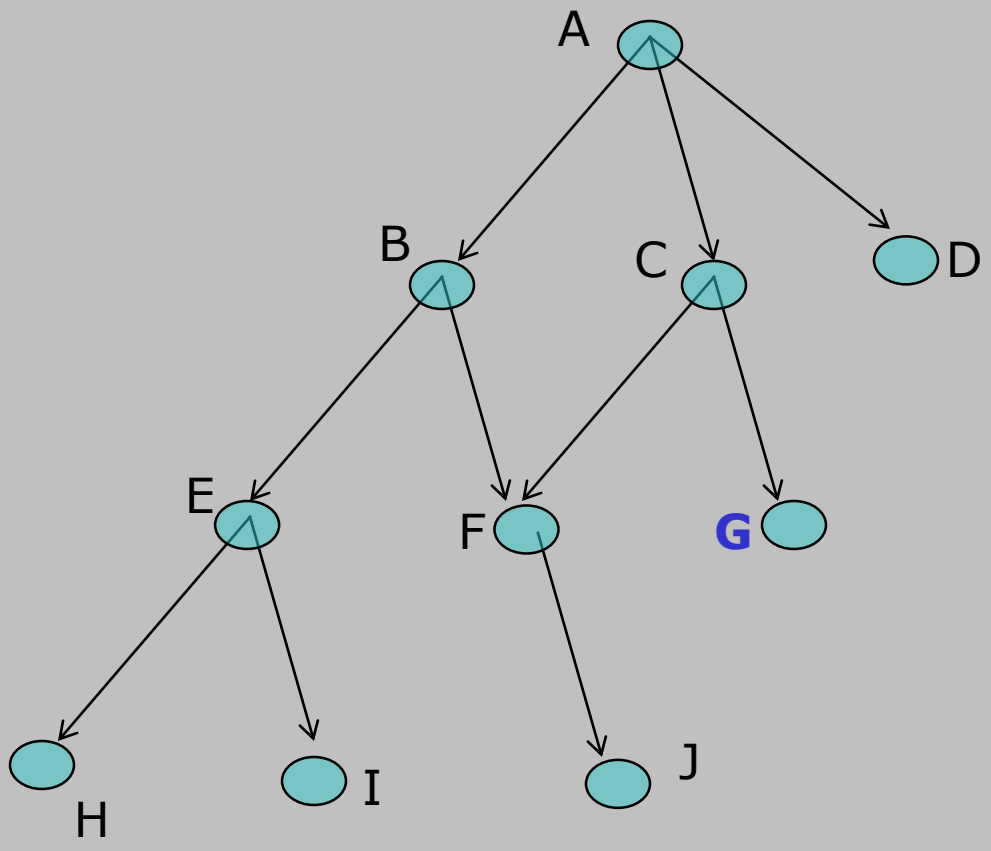
Three Lists that keep track of the algorithm:

- SL
  - for state list (in the current path being tried). if goal is found, SL contains the ordered list on the solution path.
- NSL
  - For new state list (contains nodes awaiting evaluation).
- DE
  - For dead ends (lists states whose descendants have failed to contain a goal node).
- CS
  - For current state (contains node currently under consideration)

Backtrack Algorithm

```
begin
  SL := [Start], NSL:= [Start], DE:= [], CS:= Start
  while NSL≠ [] do
       begin
          if CS = goal
           then return SL
          if CS has no child nodes (excluding those already on DE, SL, and NSL)
           then begin
            while SL is not empty and CS = first element of SL do
              begin
                add CS to DE
                remove first element from SL
                remove first element from NSL
                CS:= first element of NSL
              end
            add CS to SL
          end
          else begin
            place children of CS (except nodes already on DE, SL, or NSL) on NSL
            CS:= first element of NSL
            add CS to SL
          end
       end
       return FAIL
  end
```

Backtracking Example



Backtracking Example

Iteration#	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[BA]	[BCDA]	[ ]
2	E	[EBA]	[EFBCDA]	[ ]
3	H	[HEBA]	[HIEFBCDA]	[ ]
4	I	[IEBA]	[IEFBCDA]	[H]
5	F	[FBA]	[FBCDA]	[EIH]
6	J	[JFBA]	[JFBCDA]	[EIH]
7	C	[CA]	[CDA]	[BFJEIH]
8	G	[GCA]	[GCDA]	[BFJEIH]

Breadth First Search Algorithm

- Breadth first search explores the space in a level by level fashion.
- Only when there are no more states to be explored at a given level does the algorithm move on the next level.
- “Open” and “Closed” lists are used to keep track of the search. Open is Like NSL having list of nodes whose children have not been examined. Closed records states that have already been examined. Closed is the union of DE and SL.

BFS Example

#	Open	Closed
1	[A]	[ ]
2	[BCD]	[A]
3	[CDEF]	[BA]
4	[DEFG]	[CBA]
5	[EFG]	[DBA]
6	….	…

Breadth First Search

Analysis

- Suppose Branching Factor is "b"
- Search tree generates "b" nodes at Level-1 and b^2 at Level-2 and . . . b^d at Level d.
- Here we can see the exponential complexity bound like $O(b^d)$
- The memory requirements are a bigger problem for breadth first search.

Breadth First

Depth	Nodes	Time	Memory
0	1	1 ms	100 bytes
2	111	1 second	11 Kb
4	11111	11 seconds	1 MB
6	10^6	18 min	111MB
8	10^8	31 hrs	11GB
10	10^{10}	128 days	1 TB

Depth First Search

- Depth first is a simplification of back track algorithm
- The descendent states are added and removed from the left end of OPEN.
- OPEN is a LIFO structure. The organization of open as a stack directs search towards the most recently generated states

Depth First Example

#	Open	Closed
1	[A]	[ ]
2	[BCD]	[A]
3	[EFCD]	[BA]
4	[HIFCD]	[EBA]
5	[IFCD]	[HEBA]
6	[FCD]	[IHEBA]
7	[JCD]	[FIHEBA]
8	[CD]	[JFIHEBA]
9	….	….

Depth-First with iterative Deepening

- The depth bound forces a failure on a search path.
- This causes a breadth–like sweep of the search at that depth level.
- This algorithm increases the bound level in each iteration.
- Because the algorithm searches the space in level–by–level fashion it is guaranteed to find a shortest path to a goal.

Conclusion

- Breadth First
  - Because it always examines all the nodes in level by level fashion, it always finds the shortest path to the goal
  - In bad branching factor (states with high average number of descendants) the combinatorial explosion may prevent the algorithm from finding a solution using the available space.
  - The space utilization is measured in terms of the number of open states on OPEN is an exponential function of the length of the path at any time.

Conclusion

- Depth First
  - Gets quickly into a deep search space. If it is known that the solution will be long, depth first search will not waste time finding it.
  - It can get lost deep in the graph missing shorter paths to the goal
  - It might even get stuck in an infinitely long path that does not lead to a goal.

Heuristics

- Heuristics may be defined in a variety of ways:
  - The study of the methods and rules of discovery and invention (Polya,1945)
- Heuristics may be viewed as:
  - Rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution
- Heuristics are loosely described as rules of thumb.

Heuristics

- Two basic situations for the use of Heuristics:
  - An inherently ambiguous problem, e.g. medical diagnosis system
  - The cost of finding a solution by exhaustive methods like depth first search, is prohibitive
- A heuristic is fallible: it is only an informed guess of next step to be taken in solving a problem
- A heuristic can lead a search algorithm to a sub-optimal solution or fail to find any solution

Tic-Tac-Toe Problem

- Consider the example of state space
  - 9 nodes at the first level, 8 nodes of each node at level–2 =9x8x7 . . 2 (9!=326880) nodes to search
- Need to reduce the number of nodes or the size of the state space
- Analyze carefully the rules of playing Tic-Tac-Toe or view as expert to setup heuristics

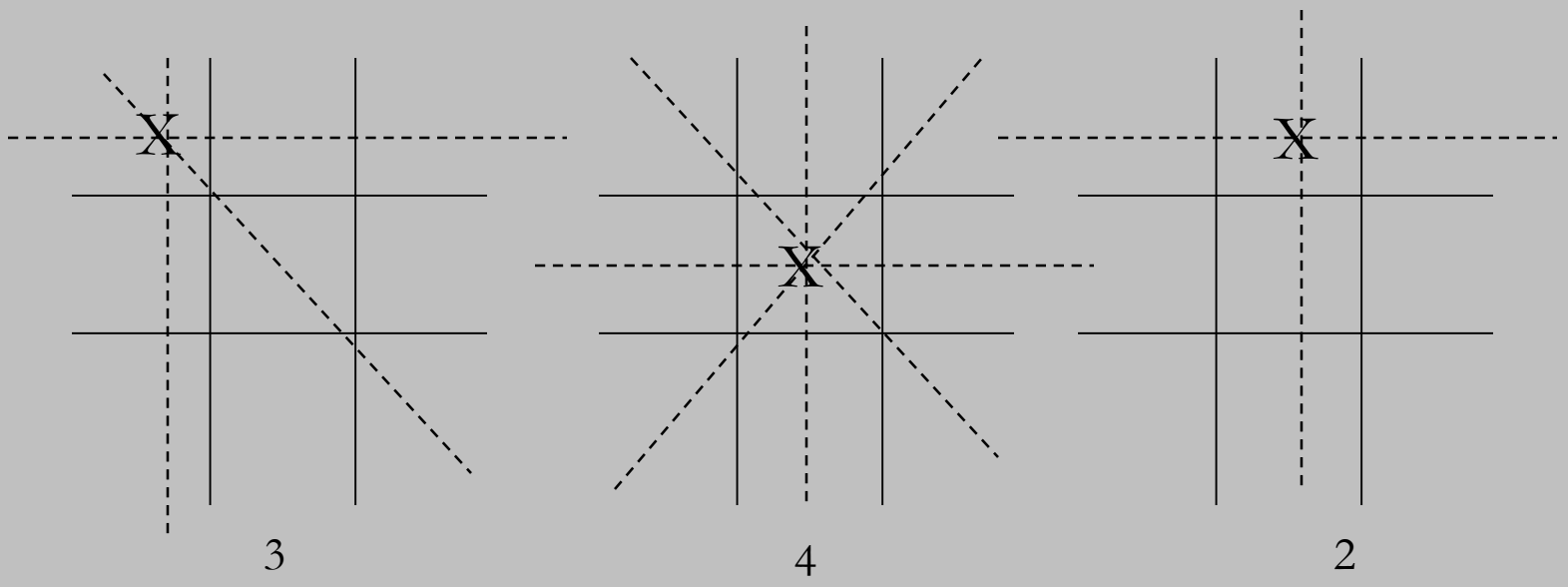
State Space Reduced by Symmetry!

Tic-Tac-Toe problem solving using Heuristic

- A simple heuristic can eliminate search entirely:
  - We may move to the board in which “X” has the most winning lines.
- In case of states with equal numbers of potential wins, take the first such state found.
- In case “X” takes the center of the grid, we can eliminate other alternates along with all of its descendants to 2/3 of the space.

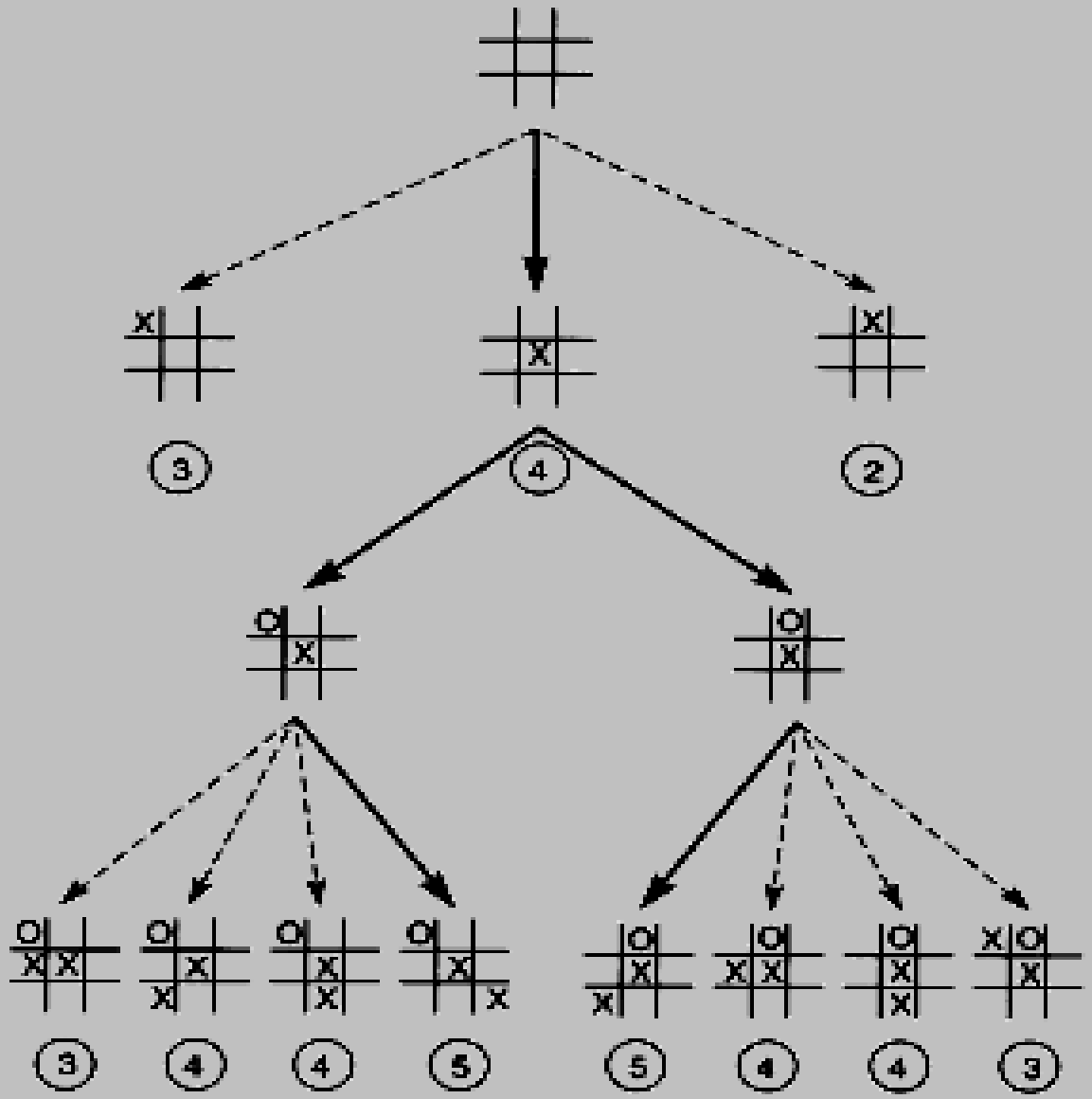
Tic-Tac-Toe problem solving using Heuristic

Heuristics for State Space for TIC-TAC-TOE

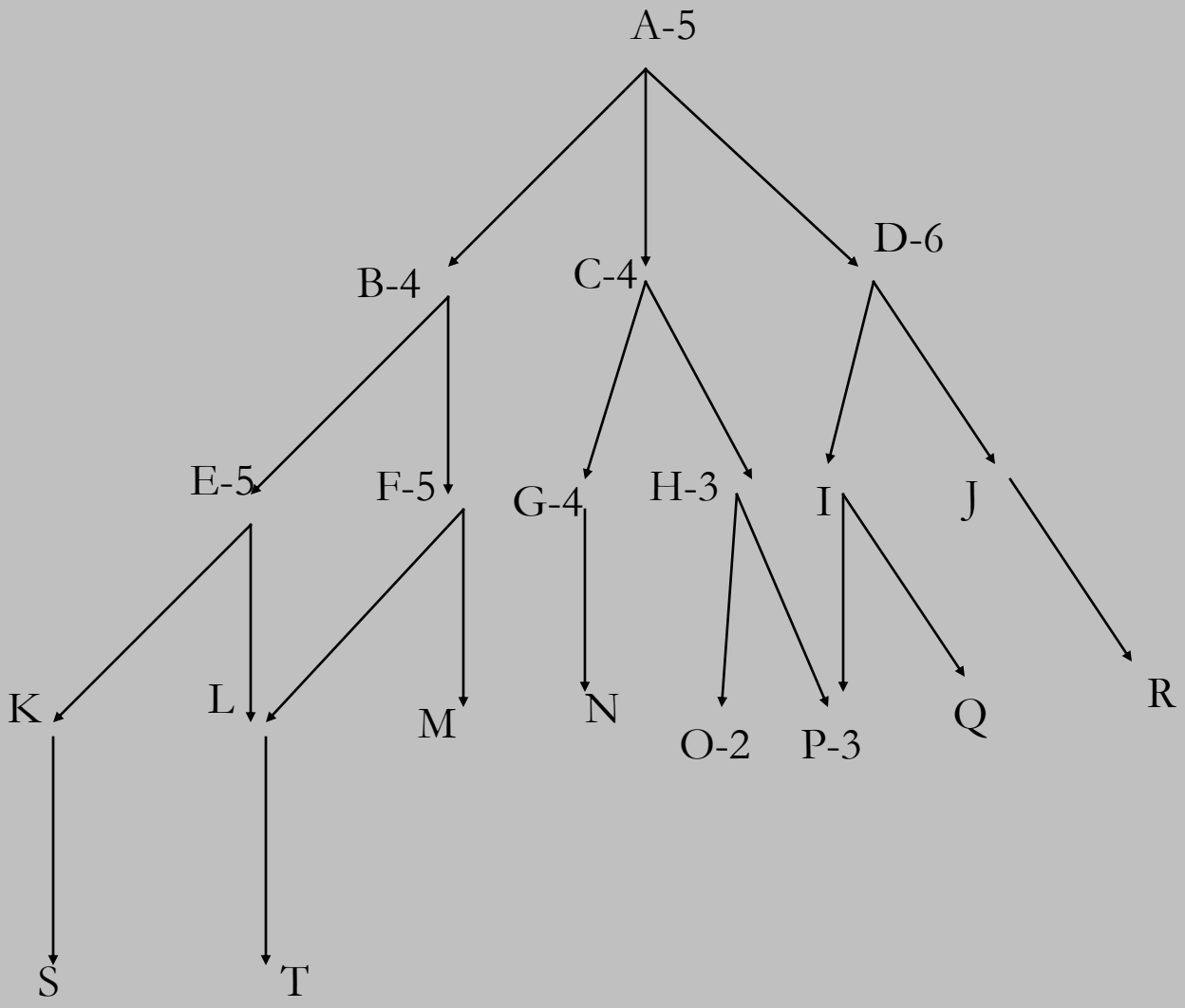


Dashed lines shows the number of potential wins at level-1

Tic-Tac-Toe using Heuristic



Best-First Search



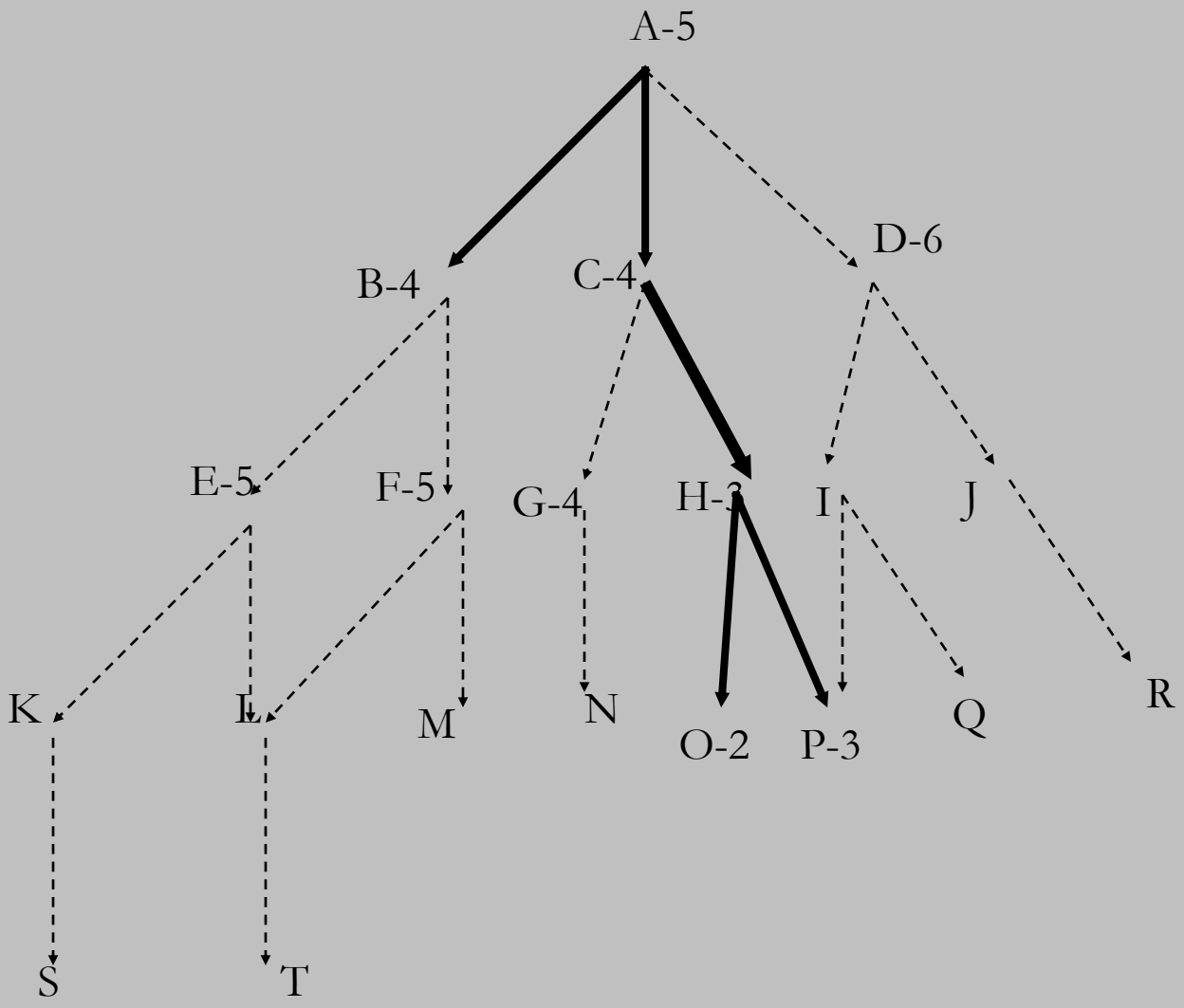
Best-First Search

- Depth first Algorithm uses LIFO style list to implement search
- Breadth First Search uses FIFO style list to search
- Best-first is an heuristic search that uses a priority queue to order the states on the OPEN

Best-First Search

Evaluate	Opened	Closed
	[A5]	[ ]
A5	[B4,C4,D6]	[A5]
B4	[C4,E5,F5,D6]	[B4,A5]
C4	[H3,G4,E5,F5,D6]	[C4,B4,A5]
H3	[O2,P3,G4,E5,F5,D6]	[H3,C4,B4,A5]
O2	[P3,G4,E5,F5,D6]	[O2,H3,C4,B4,A5]
P3	The solution	

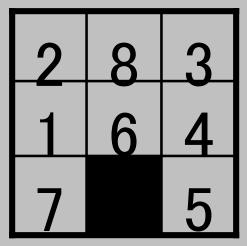
Best-First Search



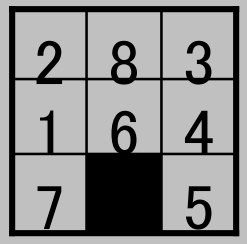
Heuristic Measures for 8 Puzzle Problem

- The simple heuristic count the tiles out place in each state when it is compared with the goal.
- Sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state

Heuristic Measures for 8 Puzzle Problem

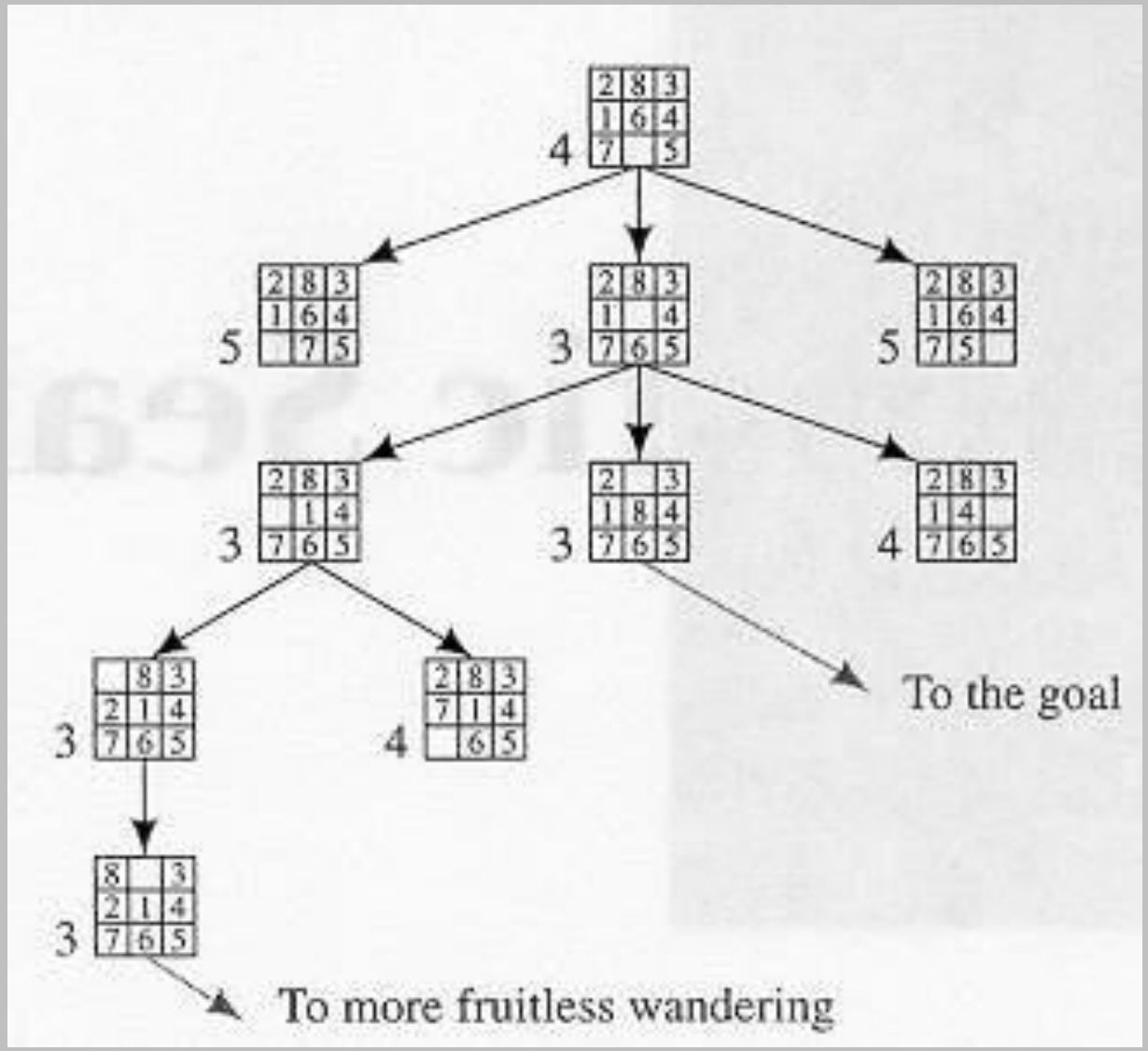


h(n) = No. of Tiles out of place = 4



h(n) = Sum of distances out of place = 5

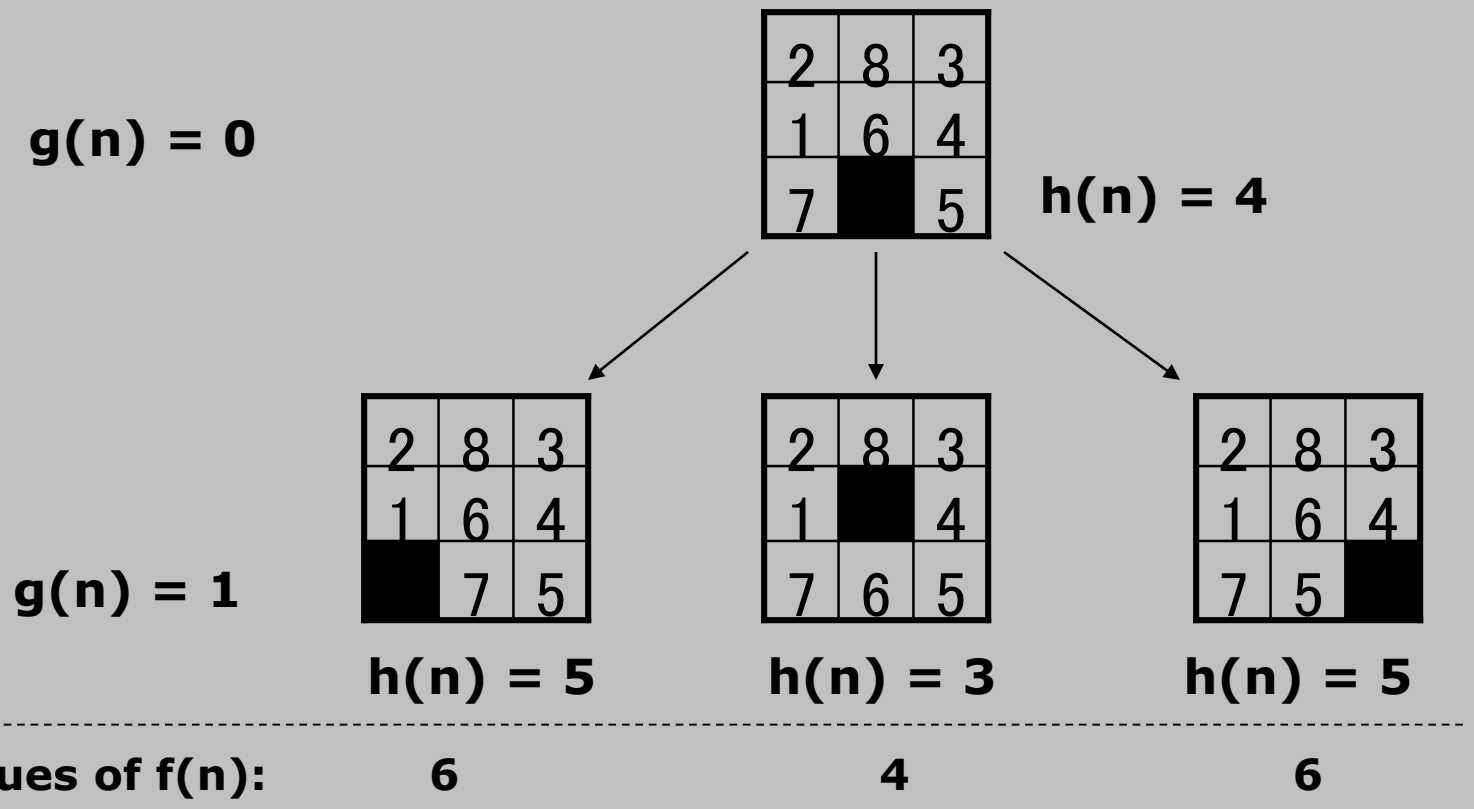
A Possible Result of a Heuristic Search Procedure



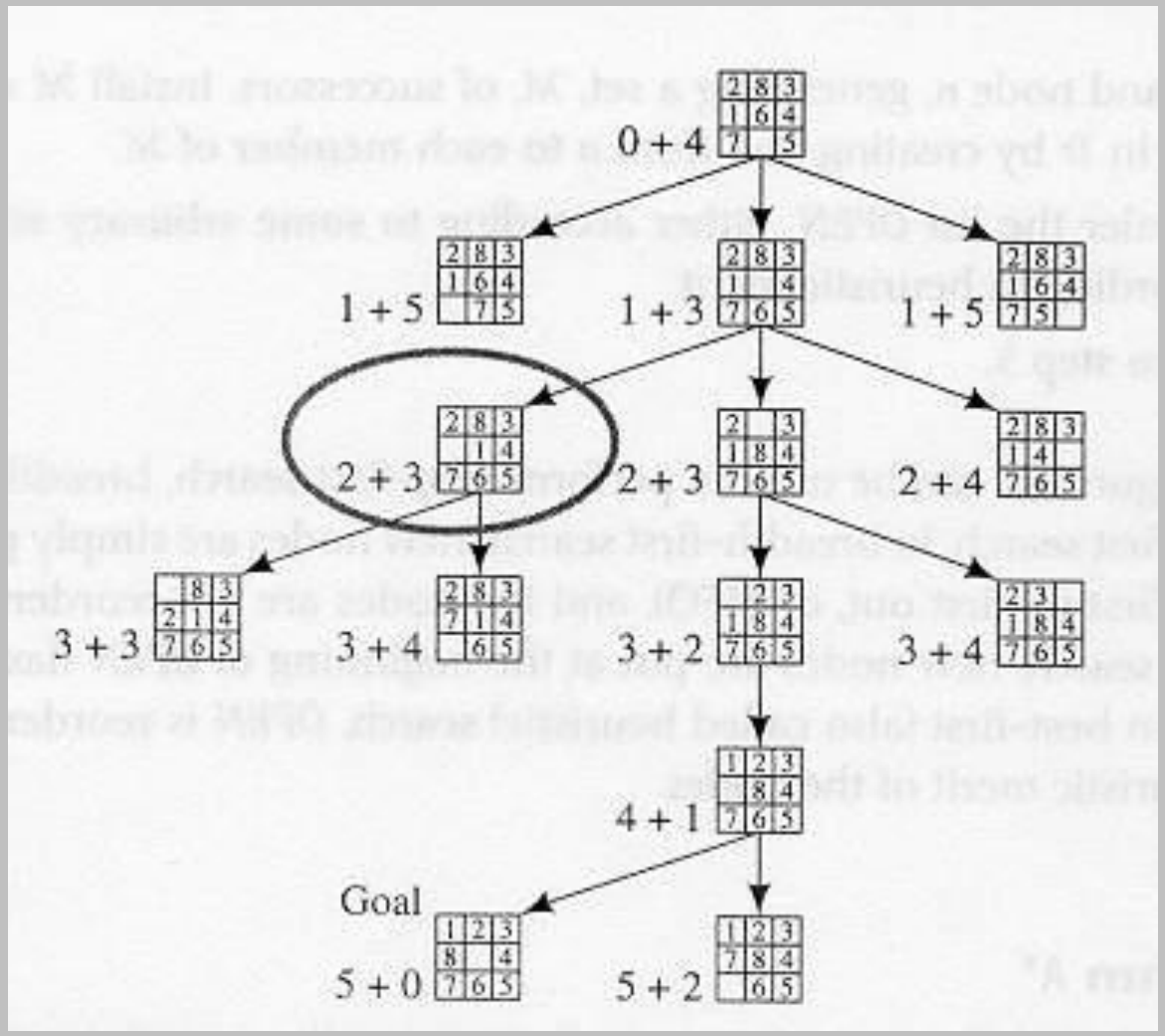
Heuristic Evaluation Function

- If two states have same or nearly same heuristic evaluations, it is generally more preferable to examine the state that is nearest to the root state
- This aspect is maintained in g(n) to give the heuristic function the following form:
  - f(n) = g(n) + h(n)
  - Where g(n) measures the actual length of the path from any state ‘n’ to the start state

Heuristic Evaluation Function



Heuristic Search using the evaluation function



Characteristics of Heuristics

We can evaluate the behavior of heuristics on the basis of following characteristics:

- Admissible:(gives shortest path)
- Informed:(has more accurate information for next step to be taken)
- Monotone:(min-cost to find shortest path)

Admissibility

- A search algorithm is admissible if, for any graph, it always terminates in the optimal solution path.
- If the ‘h’ function never overestimates the cost to reach the goal, then ‘h’ is called an admissible heuristic
- Admissible functions are optimistic, as they always think the cost to the goal is less than it actually is!

Informedness

- For two heuristics ‘h1’ & ‘h2’, if

  $$h1(n) < h2(n)$$

  for all states ‘n’ in the search space, ‘h2’ is said to be more informed than ‘h1’
- A ‘more’ informed heuristic expands less states than a relatively ‘less’ informed heuristic

Monotonicity

- A heuristic function ‘h’ is monotonic if
  - For all states n_i and n_j where n_j is descendent of n_i

    $h(n_i) - h(n_j) \leq \text{cost}(n_i, n_j)$

    where cost (n_i, n_j) is the actual cost of going from state n_i to n_j.
- The heuristic evaluation of the goal state is zero or h(goal)=0.
- A monotonic heuristic consistently finds the minimal path to each state it encounters in the search.

Heuristic Search Algorithms

- Heuristic Search Algorithms
  - Best-First Search
    - Maintains a priority queue to switch to heuristically promising paths
  - A* Search
    - Uses algorithm A (best-first search + evaluation function f(n)) with a constraint that h(n) never overestimates h*(n) -- the actual cost to reach the goal from any state ‘n’.
  - Hill-Climbing
    - Often called greedy search algorithm, does not maintain any information to backtrack to other paths in the state space if it adopts a fruitless “opportunist” path.