MACHINE INTELLIGENCE

UNINFORMED SEARCH STRATEGIES

K.S.Srinivas
Department of Computer Science and Engineering

MACHINE INTELLIGENCE

UNINFORMED SEARCH STRATEGIES

Srinivas K S.

Associate Professor, Department of Computer Science

Uniformed Search Strategy

- Uninformed search strategies use only the information available in the problem definition
- The term means that the strategies have no additional information about states beyond that provided in the problem definition.
- All they can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the order in which nodes are expanded
- We will be discussing 5 types of uninformed search
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Breadth-First Search

- Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- This is achieved very simply by using a FIFO queue for the frontier.
- let us see the pseudo code and simultaneously explore a problem

MACHINE INTELLIGENCE

Problem +algo

function BREADTH-FIRST-SEARCH(problem) **returns** a solution, or failure

node ←a node with STATE = problem.INITIAL-STATE,PATH-COST =0

if problem.GOAL-TEST(node.STATE) then **return** SOLUTION(node)

frontier ←a FIFO queue with node as the only element

explored ←an empty set

loop do

if EMPTY?(frontier) then **return** failure

node ←POP(frontier) /* chooses the shallowest node in frontier */

add node.STATE to explored

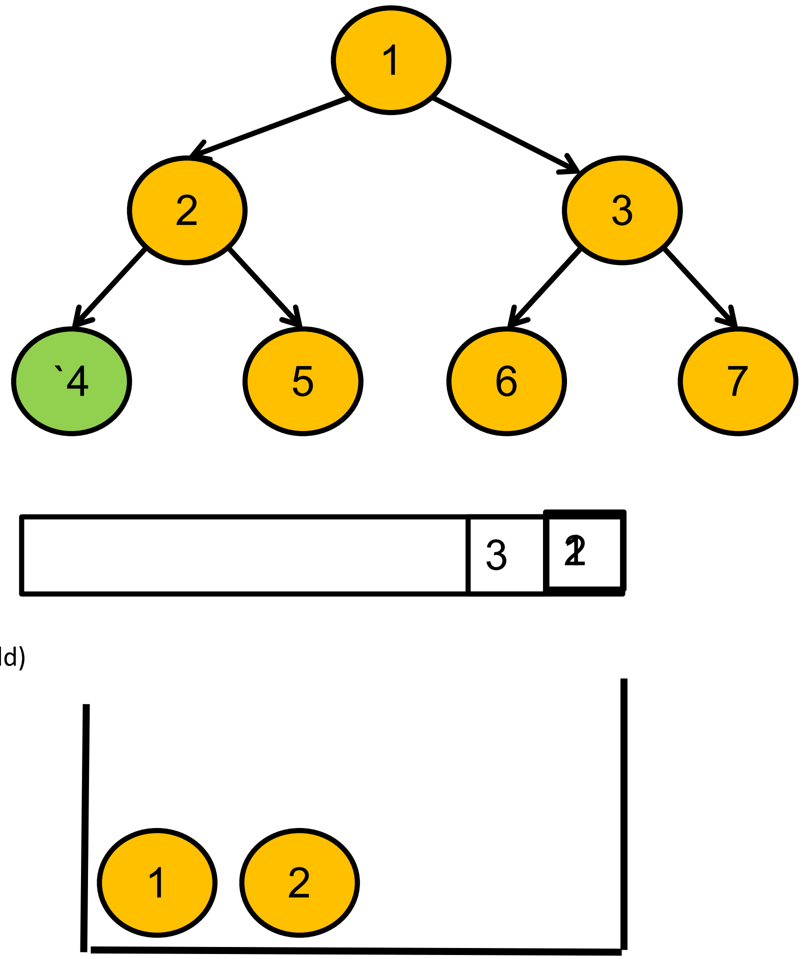
for each action in problem.ACTIONS(node.STATE) **do**

    child ←CHILD-NODE(problem,node,action)

    if child.STATE is not in explored or frontier then

        if problem.GOAL-TEST(child.STATE) then **return** SOLUTION(child)

        frontier ←INSERT(child,frontier)



How good is it???

COMPLETENESS

we can easily say its complete.If the shallowest goal node is at some finite depth d ,breadth-first search will eventually find it after generating all shallower nodes with finite branching factor b.

TIME COMPLEXITY

for a uniform tree where every state b has b successor.
root would generate b nodes at first level each of which generates b more nodes.In the worst case scenario for a tree of depth d the number of nodes generated would be $b+b^2+b^3+.......+b^d=O(b^d)$

SPACE COMPLEXITY

For breadth-first graph
every node generated remains in memory. There will be $O(b^{d-1})$ in the explored set and $O(b^d)$ in the queue.

OPTIMALITY

the shallowest goal node is not necessarily the optimal one
technically, breadth-first search is optimal if the path cost is a non decreasing function of the depth of the node. The most common such scenario is that all actions have the same cost.

Uniform Cost Search

- when all the step cost are equal bfs is optimal because it always expands the shallowest unexploded node
- uniform-cost search is an extension bf when the cost is not uniform.
- uniform-cost search expands the node n with the lowest path cost g(n)
- This is done by storing the frontier as a priority queue ordered by g.
- lets see the pseudo code and then explore a problem

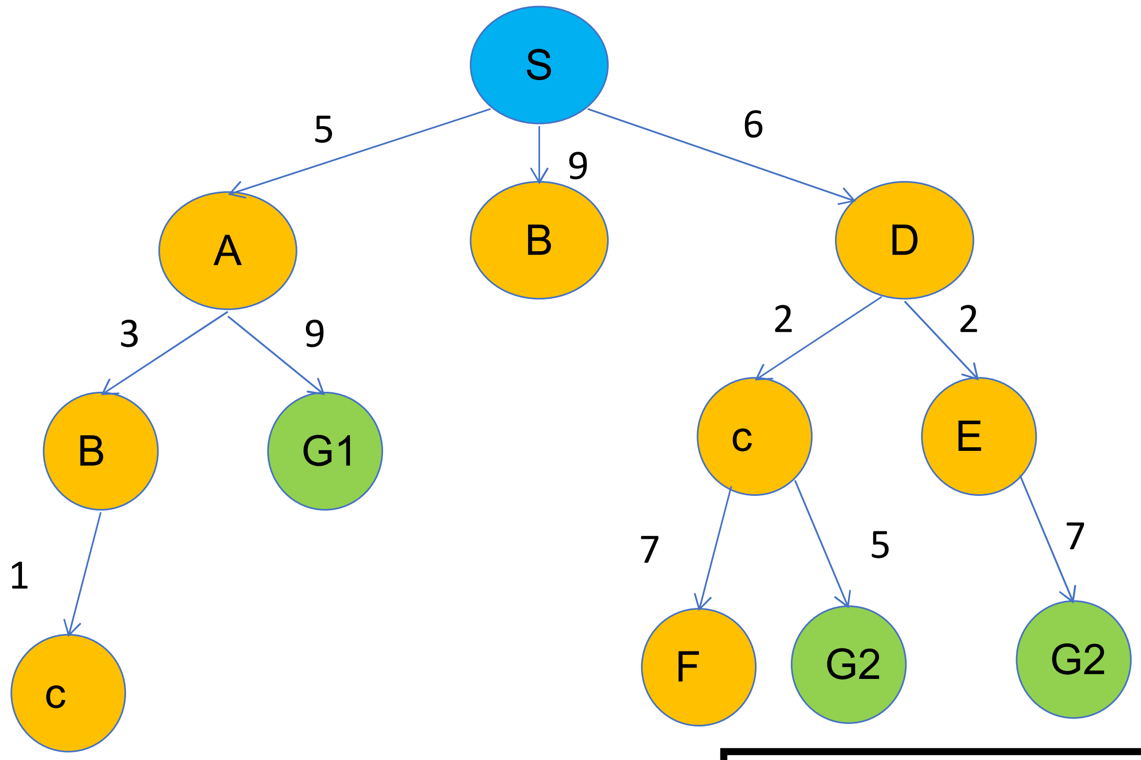
Pseudo-code

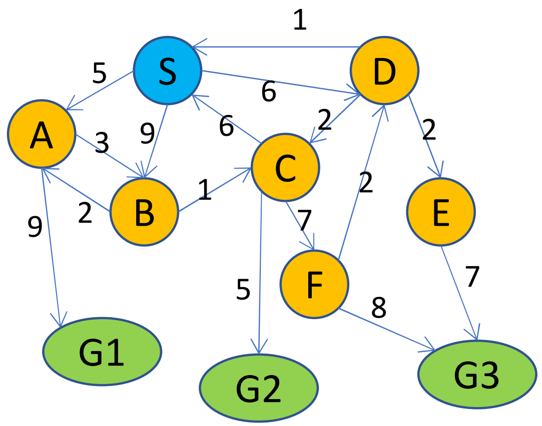
function UNIFORM-COST-SEARCH(problem) **returns** a solution, or failure
node ←a node with STATE = problem.INITIAL-STATE,PATH-COST =0
frontier ←a priority queue ordered-by PATH-COST, with node as the only element
explored ←an empty set
loop do
if EMPTY?(frontier) **then** return failure
node ←POP(frontier) /* chooses the lowest-cost node in frontier */
if problem.GOAL-TEST(node.STATE) **then return** SOLUTION(node)
add node.STATE to explored
for each action in problem.ACTIONS(node.STATE) **do**
 child ←CHILD-NODE(problem,node,action)
 if child.STATE is not in explored **or** frontier **then** frontier ←INSERT(child,frontier)
else if child.STATE is in frontier with higher PATH-COST then replace that frontier node with child

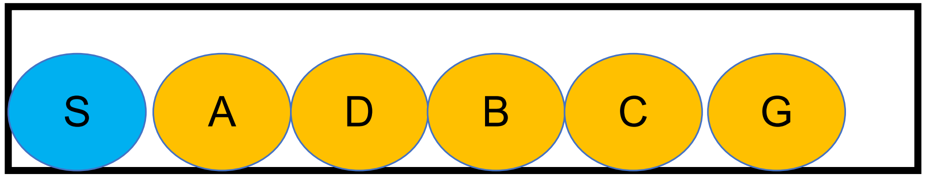
MACHINE INTELLIGENCE

Problem

[illegible] the goal from S







MACHINE INTELLIGENCE

Analyzing

- It is easy to see that uniform-cost search is optimal in general. First, we observe that whenever uniform-cost search selects a node n for expansion, the optimal path to that node has been found.
- Uniform-cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost actions. Completeness is guaranteed provided the cost of every step exceeds some small positive constant ε
- Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of b and d. Instead, let C^* be the cost of the optimal solution,and assume that every action costs at least ε. Then the algorithm's worst-case time and space complexity is $O(b^{1+[C^*/\varepsilon]})$, which can be much greater than b^d.

How good is it??

COMPLETENESS

yes its complete

TIME COMPLEXITY

$O(b^{1+[C^{*} /\varepsilon]})$,

SPACE COMPLEXITY

$O(b^{1+[C^{*} /\varepsilon]})$,

OPTIMALITY

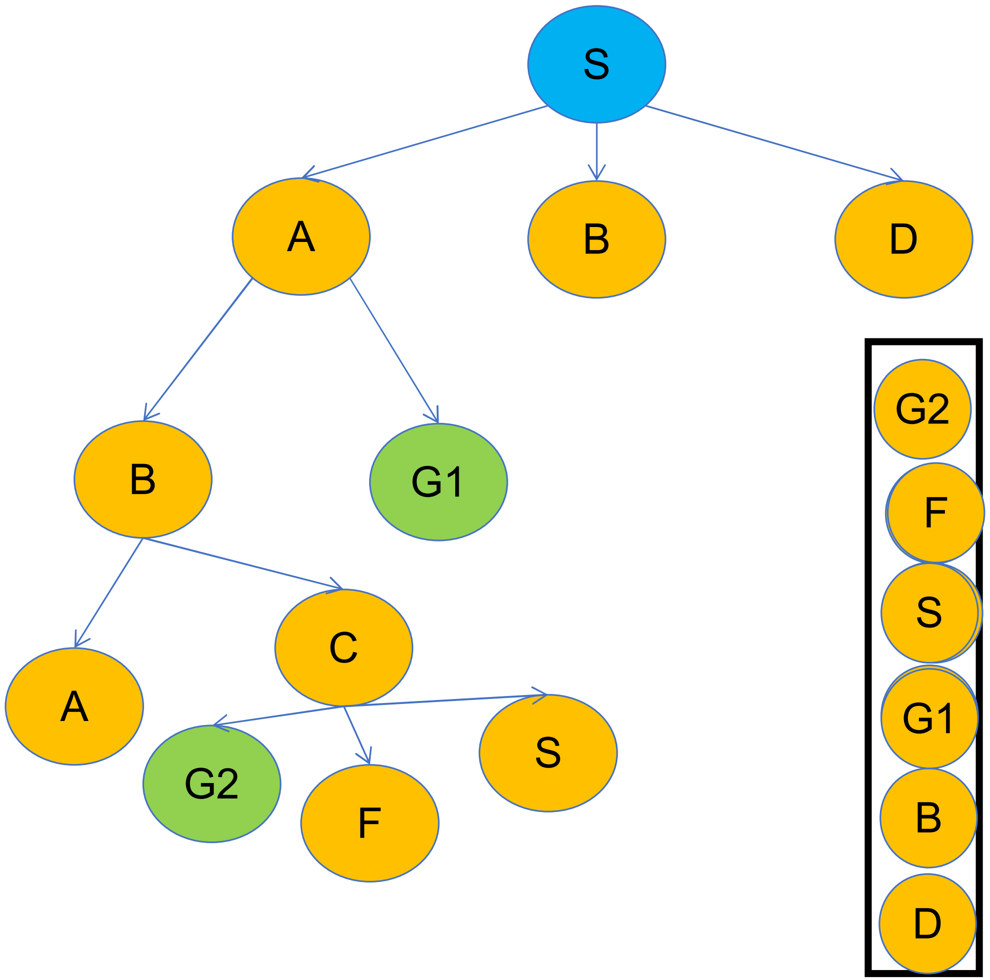
yes it finds an optimal solution

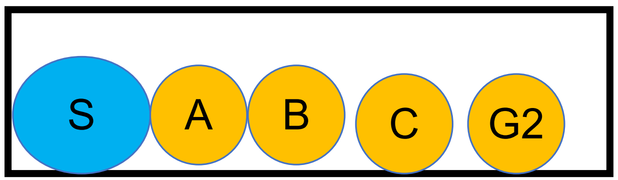
Depth-First Search

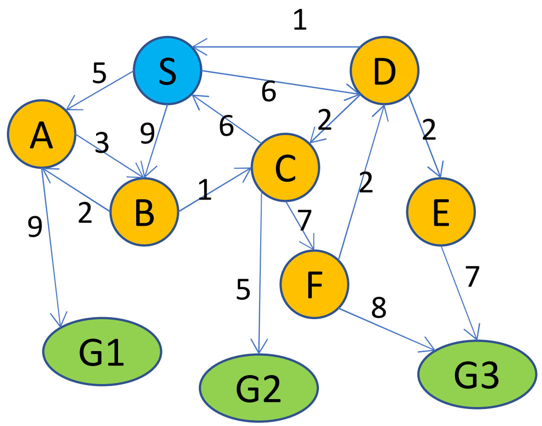
- Depth-first search always expands the deepest node in the current frontier of the search tree.
- depth-first search uses a LIFO queue.
- A LIFO queue means that the most recently generated node is chosen for expansion .
- it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn.
- lets see how it actually does the job for us.

Problem

[illegible] goal from S







How good is it???

COMPLETENESS

fails in infinite-depth spaces, spaces

TIME COMPLEXITY

O(b^m): terrible if *m* is much larger than *d* but if solutions are dense, may be much faster than breadth-first

SPACE COMPLEXITY

O(bm), i.e., linear space!

OPTIMALITY

Not optimal at all

Depth-limit Search

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit l .
- That is, nodes at depth l are treated as if they have no successors. This approach is called depth-limited search.
- let us see the pseudo code for depth-limit search

Pseudo-code

function DEPTH-LIMITED-SEARCH(problem,limit) **returns** a solution, or failure/cutoff

    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE),problem,limit)

function RECURSIVE-DLS(node,problem,limit) **returns** a solution, or failure/cutoff

    if problem.GOAL-TEST(node.STATE) then **return** SOLUTION(node)

    else if limit =0 **then return** cutoff

    else cutoff occurred?←false

    for each action in problem.ACTIONS(node.STATE)

        do child ←CHILD-NODE(problem,node,action)

        result ←RECURSIVE-DLS(child,problem,limit − 1)

        if result = cutoff t**hen** cutoff occurred?←true

        else if result != failure **then return** result

    if cutoff occurred? **then return** cutoff **else return** failure

Iterative deepeining Search

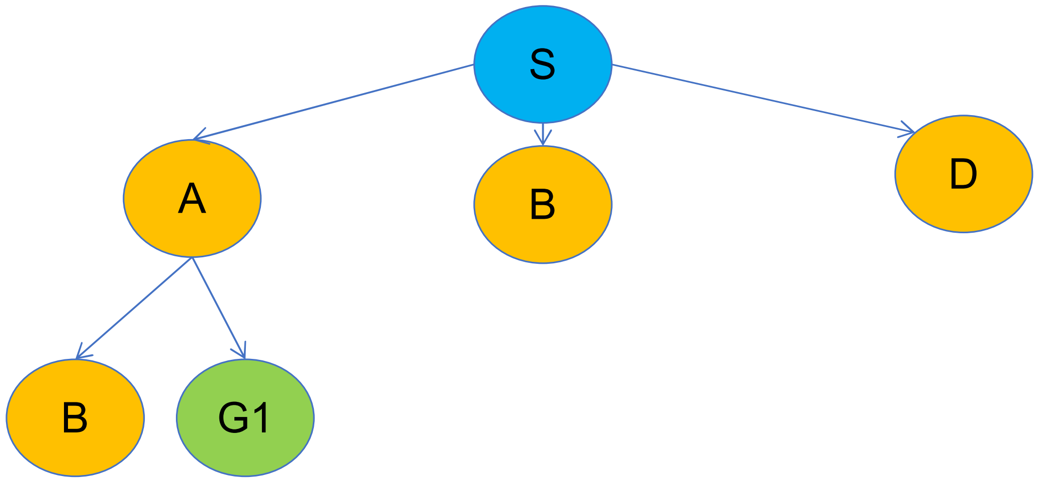
- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit.

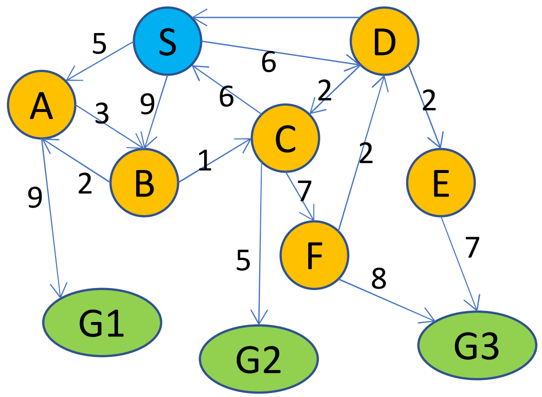
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

MACHINE INTELLIGENCE

Problem





limit l =2

Number of nodes generated in a depth-limited search to depth d with branching factor b:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

Number of nodes generated in an iterative deepening search to depth d with branching factor b:

$$N_{IDS} = (d+1)b^0 + d\ b\hat{\ }^1 + (d-1)b\hat{\ }^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

How good is it??

COMPLETENESS	TIME COMPLEXITY	SPACE COMPLEXITY	OPTIMALITY
DLS: no its not complete you may end up with not finding solution	DLS: $O(b^l)$	DLS: $O(bl)$	DLS: NO
IDS: yes it is complete	IDS: $O(b^d)$	IDS: $O(bd)$	IDS: YES

MACHINE INTELLIGENCE
Applications

- Now let us look at some of the real world example that our search algorithms include.

8-puzzle

1	2	3
6	7	
8	5	4

	1	2
3	4	5
6	7	8

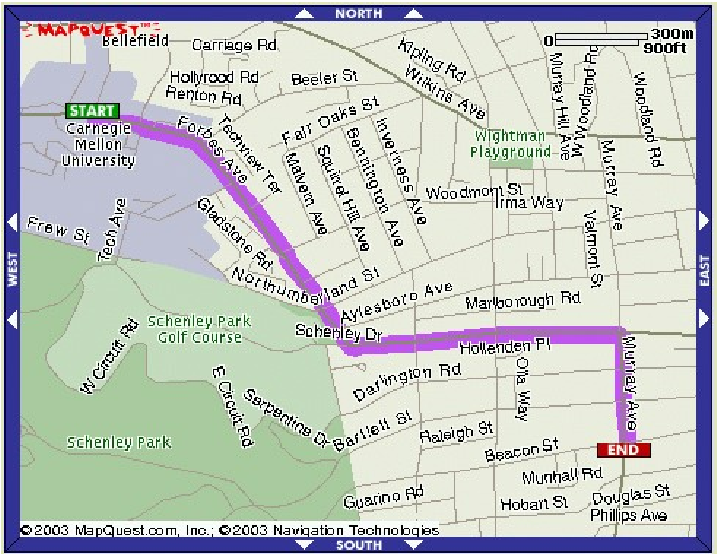
Goal State

Can you try to formalize this search problem by defining the 5 components we discussed earlier?

STATES	Locations of tiles
START STATE	Given position of tiles
GOAL STATE	GIVEN
ACTIONS	move blank left, right, up, down
COST	1 per move

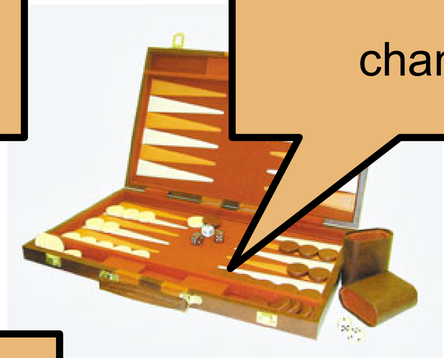
- MAZE PROBLEM

can you guess what kind of search algorithm it may be using? DFS some more applications

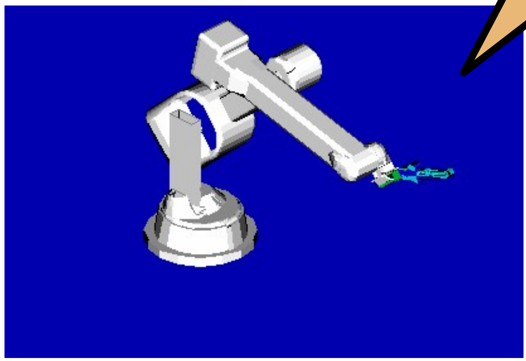


Our definition excludes

Game against adversary



chance



Continuum (infinite number) of states

All of the above, plus distributed team control

THANK YOU

K.S.Srinivas

srinivasks@pes.edu

+91 80 2672 1983 Extn 701