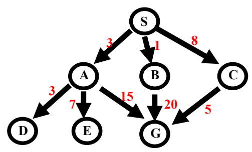
Artificial Intelligence

Uninformed Search (Ch. 3.4)

(and a little more formalization)



Some material adapted from slides by Gang Hua of Stevens Institute of Technology

Some material adapted from slides by Charles R. Dyer, University of Wisconsin-Madison

Dr. Cynthia Matuszek – CMSC 671 *Slides adapted with thanks from: Dr. Marie desJardins*

1

Homework 1

- Blackboard is open! Check access **before tomorrow**
- See corrections in Piazza:
  - Point values in III.2 should be *3, 6*, and *9*
  - Your PDF file should contain parts I, II, *and IV*
  - Example return in III.1.(b) should be in brackets:
    - lottery() ⇒ [75, 235, 7, 100]
- Common Mistakes:
  - Don't print additional information
    - Functions should return or print, not both
  - No extra arguments or return values
    - Return or output things in the order and format specified

2

2

Questions?

- Bread-first, depth-first, uniform cost search
- Generation and expansion
- Goal tests
- Queueing function
- Complexity, completeness, and optimality
- Heuristic functions (for informed search)
- Admissibility

3

3

Formalizing Search: Review

- A *state space* is a **graph** (V, E):
  - V is a set of **nodes** (states)
  - E is a set of **arcs** (agent operations/actions)
- *State space* contains all possible states

8

8

Formalizing Search: III

- **Solution:** a sequence of operators…
  - Giving a path
  - Through state space
  - From a start node to a goal node
- **Solution cost:** sum of arc costs on solution path
  - If all arcs have the same cost, then the solution cost = the length of the solution

9

9

Formalizing Search: IV

- **State-space search:** 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
  - Initially V={S}, where S is the start node
  - When S is **expanded**, its successors are **generated**; those nodes are added to V and the arcs are added to E
  - This process continues until a goal node is found
- It isn't usually practical to represent entire space

10

10

Formalizing Search: V

- Each node implicitly or explicitly represents a **partial solution path** from start node to itself
  - (And a cost!)
  - In general, from a node there are many possible paths (and therefore solutions) that have this partial path as a prefix

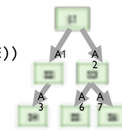
11

11

State-Space Search Algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes start state, operators, goal test,
;;    and operator costs
;; queueing-function is a comparator function that
;;    ranks two states
;; 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
```

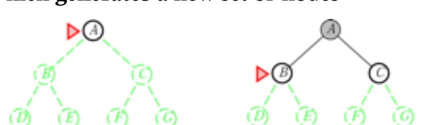


12

12

Generation vs. Expansion

- **Selecting** a state means making that node current
- **Expanding** the current state means applying every legal action to the current state
- Which **generates** a new set of nodes

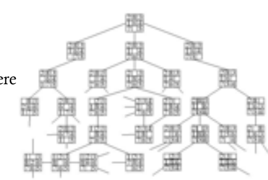


13 *R&N pg. 68, 80*

13

Key Procedures

- EXPAND
  - Generate **all** successor nodes of a given node
    - "What nodes can I reach from here (by taking what actions)?"



- GOAL-TEST
  - Test if state satisfies goal conditions
- QUEUEING-FUNCTION
  - Maintain a **ranked** list of nodes that are expansion candidates
    - "What should I explore next?"

14

14

Some Issues

- Return a path or a node depending on problem
  - In 8-queens return a **node**
  - 8-puzzle return a **path**
  - What about Sheep & Wolves?
- **Changing definition of `Queueing-Function` → different search strategies**
  - How do you choose what to expand next?*

* All of search is answering this question!

17

17

Review: Characteristics

- **Completeness:** Is the algorithm guaranteed to find a solution (if one exists)?
- **Optimality:** Does it find the optimal solution?
  - (The solution with the lowest path cost of all possible solutions)
- **Time complexity:** How long does it take to find a solution? (# of nodes expanded/visited)
- **Space complexity:** How much memory is needed to perform the search? (max # of nodes in list)

18 *R&N pg. 68, 80*

18

Uninformed vs. Informed Search

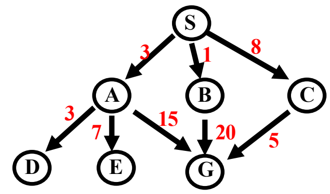
- Uninformed (aka "blind") search
  - Use no information about the "direction" of the goal node(s)
    - No way tell know if we're "doing well so far"
  - Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- Informed (aka "heuristic") search (**next** class)
  - Use domain information to (try to) (usually) head in the general direction of the goal node(s)
  - Hill climbing, best-first, greedy search, beam search, A, A*

19

19

Why Apply Goal Test Late?

- **Why does it matter when the goal test is applied (expansion time vs. generation time)?**
- Optimality and complexity of the algorithms are strongly affected!



20

20

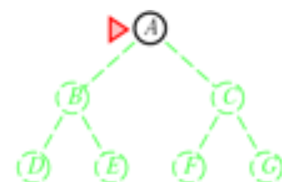
Breadth-First

- Enqueue nodes in **FIFO** (first-in, first-out) order
- Characteristics:
  - **Complete** (meaning?)
  - **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
    - *b* is the branching factor (number of children) at each node
- Takes a **long time to find long-path solutions**

22

22

BFS

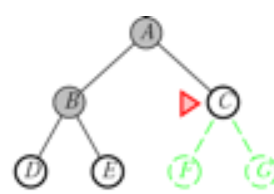


23

BFS

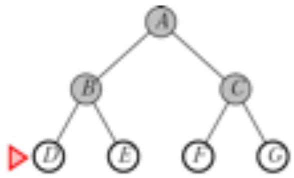
24

BFS



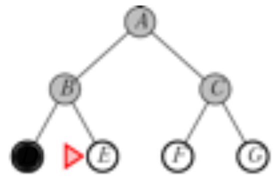
25

BFS



26

BFS



27

Breadth-First: Analysis

- Takes a **long time to find long-path solutions**
  - Must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children:

    $1 + b + b^2 + \ldots + b^d = (b^{d+1} - 1)/(b-1)$ nodes

  - What if we expand nodes when they are selected?
- **Checks a lot of short-path solutions quickly**

28

28

Breadth-First: O(Example)

$1 + b + b^2 + \ldots + b^d = (b^{d+1} - 1)/(b-1)$ nodes

- Tree where: d=12
- Every node at depths 0, ..., 11 has 10 children (b=10)
- Every node at depth 12 has 0 children
- $1 + 10 + 100 + 1000 + \ldots + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree
- If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage
- Will take 35 years to run in the worst case
- Will use 111 terabytes of memory

29

29

Depth-First (DFS)

- Enqueue nodes in **LIFO** (last-in, first-out) order
  - That is, nodes used as a stack data structure to order nodes
- Characteristics:
  - **Might 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)

Loops?

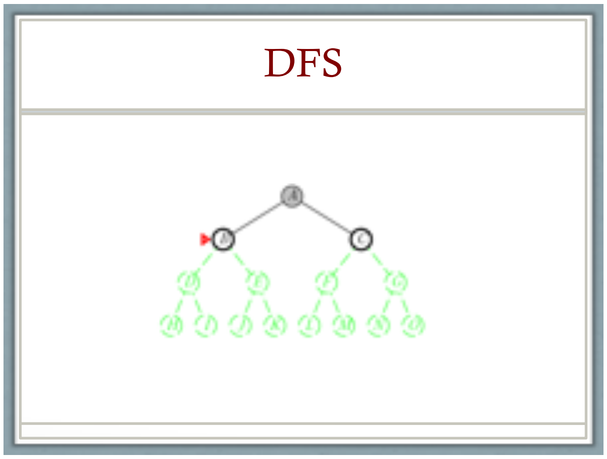
Infinite search spaces?

30

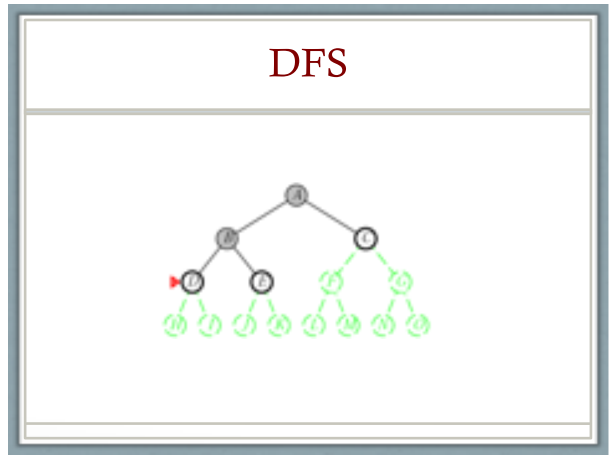
30

DFS

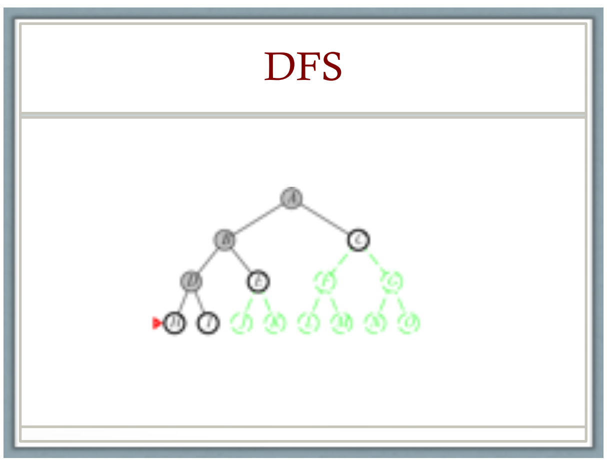
31



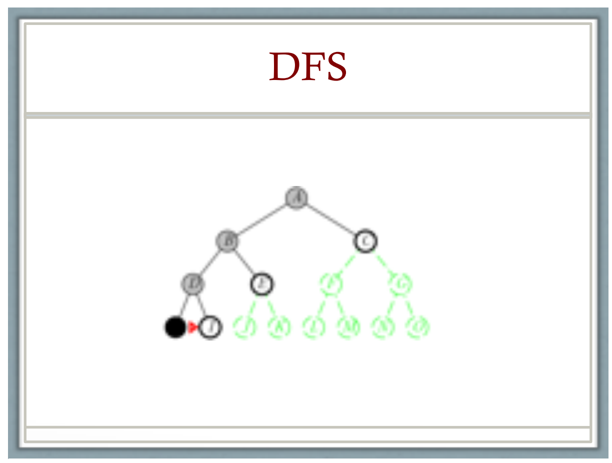
32



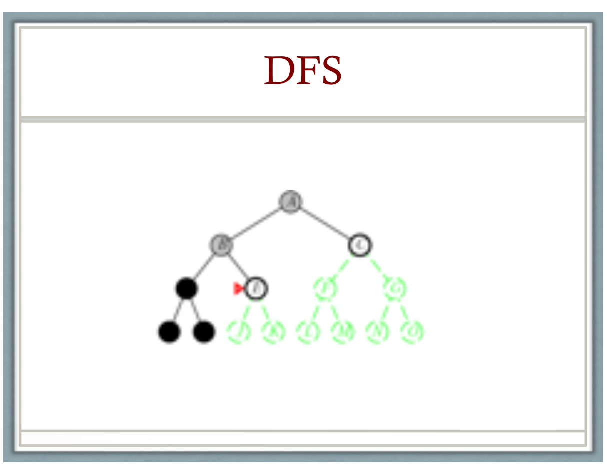
33



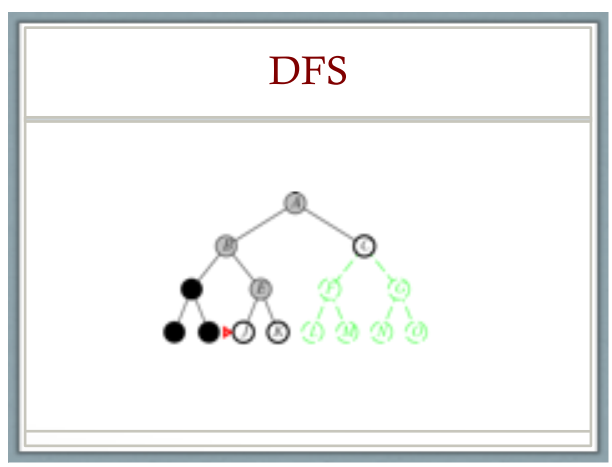
34



35



36

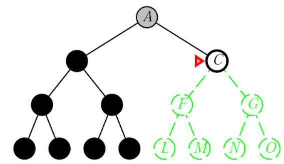


37

DFS

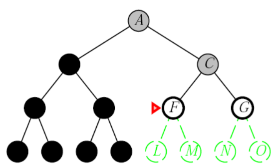
38

DFS



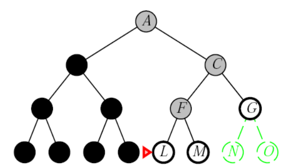
39

DFS



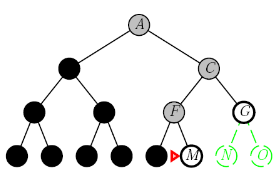
40

DFS



41

DFS



42

Depth-First (DFS): Analysis

- DFS:
  - Can find **long solutions quickly** if lucky
  - And **short solutions slowly** if unlucky
- When search hits a dead end
  - 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"

* Why?

43

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**:
  - Let g(n) = cost of path from start node to current node n
  - Sort nodes by increasing value of g
  - Identical to breadth-first search **if** all operators have equal cost
- *"Dijkstra's Algorithm"* in algorithms literature
- *"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**, $O(b^d)$

44

44

Example: Path Costs

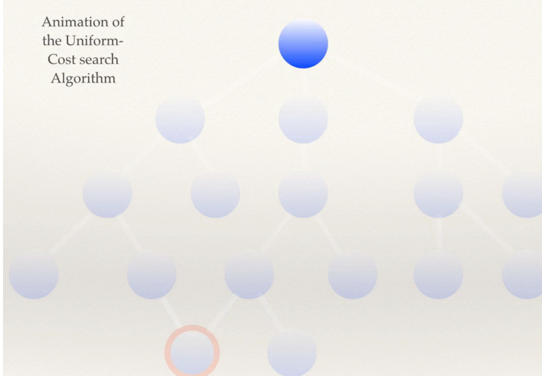
45

45

UCS Implementation

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Equivalent to breadth-first if step costs all equal
- Equivalent to Dijkstra's algorithm in general

46



Shaul Markovitch © 2018

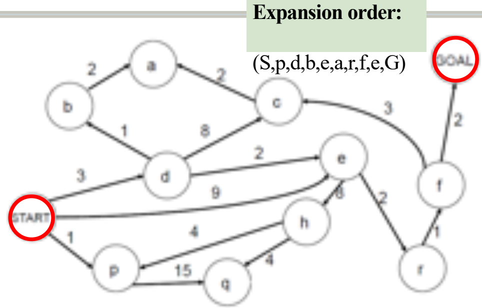
https://www.youtube.com/watch?v=XyoucHYKYSE

47

Uniform-cost search example

Expansion order:

(S,p,d,b,e,a,r,f,e,G)



48

Depth-First Iterative Deepening (DFID)

1. DFS to depth 0 (i.e., treat start node as having no successors)
2. Iff no solution, do DFS to depth 1

until solution found do:
DFS with depth cutoff c;
c = c+1

- **Complete**
- **Optimal/Admissible** if all operators have the same cost
  - Otherwise, not optimal, but guarantees finding solution of shortest length
- **Time complexity** is a little worse than BFS or DFS
- **Nodes near the top of the tree are generated multiple times**
  - Because most nodes are near the bottom of a tree, worst case time complexity is still exponential, O(bd)

50

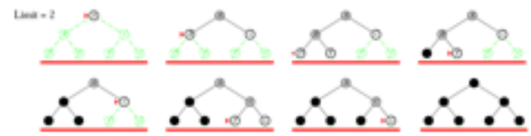
50

Iterative deepening search (c=1)

Nodes visited: 3

51

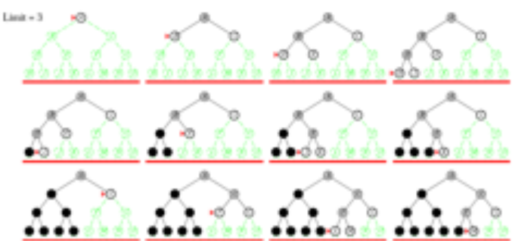
Iterative deepening search (c=2)



Nodes visited: 3+4 = 7

52

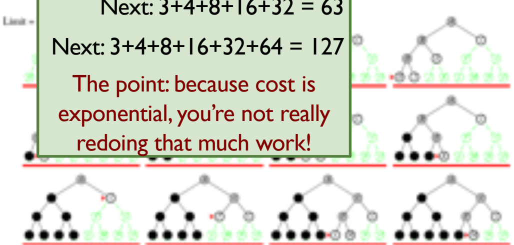
Iterative deepening search (c=3)



Nodes visited: 3+4+8 = 15

53

Iterative deepening search (c=3)



Next: 3+4+8+16 = 31

Next: 3+4+8+16+32 = 63

Next: 3+4+8+16+32+64 = 127

The point: because cost is exponential, you're not really redoing that much work!

Nodes visited: 3+4+8 = 15

54

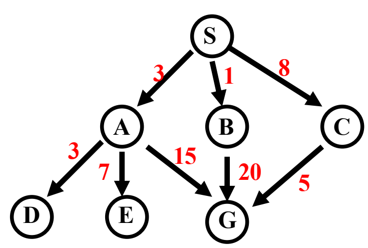
Depth-First Iterative Deepening

- 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 $b^d + 2b^{(d-1)} + \ldots + db \leq b^d / (1 - 1/b)^2 = O(b^d)$.
  - 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 both BFS (completeness) and DFS (limited space, finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

55

55

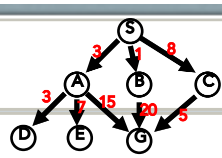
Example for Illustrating Search Strategies



56

56

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 }

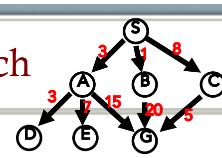
Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

57

57

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} }

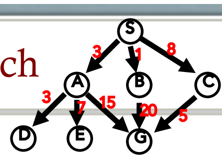
Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

58

58

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 C G, cost 13

Number of nodes expanded (including goal node) = 7

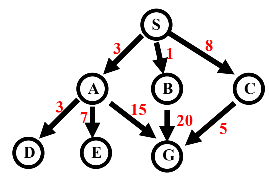
59

59

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 C 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



60

Comparing Search Strategies

	Complete	Optimal	Time complexity	Space complexity
Breadth first search:	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	O(bm)
Depth limited search	if l >= d	no	$O(b^l)$	O(bl)
depth first iterative deepening search	yes	yes	$O(b^d)$	O(bd)
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

b is branching factor, d is depth of the shallowest solution, m is the maximum depth of the search tree, l is the depth limit

61

61

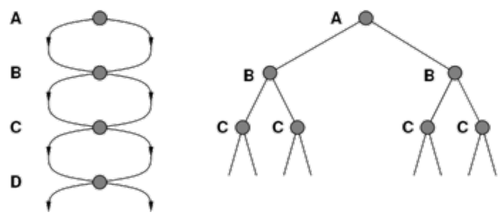
Avoiding Repeated States

- Ways to reduce size of state space (with increasing computational costs)
- In increasing order of effectiveness:
  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.
- Effect depends on frequency of loops in state space.
  - Worst case, storing as many nodes as exhaustive search!

62

62

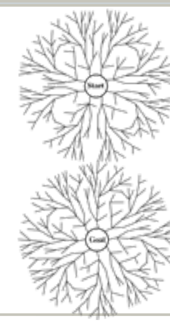
A State Space that Generates an Exponentially Growing Search Space



63

Bi-directional Search

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

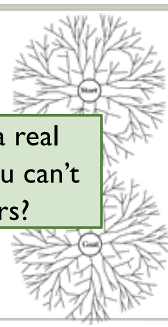


64

Bi-directional Search

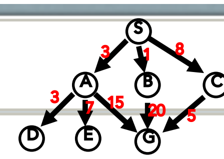
- Alternate searching from
  - start state → goal
  - goal state → start
- St
- W
  u
- R
  “predecessor” states.
- Can (sometimes) find a solution fast

For next time: What’s a real world problem where you can’t generate predecessors?



65

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 (minimum possible!)

66

Holy Grail Search

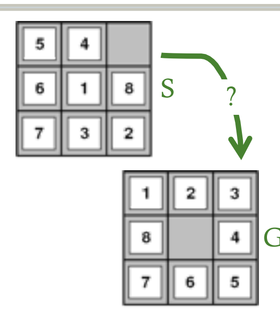
- Why not go straight to the solution, without any wasted detours off to the side?
- If we knew where the solution was we wouldn’t be searching!

If only we knew where we were headed…

67

8-Puzzle Revisited

- What’s a good algorithm?
  - Depth-first search?
  - Breadth-first search?
  - Uniform-cost?
  - Iterative deepening?



68

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - **Preconditions:**
    - <x,y> is empty — 1
    - <x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; … — 3
    - <(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2 — 3
    - **if** <x,y> **in** A, **then** 3 $\notin$ A; … — 4

  × 4

	3		
			1
3			
		2	

- How many operators is that? How many preconditions?
- **Goal:** all blocks are filled

69

69

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - <x,y> is empty — 1
    - <x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; … — 3
    - <(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2 — 3
    - **if** <x,y> **in** A, **then** 3 $\notin$ A; … — 4

  × 4

	3		
			1
3			2
		2	

- How many operators is that?
- **Goal:** all blocks are filled

70

70

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ **<x,y> is empty** — 1
    - <x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; … — 3
    - <(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2 — 3
    - **if** <x,y> **in** A, **then** 3 $\notin$ A; … — 4

  × 4

	3		
			1
3			2
		2	

- How many operators is that?
- **Goal:** all blocks are filled

71

71

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty — 1
    - ✓ **<x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; ..** — 3
    - <(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2 — 3
    - **if** <x,y> **in** A, **then** 3 $\notin$ A; … — 4

  × 4

	3		
			1
3			2
		2	

- How many operators is that?
- **Goal:** all blocks are filled

72

72

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty — 1
    - ✓ <x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; … — 3
    - ✓ **<(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2** — 3
    - **if** <x,y> **in** A, **then** 3 $\notin$ A; … — 4

  × 4

	3		
			1
3			2
		2	

- How many operators is that?
- **Goal:** all blocks are filled

73

73

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty — 1
    - ✓ <x, (y$\pm$1)> $\neq$ 2; <x, (y$\pm$2)> $\neq$ 2; … — 3
    - ✓ <(x$\pm$1), y> $\neq$ 2; … <(x$\pm$3), y> $\neq$ 2 — 3
    - ✗ **if <x,y> in A, then 3 $\notin$ A; …** — 4

  × 4

	3		
			1
3			2
		2	

- How many operators is that?
- **Goal:** all blocks are filled

74

74

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty 1
    - ✓ <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … 3 × 4
    - ✓ <(x±1), y> ≠ 2; … <(x±4), y> ≠ 2 3
    - ✗ **if <x,y> in A, then 3 ∉ A; …** 4

	3		
			1
3			
		2	

- How many operators is that?
- **Goal:** all blocks are filled

75

"Satisficing"

- Wikipedia: "**Satisficing** is … searching until an **acceptability threshold** is met"
- Contrast with **optimality**
  - Satisficable problems *do not get more benefit from* finding an optimal solution

Another piece of **problem definition**

- Ex: You have an A in the class. Studying for four hours will get you a 95 on the final. Studying for four more (eight hours) will get you a 99 on the final. What to do?
- A combination of *satisfy* and *suffice*
- Introduced by Herbert A. Simon in 1956

76