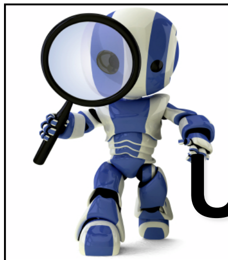
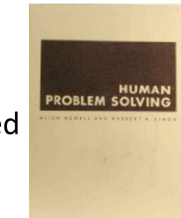


Uninformed Search

Chapter 3

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Big Idea



Allen Newell and Herb Simon developed the *problem space principle* as an AI approach in the late 60s/early 70s

"The rational activity in which people engage to solve a problem can be described in terms of (1) a set of states of knowledge, (2) operators for changing one state into another, (3) constraints on applying operators and (4) control knowledge for deciding which operator to apply next."

Newell A & *Simon* H A. *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall. 1972.

Today's topics

- Goal-based agents
- Representing states and operators
- Example problems
- Generic state-space search algorithm
- Specific algorithms
  - Breadth-first search
  - Depth-first search
  - Uniform cost search
  - Depth-first iterative deepening
- Example problems revisited

Example: 8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.

5	4	
6	1	8
7	3	2

Start State

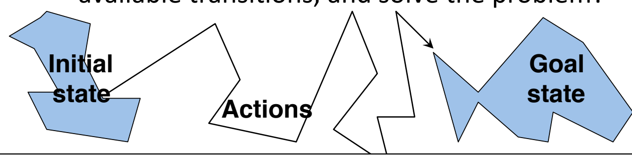
1	2	3
8		4
7	6	5

Goal State

Building goal-based agents

To build a goal-based agent we need to answer the following questions:

- How do we represent the **state** of the 'world'?
- What is the **goal** to be achieved?
- What are the **actions**?
- What *relevant* information should be encoded to describe the state of the world and the available transitions, and solve the problem?



What is the goal to be achieved?

- Could describe a situation we want to achieve, a set of properties that we want to hold, etc.
- Requires defining a **"goal test"** so that we know what it means to have achieved/satisfied our goal.
- A hard question, rarely tackled in AI; usually assume the system designer or user specifies the goal to be achieved.
- Psychologists and motivational speakers stress the importance of establishing clear goals as a first step towards solving a problem.
- What are your goals???

What are the actions?

- Characterize the **primitive actions** or events that are available for making changes in the world in order to achieve a goal.
- **Deterministic** world: no uncertainty in an action's effects. Given an action (a.k.a. **operator** or move) and a description of the **current world state**, the action completely specifies
  - whether that action *can* be applied to the current world (i.e., is it applicable and legal), and
  - what the exact state of the world will be after the action is performed in the current world (i.e., no need for "history" information to compute what the new world looks like).

Representing actions

- Actions in this framework can all be considered as **discrete events** that occur at an **instant of time**, e.g.:
  if "Mary is in class" and performs action "go home," then the next state she is "at home." There is no representation of a point in time where she's neither in class nor at home (i.e., in the state of "going home").
- Number of actions/operators depends on the **representation** used in describing a state
  - 8-puzzle: we could specify 4 possible moves for each of the 8 tiles, resulting in a total of **4*8=32 operators**.
  - On the other hand, we could specify four moves for the "blank" square and we would only need **4 operators**.
- **Representational shift can simplify a problem!**

Representing states

- What information is necessary to describe all relevant aspects to solving the goal?
- The **size of a problem** is usually described in terms of the **number of states** that are possible.
  - Tic-Tac-Toe has about 3^9 states.
  - Checkers has about 10^{40} states.
  - Rubik's Cube has about 10^{19} states.
  - Chess has about 10^{120} states in a typical game.
  - Theorem provers may deal with an infinite space
- state space size ≈ solution difficulty

Closed World Assumption

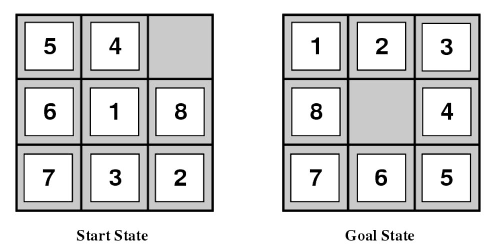
- We will generally use the **Closed World Assumption**.
- All necessary information about a problem domain is available in each percept so that each state is a complete description of the world.
- There is no incomplete information at any point in time.

Some example problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - Missionaries and Cannibals
  - Cryptarithmetic
  - Remove 5 Sticks
  - Water Jug Problem
- Real-world problems

8-Puzzle

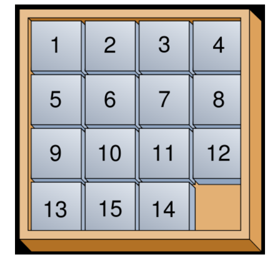
Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.



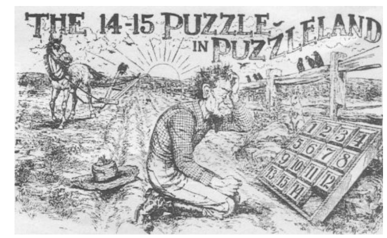
8 puzzle

- **State:** 3 x 3 array configuration of the tiles on the board
- **Operators:** Move Blank Square Left, Right, Up or Down
  – A more efficient operator encoding than one with four possible moves for each of eight distinct tiles
- **Initial State:** A particular configuration of the board
- **Goal:** A particular configuration of the board

15 puzzle

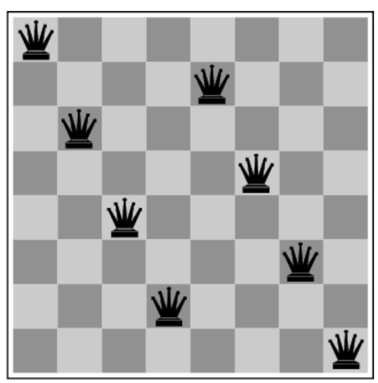


- Popularized, but not invented by, Sam Loyd
- In the late 1800s he offered $1000 to all who could find a solution
- He sold many puzzles
- The states form two disjoint spaces
- From the initial configuration, there is no path to the solution



The 8-Queens Problem

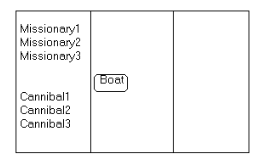
Place eight queens on a chessboard such that no queen attacks any other



Missionaries and Cannibals

There are 3 missionaries, 3 cannibals, and 1 boat that can carry up to two people on one side of a river.

- **Goal**: Move all the missionaries and cannibals across the river.
- **Constraint:** Missionaries can't be out-numbered by cannibals on either side of river, or else the missionaries are killed.
- **State:** configuration of missionaries and cannibals and boat on each side of river.
- **Operators:** Move boat containing some set of occupants across the river (in either direction) to the other side.



3 Missionaries and 3 Cannibals wish to cross the river. They have a boat that will carry two people. Everyone can navigatethe boat. If at any time the Cannibals outnumber the missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross the river safely.
The problem can be solved in 11 moves. But people rarely get the optimal solution, because the MC problem contains a 'tricky' state at the end, where two people move back across the river.

Missionaries and Cannibals Solution

```
                                      Near side      Far side
0 Initial setup:                       MMMCCC  B          -
1 Two cannibals cross over:            MMMC          B  CC
2 One comes back:                      MMMCC   B          C
3 Two cannibals go over again:         MMM           B  CCC
4 One comes back:                      MMMC    B          CC
5 Two missionaries cross:              MC            B  MMCC
6 A missionary & cannibal return:      MMCC    B          MC
7 Two missionaries cross again:        CC            B  MMMC
8 A cannibal returns:                  CCC     B          MMM
9 Two cannibals cross:                 C             B  MMMCC
10 One returns:                        CC      B          MMMC
11 And brings over the third:          -             B  MMMCCC
```

Cryptarithmetic

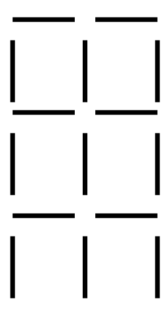
- Find an assignment of digits (0..9) to letters so that a given arithmetic expression is true. examples: SEND + MORE = MONEY and

```
  FORTY     Solution:  29786
+  TEN                   850
+  TEN                   850
  -----                -----
  SIXTY                31486
F=2, O=9, R=7, etc.
```

- The solution is NOT a sequence of actions that transforms the initial state into the goal state
- The solution is a goal node that includes an assignment of digits to each of the distinct letters in the given problem.

Remove 5 Sticks

Given the following configuration of sticks, remove exactly five sticks so that the remaining ones form exactly three squares



Other tasks:

- Remove 4 sticks and leave 4 squares
- Remove 3 sticks and leave 4 squares
- Remove 4 sticks and leave 3 squares

Water Jug Problem

Given a full 5 gallon jug and an empty 2 gallon jug, the goal is to fill the 2 gallon jug with exactly one gal-lon of water.

- State = (x,y), where x is water in the 5G jug and y is water in the 2G gallon jug
- Initial State = (5,0)
- Goal State = (*,1), where * means any amount

Operator table

Name	Cond.	Transition	Effect
Empty5	–	$(x,y)\rightarrow(0,y)$	Empty 5G jug
Empty2	–	$(x,y)\rightarrow(x,0)$	Empty 2G jug
2to5	$x \leq 3$	$(x,2)\rightarrow(x+2,0)$	Pour 2G into 5G
5to2	$x \geq 2$	$(x,0)\rightarrow(x-2,2)$	Pour 5G into 2G
5to2part	$y < 2$	$(1,y)\rightarrow(0,y+1)$	Pour partial 5G into 2G

Some more real-world problems

- Route finding
- Touring (traveling salesman)
- Logistics
- VLSI layout
- Robot navigation
- Theorem proving
- Learning

Knowledge representation issues

- What's in a **state**?
  - Is the boat color relevant to solving the M&C problem? Is sunspot activity relevant to predicting the stock market? What to represent is a very hard problem that is usually left to the system designer to specify.
- What's the best **level of abstraction** to describe the world?
  - Too fine-grained and we'll "miss the forest for the trees." Too coarse-grained and we'll miss critical details for solving the problem.
- The number of states depends on the representation and level of abstraction chosen. E.g., for the Remove-5-Sticks
  - if we represent individual sticks, there are 17-choose-5 possible ways of removing 5 sticks.
  - If we represent the "squares" defined by 4 sticks, there are 6 squares initially and we must remove 3 squares, so only 6-choose-3 ways of removing 3 squares.

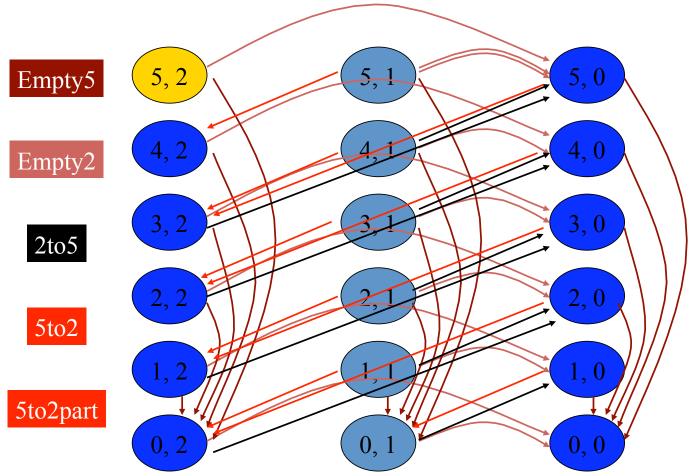
Formalizing search in a state space

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node
- Each **node** is a data structure with a state description plus other information such as the node's parent, the name of the operator that generated it from that parent, and other bookkeeping data
- Each **arc** corresponds to an instance of one of the operators. When the operator is applied to the state associated with the arc's source node, then the resulting state is the state associated with the arc's destination node

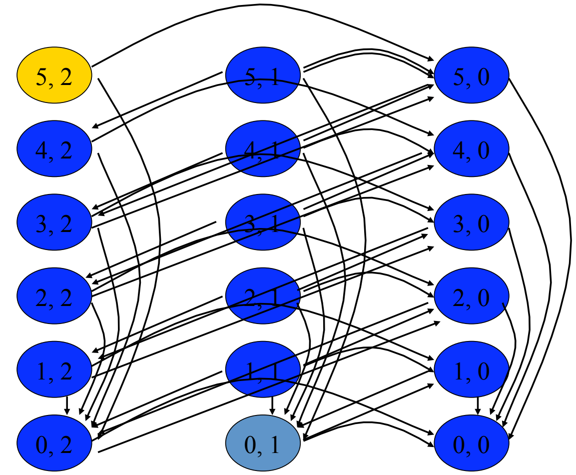
Formalizing search (2)

- Each arc has a fixed, positive **cost** associated with it corresponding to the cost of the operator.
- Each node has a set of **successor nodes** corresponding to all of the legal operators that can be applied at the source node's state.
  - Expanding a node means generating its successor nodes and adding them and their associated arcs to the state-space graph
- One or more nodes are designated as **start nodes.**
- A **goal test** predicate is applied to a state to determine if its associated node is a goal node.

Water jug state space



Water jug solution



CLASS EXERCISE

- Representing a 2x2 Sudoku puzzle as a search space
- Fill in the grid so that every row, every column, and every 2x2 box contains the digits 1 through 4.
  - What are the states?
  - What are the operators?
  - What are the constraints (on operator application)?
  - What is the description of the goal state?

	3		
			1
3			
		2	

Formalizing search (3)

- 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.
- The **cost of a solution** is the sum of the arc costs on the solution path.
  - If all arcs have the same (unit) cost, then the solution cost is just the length of the solution (number of steps / state transitions)

Formalizing search (4)

- **State-space search** is the process of searching through a state space for a solution by **making explicit** a sufficient portion of an **implicit** state-space graph to find a goal node.
  - For large state spaces, it's impractical to represent the whole space
  - Initially V={S}, where S is the start node; when S is expanded, its successors are generated and those nodes are added to V and the associated arcs are added to E. This process continues until a goal node is found.
- A node implicitly or explicitly represents a partial solution path (+ cost of partial solution path) from S to the given node.
  - In general, from this node there are many possible paths (and therefore solutions) that have this partial path as a prefix.

State-space search algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes the start state, operators, goal test, and operator costs
;; queueing-function is a comparator function that ranks two states
;; general-search returns either a goal node or failure
nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
  ;; Note: The goal test is NOT done when nodes are generated
  ;; Note: This algorithm does not detect loops
```

Key procedures to be defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

Bookkeeping

- Typical node data structure includes:
  - State at this node
  - Parent node
  - Operator applied to get to this node
  - Depth of this node (number of operator applications since initial state)
  - Cost of the path (sum of each operator application so far)

Some issues

- Search process constructs a search tree, where
  - **root** is the initial state and
  - **leaf nodes** are nodes
    - not yet expanded (i.e., they are in the list "nodes") or
    - having no successors (i.e., they're "deadends" because no operators were applicable and yet they're not goals)
- Search tree may be infinite because of loops even if state space is small
- Return a path or a node depending on problem.
  - E.g., in cryptarithmetic return a node; in 8-puzzle, a path
- Changing definition of the QUEUEING-FUNCTION leads to different search strategies

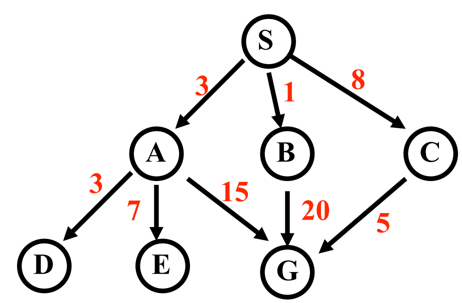
Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the *number of nodes expanded*
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the "nodes" list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost

Uninformed vs. informed search

- **Uninformed search strategies**
  - Aka "blind search,"
  - use no information about the likely "direction" of the goal node(s)
  - Example methods: Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- **Informed search strategies**
  - aka "heuristic search"
  - use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
  - Example methods: hill climbing, best-first, greedy search, beam search, A, A*

Example for illustrating uninformed search strategies



Classic uninformed search methods

- The four classic uninformed search methods
  - Breadth first search
  - Depth first search
  - Uniform cost search *(a generalization of BFS)*
  - Iterative deepening *(a blend of DFS and BFS)*
- To which we can add another technique
  - Bi-directional search *(a hack on BFS)*

Breadth-First

- Enqueue nodes on nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes
  - For a complete search tree of depth 12, where nodes at depths 0..11 have 10 children and nodes at depth 12 have 0, there are $1+10+100+1000...10^{12} = (10^{13}-1)/9 = O(10^{12})$ nodes in the complete search tree
  - If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Breadth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Depth-First (DFS)

- Enqueue nodes on nodes in **LIFO** (last-in, first-out) order, i.e., nodes used as a stack data structure to order nodes.
- **May not terminate** without a “depth bound,” i.e., cutting off search below a fixed depth D (“depth-limited search”)
- **Not complete** (with or without cycle detection, and with or without a cutoff depth)
- **Exponential time**, $O(b^d)$, but only **linear space**, O(bd)
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits a deadend, can only back up one level at a time even if the “problem” occurs because of a bad operator choice near the top of the tree. Hence, only does “chronological backtracking”

Depth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**. I.e., let g(n) = cost of path from start to current node n. Sort nodes by increasing value of g.
- Called *"Dijkstra's Algorithm"* in the algorithms literature and similar to *"Branch and Bound Algorithm"* in operations research literature
- **Complete (*)**
- **Optimal/Admissible** (*)
  - Admissibility depends on the goal test being applied *when a node is removed from the nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$

Uniform-Cost Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }

Solution path found is S B G, cosst 13

Number of nodes expanded (including goal node) = 7

Depth-First Iterative Deepening (DFID)

- First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.
- **Complete**
- **Optimal/Admissible** if all operators have the same cost. Otherwise, not optimal but guarantees finding solution of shortest length (like BFS).
- Time complexity is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but since almost all nodes are near the bottom of a tree, the worst case time complexity is still exponential, $O(b^d)$

Depth-First Iterative Deepening

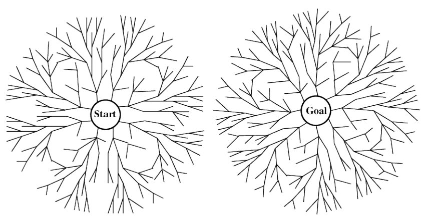
- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db <= b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, then worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- **Linear space complexity**, O(bd), like DFS
- Has advantage of BFS (i.e., completeness) and also advantages of DFS (i.e., limited space and finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

How they perform

- **Depth-First Search:**
  - Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - Expanded nodes: S A D B C E G
  - Solution found: S B G (cost 13)

  This is the only uninformed search that worries about costs.
- **Iterative-Deepening Search**:
  - nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)

Bi-directional search



- Alternate searching from the start state toward the goal and from the goal state toward the start.
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states.
- Requires the ability to generate “predecessor” states.
- Can (sometimes) lead to finding a solution more quickly.

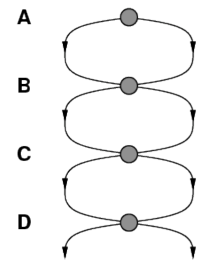
Comparing Search Strategies

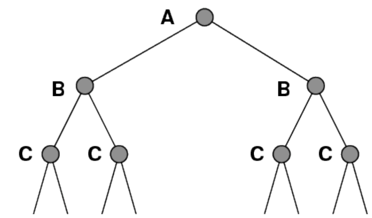
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Avoiding Repeated States

- In increasing order of effectiveness in reducing size of state space and with increasing computational costs:
  1. Do not return to the state you just came from
  2. Do not create paths with cycles in them
  3. Do not generate any state that was ever created before
- Net effect depends on frequency of “loops” in state space.

A State Space that Generates an Exponentially Growing Search Space





Holy Grail Search

Expanded node	Nodes list
	$\{ S^0 \}$
S^0	$\{C^8\ A^3\ B^1 \}$
C^8	$\{ G^{13}\ A^3\ B^1 \}$
G^{13}	$\{ A^3\ B^1 \}$

Solution path found is S C G, cost 13 **(optimal)**

Number of nodes expanded (including goal node) = 3
(as few as possible!)

If only we knew where we were headed…