Introduction to Search

- Search is the operational task that characterizes the AI programs best. Almost every AI program depends on a search procedure to perform its prescribed functions.
- Problems are typically defined in terms of states and a solution corresponds to a goal state.
- Solving a problem then amounts to searching through different states until one or more goal states are found.

e.g., chess: 20 alternative moves for each board configuration

>100 different configurations

Therefore > 20^{100} moves !!! *Is it possible to evaluate all?*

eliminate questionable states, look ahead

Approaches to AI

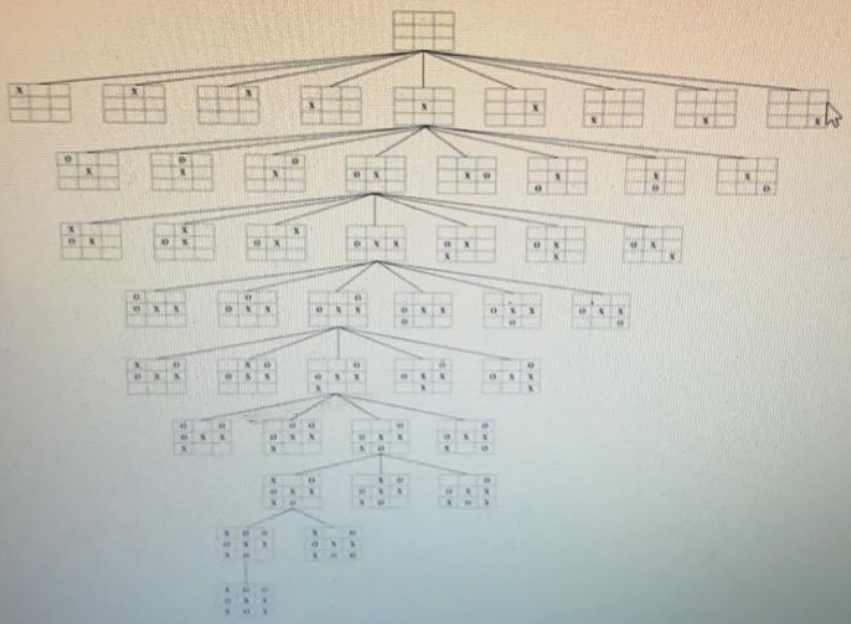
- Pattern recognition: recognition/classification ability
- Image processing/ computer vision: vision aspect
- Natural language processing/speech recognition: communication aspect
- Machine learning: generalization
- Knowledge based systems: information storage/ reasoning
- **Search strategies: reasoning/decision making**
- Evolutionary computing: mimics natural evolution
- Neural networks: emulation of human nervous system
- Fuzzy logic: human reasoning process/uncertainty handling
- Expert systems: mimic experts
- Robotics
- Artificial life

State space graph

- Keeps track of effects of several alternative sequences of actions; represents all possible states and actions

- A state space is a directed graph (V, E); V: a set of nodes; E: a set of arcs
  - assumed solutions : nodes of a tree
  - initial state: root of the tree
  - goal state: leaf/ leaves of the tree

- Each arc corresponds to the instance of one of the operators; may have a positive cost associated with it corresponding to the cost of the operator


{Winner - X}

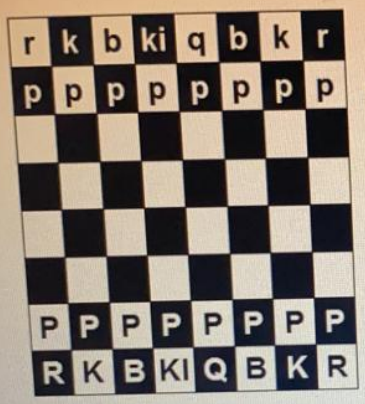
State space graph

- Node generation is performed by computing the identification/ representation code of children nodes from a parent node. Once this is done a child is said to be generated.
- A node is explored if some of its successors are generated.
- The process of generating all of the children of a parent is known as expanding the node.

State space graph

- A search procedure is a strategy for selecting the order in which nodes are generated and a given path is selected
- A solution is a sequence of operators that is associated with a path in a state space from a start node to a goal node
- State space search is a process of searching through a state space by making explicit a sufficient portion of an implicit state space graph to include a goal node
- Initially, V = {S}, when S is expanded, its successors are generated and those nodes are added to V and associated arcs are added to E. This process continues until a goal node is found.

‘State Space’

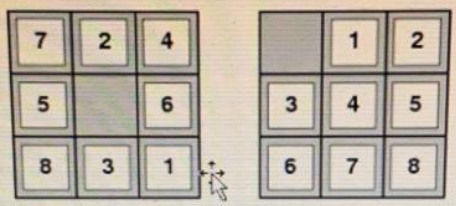


Initial state

operators

Goal state(s)

Example: The 8-puzzle

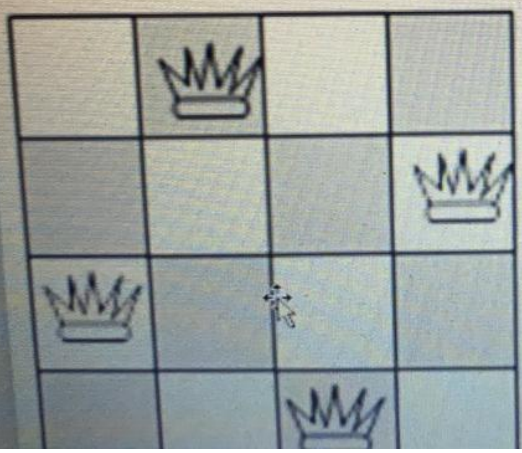


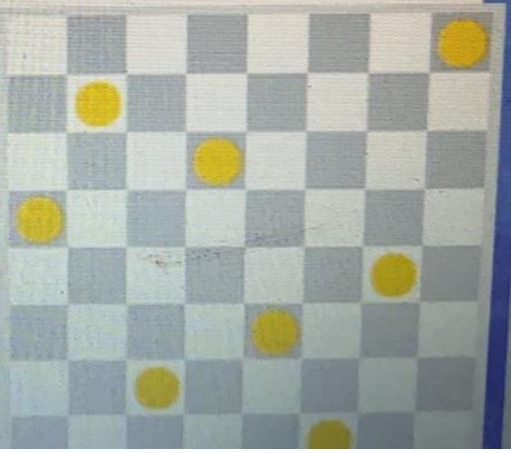
Start State Goal State

- states?
- actions?
- goal test?
- path cost?

Click to add notes

Example: *n* queens
(n = 4, n = 8)





Formal Search Problem

4-tuple <X,S,G,δ>

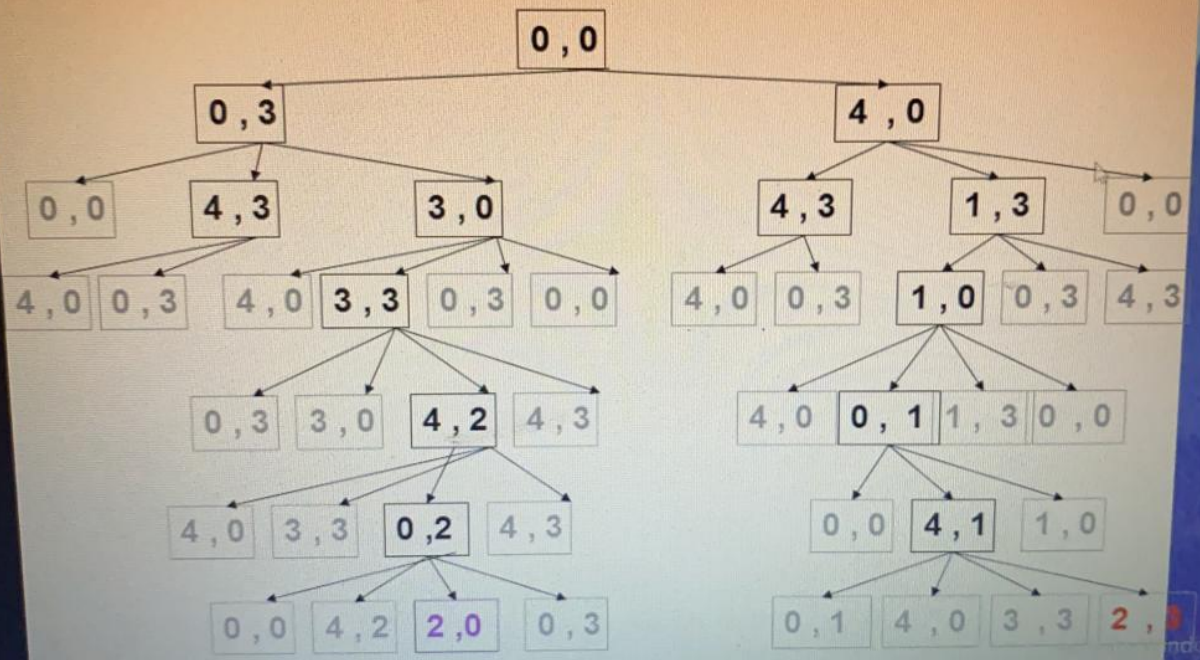
X: set of states

S: start state

G: goal state

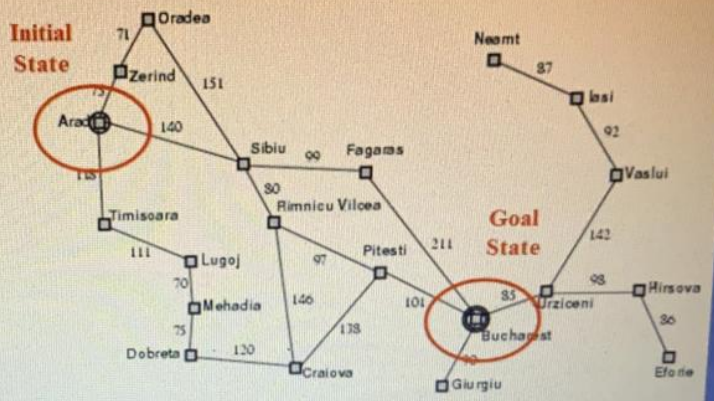
δ: state transition function

The Water Jugs Problem – Search Tree



Path Finding Problem

- **Formulate goal:**
  - be in Bucharest (Romania)
  -
- **Formulate problem:**
  - **action**: drive between pair of connected cities (direct road)
  -
  - **state**: be in a city (20 world states)
- **Find solution:**
  - sequence of cities leading from start to goal state, e.g., Arad, Sibiu, Fagaras, Bucharest
  -
- **Execution**
  - drive from Arad to Bucharest according to the solution



Environment: fully observable (map), deterministic, and the agent knows effects of each action. Is this really the case?

Note: Map is somewhat of a "toy" example. Our real interest: *Exponentially large spaces*, with e.g. 10^100 or more states. Far beyond full search. Humans can often still handle those!
One of the mysteries of cognition

Example: The Water Jugs Problem

- 2 jugs
  - 4 gallon
  - 3 gallon

  - How can you get exactly 2 gallons into the 4 gallon jug?

- Possible operators:
  1. Empty jug
  2. Fill jug from tap
  3. Pour contents from one jug into another

General Graph Searching Algorithm

1. *Create a search Tree Tr, consisting solely of the start node n_o on an ordered list called OPEN.*
2. *Create a list called CLOSED that is initially empty*
3. *If OPEN is empty, exit with failure*
4. *Select the first node on OPEN, put it in CLOSED. Call this node n*
5. *If n is goal node, exit successfully with the solution by tracing a path backward along the arcs in Tr from n to n_o (Arcs are created in Step 6)*
6. *Expand node n, generating a set M of successors. Install M as successor of n in Tr by creating arcs from n to each member of M. Successor will be added to OPEN.*
7. *Reorder the list OPEN either according to some arbitrary scheme or according to heuristic merit.*
8. *Goto Step 3*

General Graph Searching Algorithm

1. *Create a search Tree Tr, consisting solely of the start node n_o on an ordered list called OPEN.*
2. *Create a list called CLOSED that is initially empty*
3. *If OPEN is empty, exit with failure*
4. *Select the first node on OPEN, put it in CLOSED. Call this node n*
5. *If n is goal node, exit successfully with the solution by tracing a path backward along the arcs in Tr from n to n_o (Arcs are created in Step 6)*
6. *Expand node n, generating a set M of successors. Install M as successor of n in Tr by creating arcs from n to each member of M. Successor will be added to OPEN.*
7. *Reorder the list OPEN either according to some arbitrary scheme or according to heuristic merit.*
8. *Goto Step 3*

Evaluating Search Strategies

- Time complexity – How much time is required to find the solution. (#nodes expanded)
- Space complexity – How much space is required to find the solution (maximum size of 'node-list' during search)
- Completeness – Is the algorithm guaranteed to find the solution, if one exists?
- Optimality – Does it find an optimal solution (if more than one solutions are present)?

Click to add title

How to execute BFS, DFS, Best first Search then???

What do OPEN and CLOSED signify?

Can we execute other search algorithm also using this framework?

Types of search

- Blind/uninformed – no information about goodness of successor state (exhaustive search)

Breadth first search (BFS), depth first search (DFS), depth limited search (DLS), iterative deepening search (IDS), iterative broadening search (IBS), bidirectional breadth first search (BiBFS), island-driven search

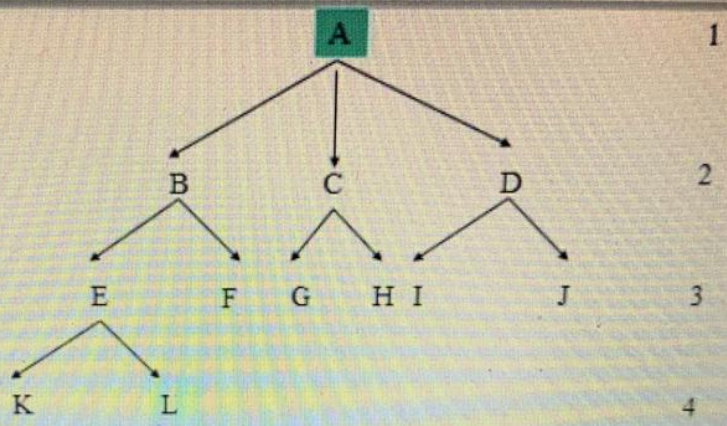
- Uniform cost search (UCS)
- Informed – problem specific information to help focus the search (heuristic search)

Greedy search, A search, iterative deepening A* (IDA*)*

Uninformed search

- BFS :
  - ✓ Expands all nodes at depth i before expanding those at depth i+1
  - ✓ Complete, optimal, time ~ $O(b^d)$, space ~ $O(b^d)$ (b-branching factor, d depth of the tree)

- DFS :
  - ✓ Generates successors of a node just one at a time; follow one branch, pursue as deep as possible before trying alternatives.
  - ✓ If at dead end, back-up and expand the deepest nodes visited (prune repeated states)
  - ✓ Not complete, not optimal, time ~$O(b^d)$, space ~ O(bd)



A: start state; L: goal state

BFS: order: A B C D E F G H I J K L; path: A B E L

DFS: order: A B E K L; path: A B E L

DLS: order: A B E F C G H D I J (considering depth bound as 3)
no goal found within this bound

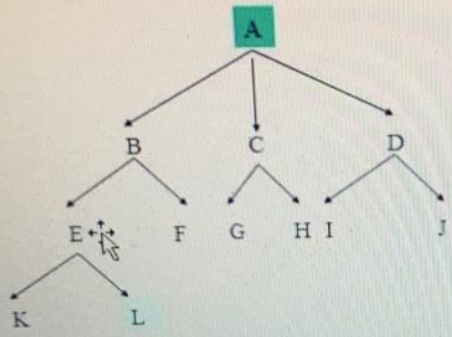
Click to add title

- Prove 3 is int
- We know 0 is an int
- Succ(int) = int
- Pred(int) = int
- 3
- 4 2
- 5 3 3 1
- 0

Uninformed Search (contd.)

- DLS :
  - to prevent search process running away towards nodes of unbounded depth from start node, a depth bound (say, l) is used; no successor is generated whose depth is greater than depth bound
  - complete, not optimal, time ~ $O(b^l)$, space ~ O(bl)
  - assumes that a goal node will be found out within depth l

Example



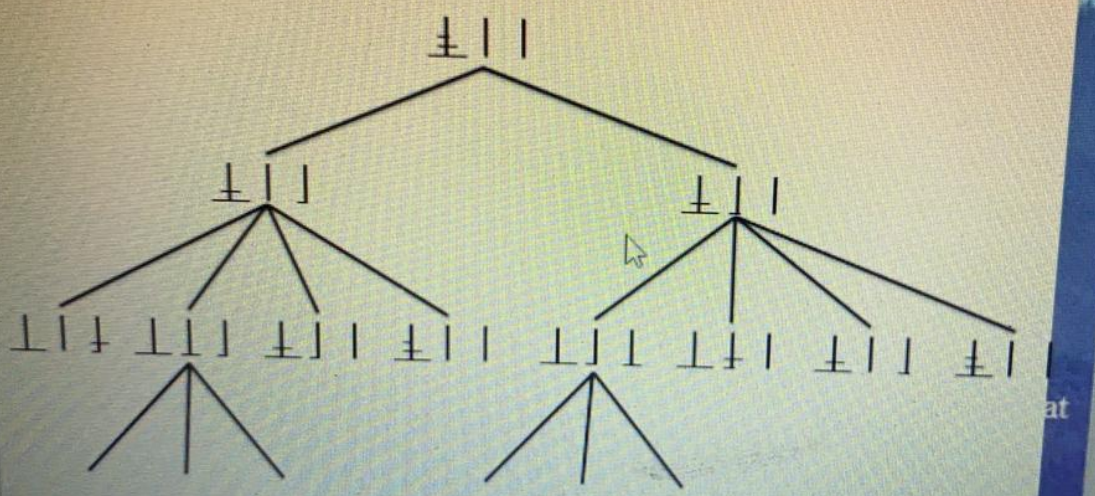
A: start state; L: goal state

BFS: order: A B C D E F G H I J K L; path: A B E L

DFS: order: A B E K L; path: A B E L

DLS: order: A B E F C G H D I J (considering depth bound as 3)
no goal found within this bound

The Towers of Hanoi: BFS

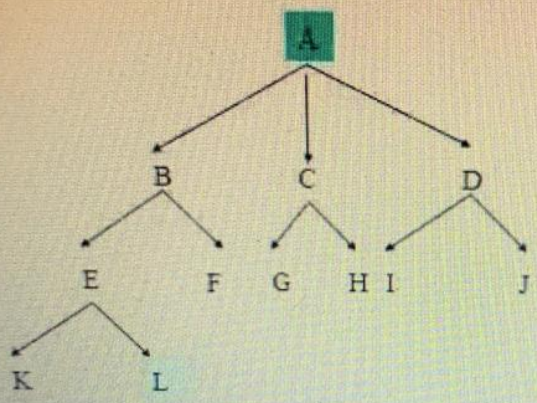


Breadth-First vs. Depth-First search

- Depth-first
  - requires less memory (O(bd))
  - may find a solution without searching much of the search space
- Breadth-first
  - will not get trapped exploring a blind alley
  - guaranteed to find solution (if one exists)
  - will find minimal solution (if more than one exist)

bfs is computationally expensive than dfs by a factor of (1+1/b)

Example



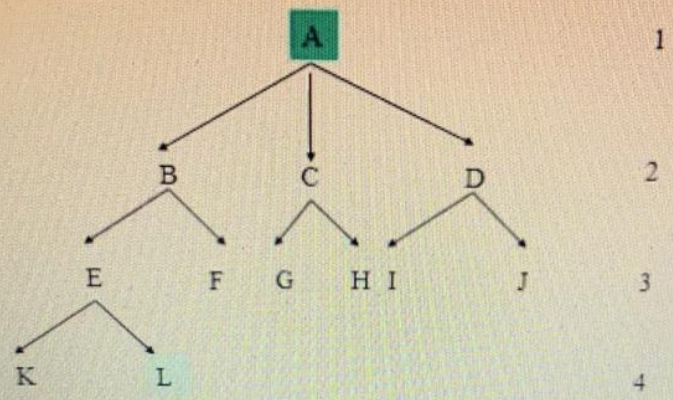
- A: start state; L: goal state

IBS:

order: iteration 1: A
iteration 2: A B E K
iteration 3: A B E K L

path: A B E L

Example



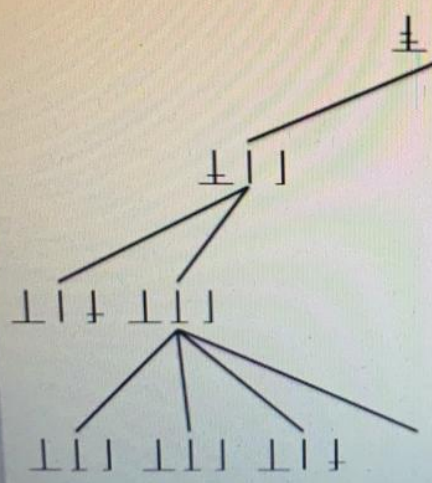
- A: start state; L: goal state

IDS:

order: iteration 1: A
iteration 2: A B C D
iteration 3: A B E F C G H D I J
iteration 4: A B E K L

path: A B E L

The Towers of Hanoi: DFS



Text Symbols Media

Click to add title

- Bfs: $(b^{(d+1)} + b^{d+b-3}) / (2(b-1))$

Uninformed search (contd.)

IDS :

- Successive DFS are conducted, each with depth bound increasing by 1 until a goal node is found
- Complete, optimal, time ~ $O(b^d)$, space ~ $O(bd)$
- Used if search space is deep
- Optimal blind search procedure

IBS :

- Performs DFS on subgraphs of breadth1, breadth2, … and so on until a goal node is found
- Complete, not optimal, time ~ $O(c^d)$, space ~ $O(c^d)$ c: breadth
- Used if search space is wide, many goals

ID Time Complexity

- At depth j, total nodes = (b^(j+1) -1)/(b-1)
- This goes from j = 0 to j = d-1

..... = (b^(d+1) –bd – b +d)/ (b-1)^2

(b^(d+2) + b^(d+1) +(b^2)d + b^2 – 4bd-5b+3d+2)/(2*(b-1)^2)

(b^(d+2) + b^(d+1))/(2*(b-1)^2)

ID: (b-1)*b^(d-1)/(2*(b-1)^2)

Dfs : b^(d+1)/ 2*(b-1)

ID dfs = (b-1)/(b-1)

Uninformed search (contd.)

BiBFS

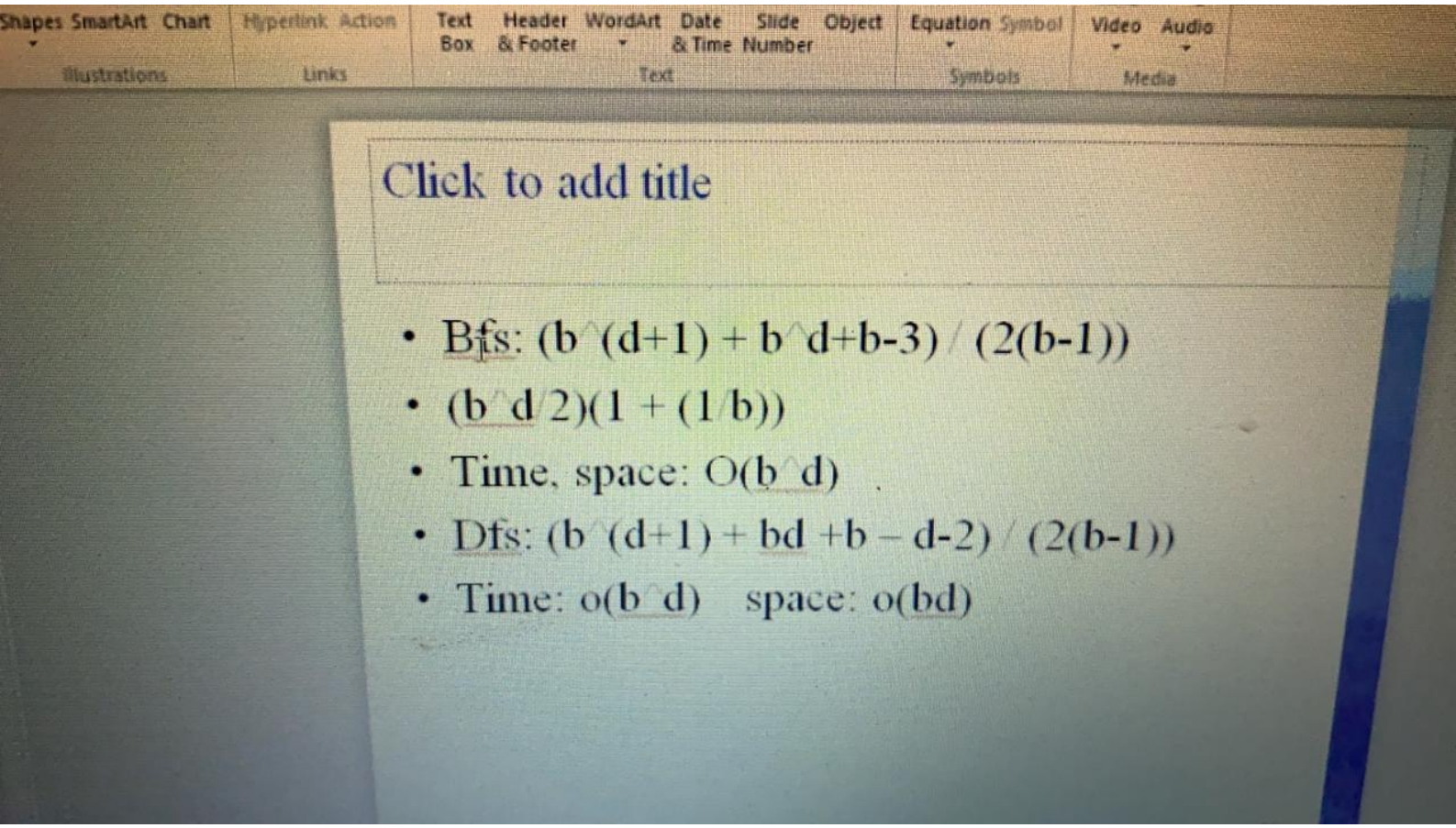
- Search proceeds simultaneously in two directions; both forward from a initial state and backward from a goal state; terminated when the two meet
- Used when S and G configurations are known and operators can be used in reverse manner e.g., 8-puzzle
- Complete; optimal; time: $O(2b^{d/2})$; space: $O(b^{d/2})$

Island-Driven Search

- A series of intermediate goals (islands) is identified between start and goal states

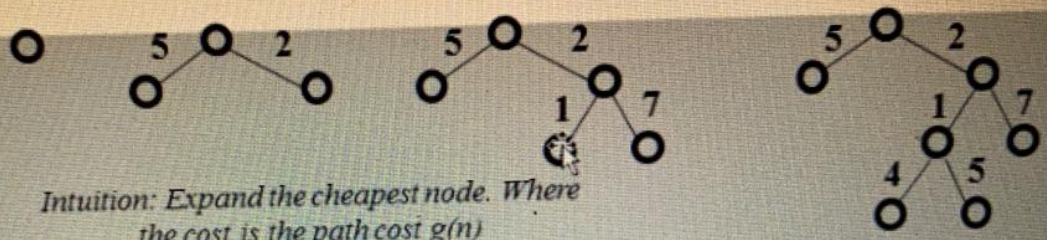
S→I1→I2→I3→…→Im→G

- One large search problem is equivalent to m small search problems
- time: $O(mb^{d/m}) << b^d$ if islands are evenly spaced


Shapes SmartArt Chart
Hyperlink Action
Text Box
Header & Footer
WordArt
Date & Time
Slide Number
Object
Equation Symbol
Video Audio
Illustrations
Links
Text
Symbols
Media
Click to add title
• Bfs: (b^(d+1) + b^d+b-3) / (2(b-1))
• (b^d/2)(1 + (1/b))
• Time, space: O(b^d)
• Dfs: (b^(d+1) + bd +b – d-2) / (2(b-1))
• Time: o(b^d) space: o(bd)

Uniform Cost Search

Enqueue nodes in order of cost



Intuition: Expand the cheapest node. Where the cost is the path cost g(n)

Click to add notes

Informed Search

- Add domain specific information to select the path
- Controls/guides search processes
- Heuristic function ($\hat{h}$) estimates the cost of reaching a goal node from the present state in the search space, small values of $\hat{h}$ → best node
- A **heuristic** is a method that might not always find the best solution but is guaranteed to find a **good solution in reasonable time**
- By sacrificing completeness it increases efficiency

  e.g., 8-puzzle:

  # tiles misplaced

  Manhattan distance

Uniform Cost Search

- Variant of BFS in which nodes are expanded outward from the start node along the contours of equal cost rather than along the contours of equal depth
- Like Dijkstra's shortest path algorithm
- UCS reduces to BFS if all arc costs are identical
- Complete, optimal, time ~ $O(b^d)$, space ~ $O(b^d)$

Heuristics for 8-puzzle II: Manhattan distance

- The **Manhattan Distance** (not including the blank)

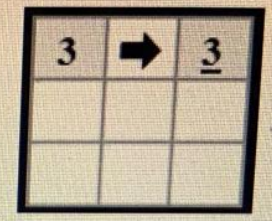
Current State

3	2	8
4	5	6
7	1	

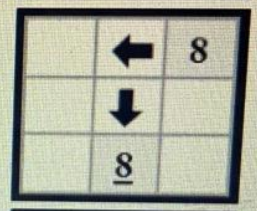
Goal State

1	2	3
4	5	6
7	8	

In this case, only the "**3**", "**8**" and "**1**" tiles are misplaced, by 2, 3, and 3 squares respectively, so the heuristic function evaluates to 8.



2 spaces



3 spaces

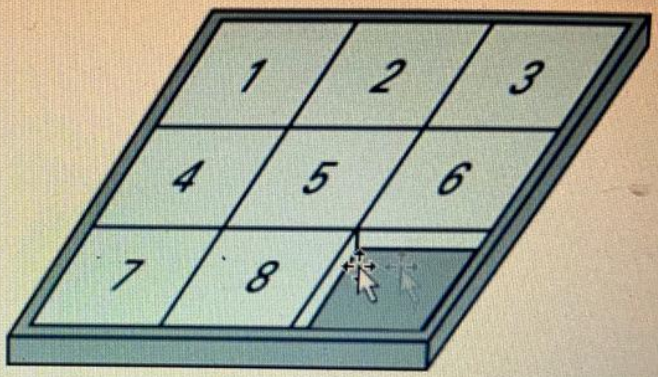
3 spaces

Total 8

Notation: h(n) h(current state) = 8

The Eight-Puzzle (Sliding Tiles) Problem

1	3	5
4	2	
7	8	6

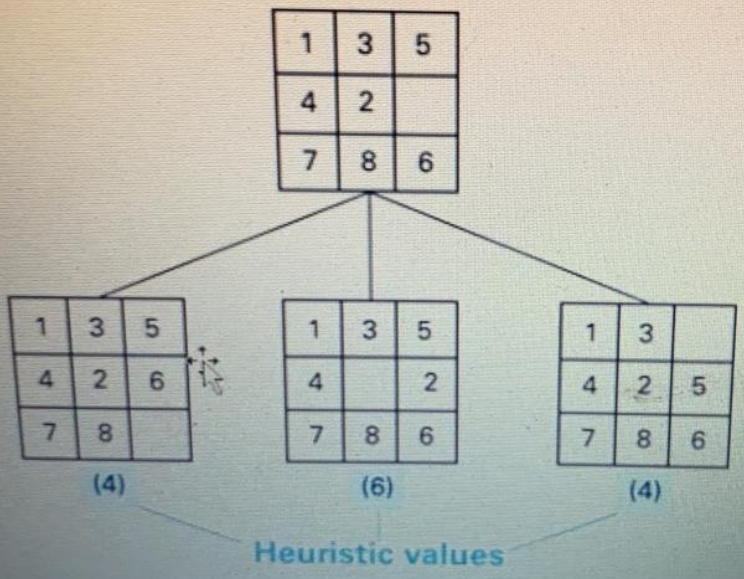


Note: finding **optimal** solution of *n*-puzzle family is NP-hard!

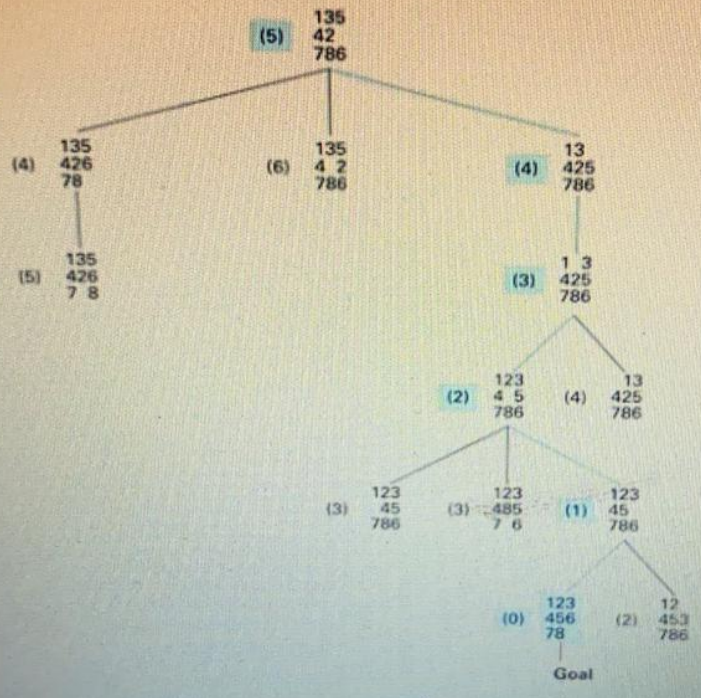
Also, from certain states you can't reach the goal.

Total number of states 9! = **362,880** (more interesting space; not all connected... only half can reach goal state)

The beginning of our heuristic search (Greedy): Sliding Tiles



The complete search tree formed by our heuristic system



Informed search (contd.)

Best first greedy search

- Evaluation function (f) = $\hat{h}$
- Shallowest goal may not be reached

A*

- Evaluation function (f) = $\hat{g} + \hat{h}$

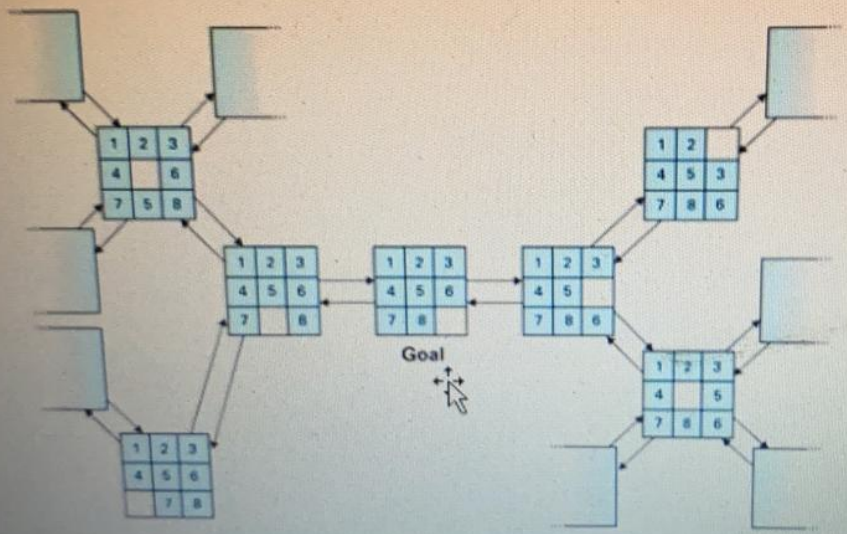
$\hat{g}$ = cost of reaching node n from start node

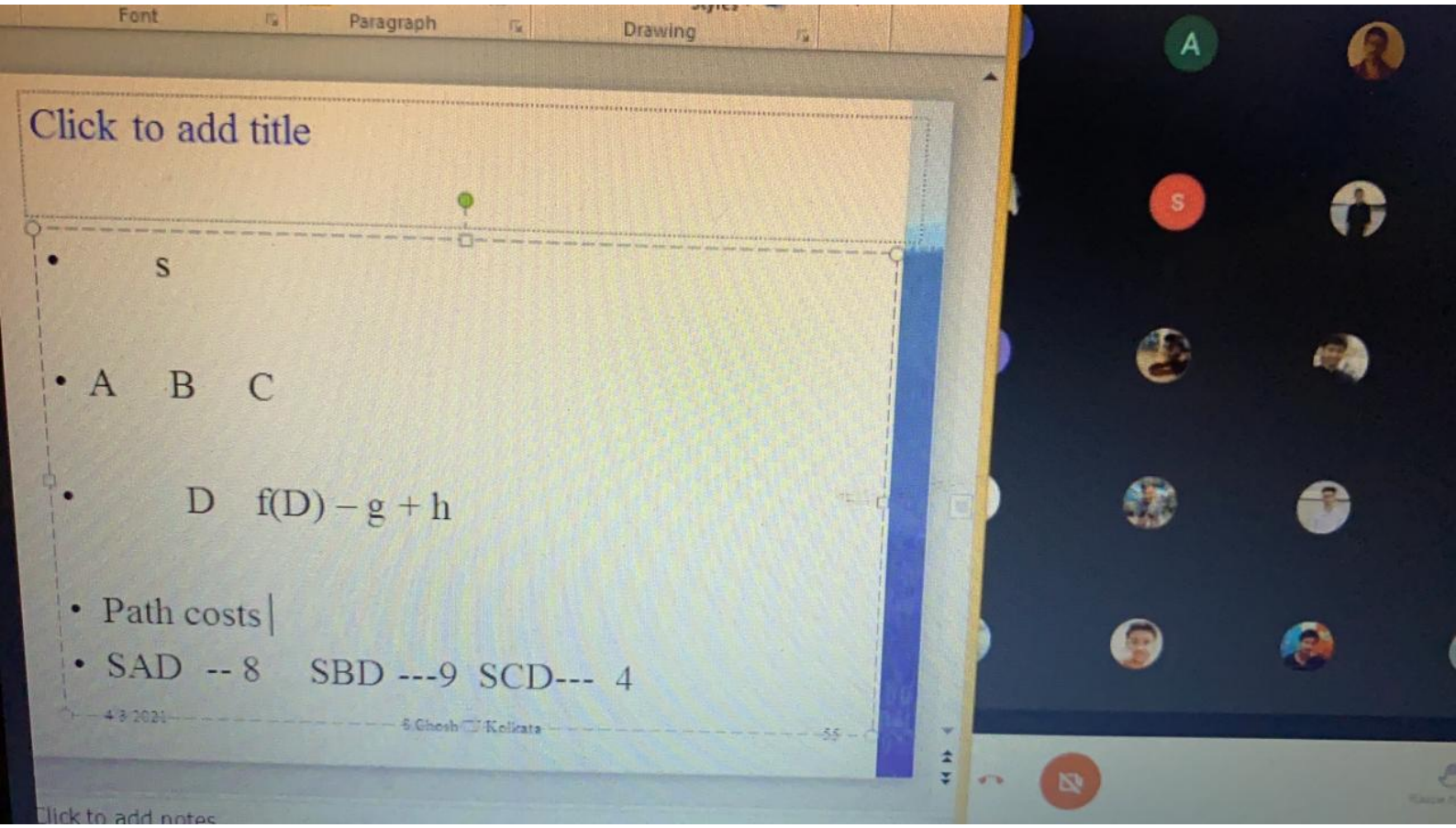
A will always satisfy optimality if the heuristic is admissible*

$\hat{h}(n)$ is said to be admissible if

$\hat{h}(n) <= h(n)$, for all n (for minimization problem)

A small portion of the eight-puzzle's state graph




Font
Paragraph
Drawing
Click to add title
• s
• A B C
• D f(D) – g + h
• Path costs
• SAD -- 8 SBD ---9 SCD--- 4
4/3/2021
S Ghosh Kolkata
55
Click to add notes
A
S

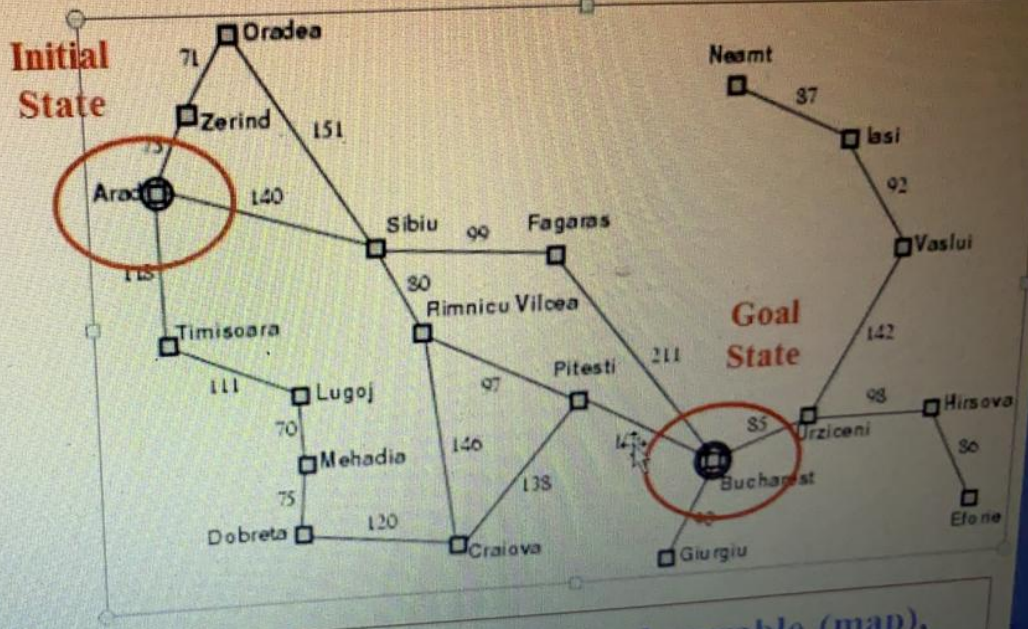
n Bucharest
mania)

late problem:

tion: drive between
ir of connected cities
irect road)

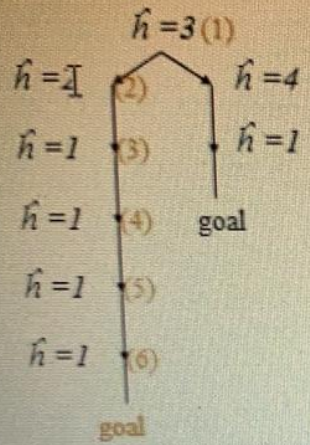
tate: be in a city
20 world states)

l solution:
sequence of cities
leading from start to
goal state, e.g., Arad,



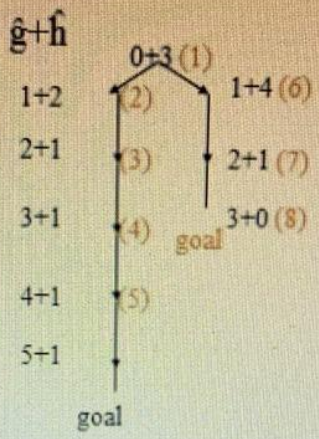
Environment: fully observable (map), deterministic, and the agent knows effects of each action. Is this really the case?

Example



f = ĥ

greedy



f =ĝ+ĥ

A*

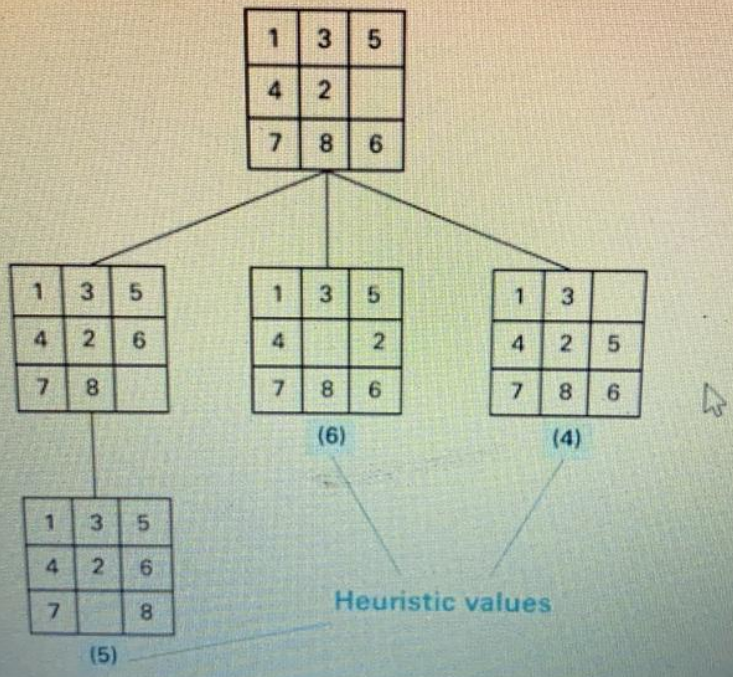
Font Paragraph Drawing Styles Shape Effects

Click to add title

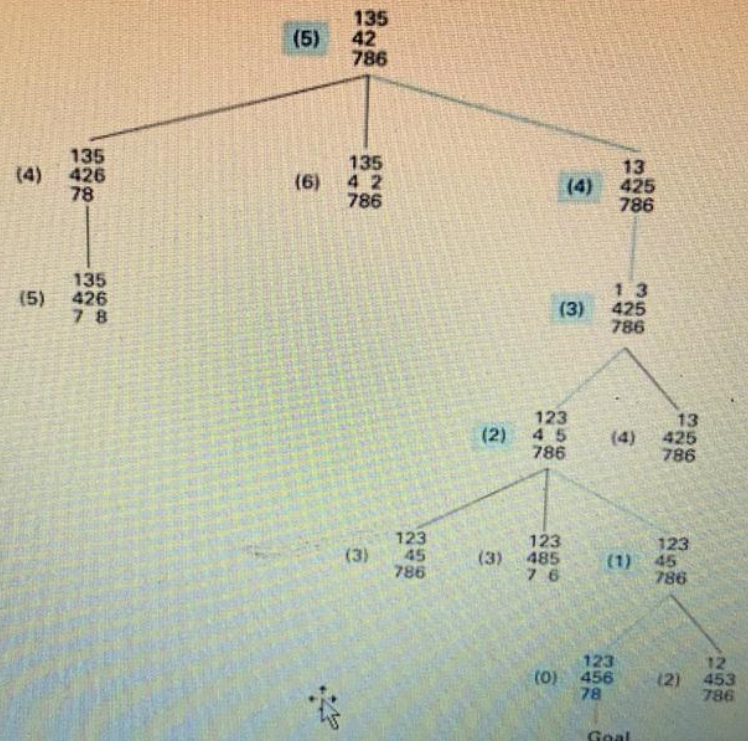
- S
- A B C
- D f(D) – g + h (path cost
- Path costs
- SAD --8 SBD ---9 SCD--- 4
- G = 8 ; G =

- A B C
- D f(D) – g + h (path cost
- Path costs
- SAD -- 8 SBD ---9 SCD--- 4
- G = 8 ; G = 8, G = 4
- All paths explore: g = min |

The search tree after two passes

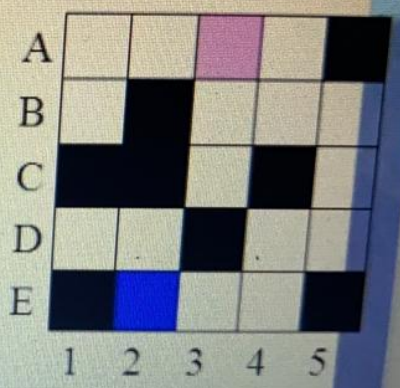


The complete search tree formed by our heuristic system



A* : Maze Traversal

Problem: To get from square A3 to square E2, one step at a time, avoiding obstacles (black squares).



Operators: (in order)

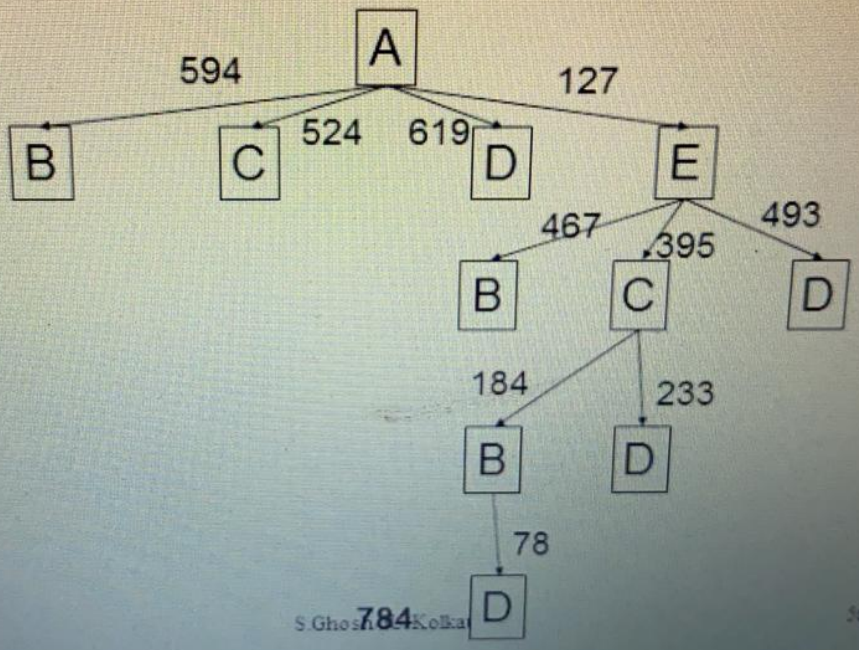
- `go_left`(n)
- `go_down`(n)
- `go_right`(n)

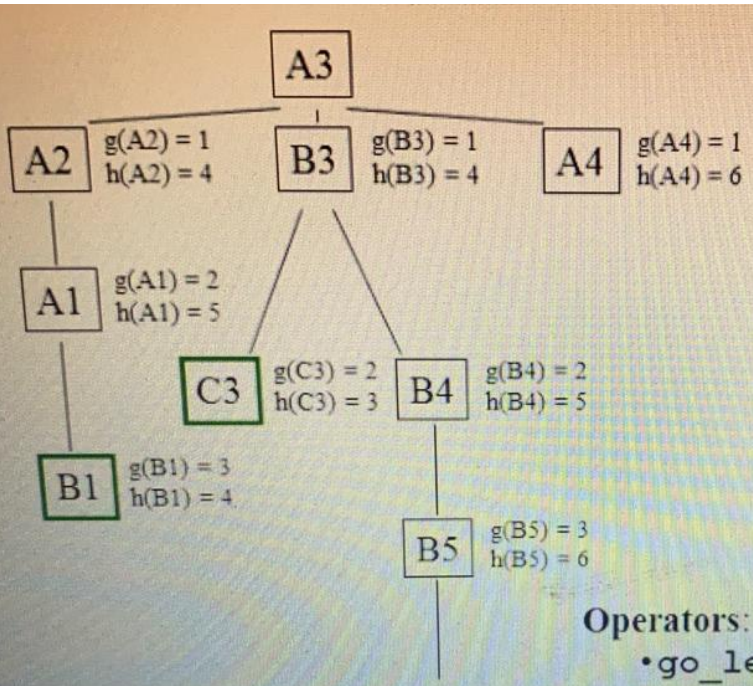
each operator costs 1.

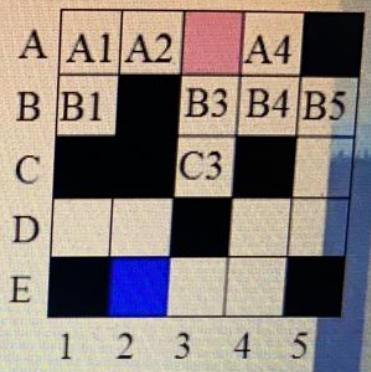
Heuristic: Manhattan distance

Informed/ Heuristic Search: TSP

- heuristic = rule of thumb







Operators: (in order)

- `go_left(n)`
- `go_down(n)`
- `go_right(n)`

each operator costs 1.

to add notes

Informed Search (contd.)

IDA*

- Repeatedly search in DFS fashion over subgraphs with f-cost less than ε. Less than 2ε … until a goal node is found (where, $\varepsilon <= g(n,m)$ for all $n,m \in \delta(n)$
- Time: in worst case if A* expands N nodes, IDA* expands $O(N^2)$ nodes
- Space: $O(b (f^*/ \varepsilon))$, f^*: optimal cost

Admissible Heuristic

$\hat{h}(n)$: heuristic function (applied to node n); estimated distance of node n from goal node

h(n) : actual distance of node n from goal node

$\hat{h}(n)$ is admissible if $\hat{h}(n) <= h(n)$, for all n (for minimization)

If $h >= \hat{h}1(n) > \hat{h}2(n)$ then $\hat{h}1$ is more informed than $\hat{h}2$

Admissibility condition may be relaxed for efficiency but then the obtained solution may not be optimal

g(n)	h(n)	characteristics
1	0	BFS
Cost	0	UCS
0	0	random search
g(n)	admissible	optimal path

$\hat{h}(n)$ is admissible if $\hat{h}(n) <= h(n)$, for all n (for minim

If $h >= \hat{h}1(n) > \hat{h}2(n)$ then $\hat{h}1$ is more informed than $\hat{h}2$

Admissibility condition may be relaxed for efficiency bu
obtained solution may not be optimal

g(n)	h(n)	characteristics
1	0	BFS
Cost	0	UCS
0	0	random search
g(n)	admissible	optimal path

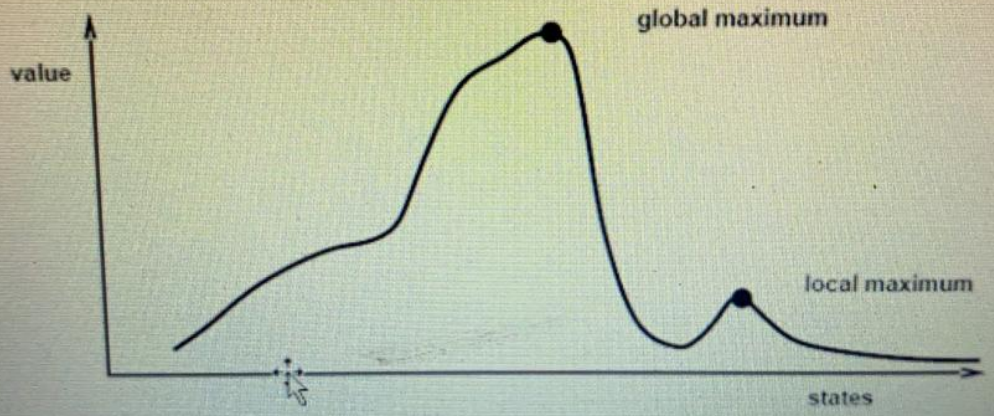
Informed Search (contd.)

IDA*

- Repeatedly search in DFS fashion over subgraphs with *f*-cost less than ε. Less than 2ε … until a goal node is found (where, $\varepsilon <= g(n,m)$ for all $n,m \in \delta(n)$
- Time: in worst case if A* expands N nodes, IDA* expands $O(N^2)$ nodes
- Space: O(b (f*/ ε)), f*: optimal cost

Hill Climbing

Problem: depending on initial state, can get stuck on local maxima



In continuous spaces, problems w/ choosing step size, slow convergence

Iterative Improvement Search

- Every state represents a complete solution to the problem, although not necessarily an optimal one
- Initialize the algorithm at some (random) state, iteratively perturb the current solution, update if any of the tested perturbations yields an improvement
- Never backs up, no search tree

Hill Climbing, Simulated Annealing, Genetic Algorithm

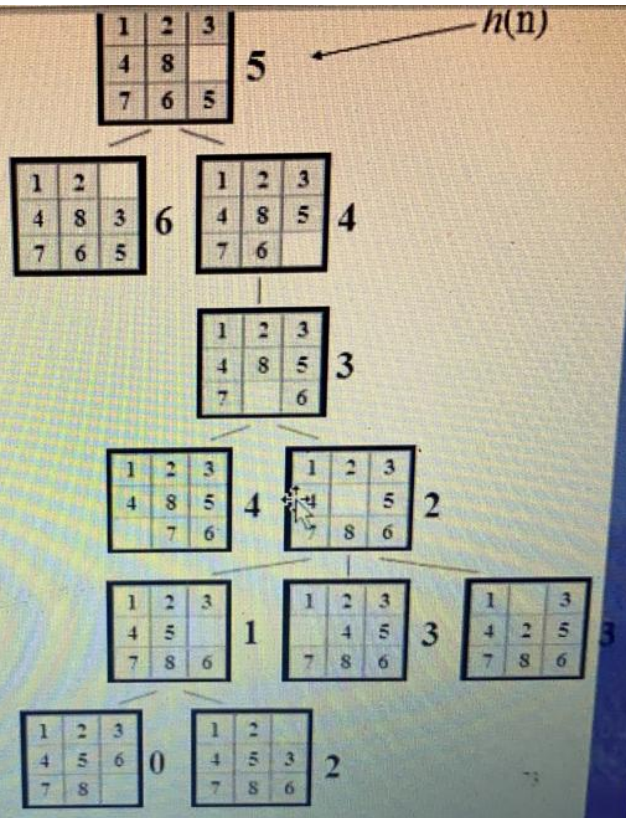
Iterated Hill Climbing (Algorithm)

begin
t= o
repeat
local = FALSE
select a random state sc and compute its functional value f(sc)
repeat
select n new states in the neighborhood of sc (by flipping randomly single bit of sc) and compute their functional values
select the state sn from the set of new states with largest functional value f(sn)
If f(sc) < f(sn) set sc = sn else local = TRUE
until local
t = t+1
until t = MAX
end

We can use heuristics to guide "hill climbing" search.

In this example, the Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle.

But "hill climbing" has a problem...



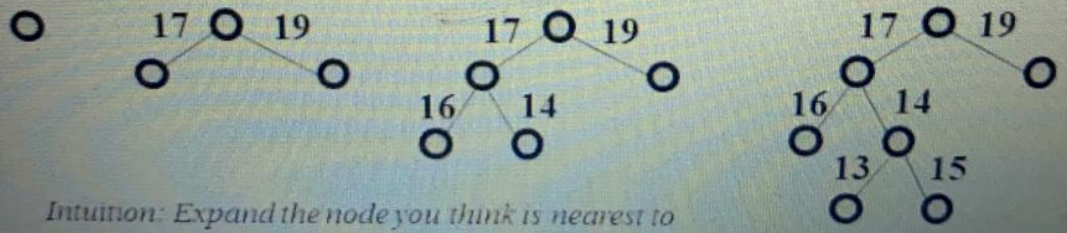
Click to add notes

Drawbacks of Hill climbing

- Local optima / foothill problem – attracted to local optima
- Plateau problem – nowhere to turn on the flat
- Ridge problem – no available operator to move from the region

Hill Climbing Search

Enqueue nodes in order of estimated distance to goal



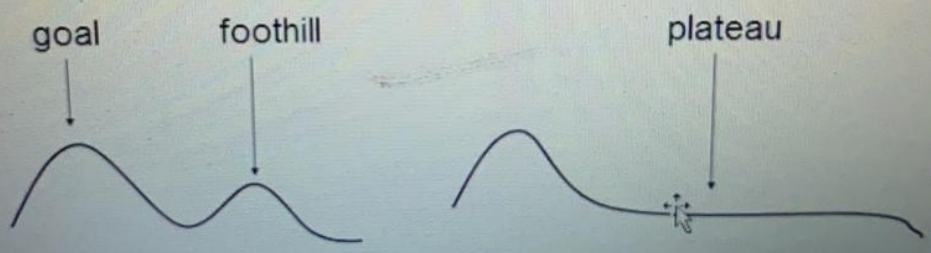
Intuition: Expand the node you think is nearest to goal. Where the estimate of distance to goal is h(n)

Click to add notes

Hill Climbing

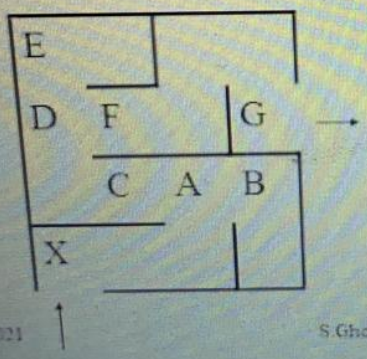
- Walk as much uphill as possible
- Move from one point to that adjacent point having highest elevation
- Solution found, but not necessarily an optimal one

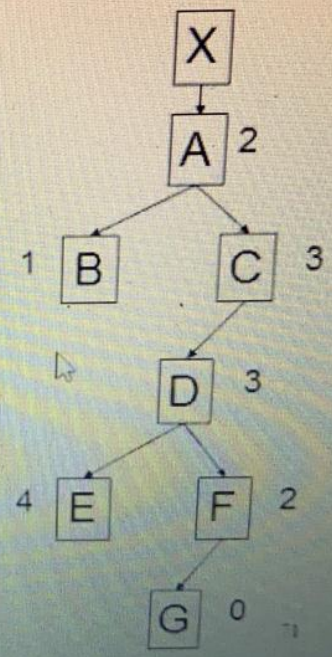
Heuristic Search – hill climbing



Iterative Improvement Search – Hill Climbing – Maze Problem

- expand node
- sort children according to Heuristic Evaluation Function
- choose best value





Click to add notes

Simulated Annealing (Algorithm)

```
begin
  t= tmax
  select a random state sc and compute its functional value f(sc)
      while (t >= tmin)
       for i = 1 to n
            begin
            pick an adjacent state sn from sc  at random and
    compute  f(sn)
             If f(sc) >= f(sn) set  sc =  sn
             else set sc =  sn with probability {exp(- (f(sn) – f(sc))/t}
             end
        decay t
        end
end
```

Font Paragraph

Simulated Annealing

- Simulates the concept of annealing of physical systems
- State transition follows Boltzmann distribution
- It is proved that it will reach global optima if t (temperature) tends to zero and number of alternatives examined at each temperature tends to infinity
- *Escapes local maxima by allowing some bad moves, but gradually decreasing their frequency*

Drawback: too slow

Click to add notes

Summary

- All uninformed search techniques are more alike than different.
- Breadth-first has space issues.
- Depth-first has optimality and completeness issues.
- Depth-limited search has optimality and completeness (?) issues.
- Iterative deepening is the best uninformed search we have discussed.