CSC 511: Module 3:

Search Strategies – Uninformed (Blind) Search … exploring the alternatives in a systematic way

Reading: Chap. 3, Section…..

Recall:

Simple Problem-Solving-Agent Algorithm

1. s_0 ≋ sense/read initial state
2. GOAL? ≋ select/read goal test
3. Succ ≋ read successor function 4. solution ≋

search(s_0, GOAL?, Succ)

5. perform(solution)

Search Tree

Search tree

Note that some states may be visited multiple times

3

State graph

Search Nodes and States

3 4

5 6 7

3 4

5 6 3 4

3 4

4~~7~~2

5~~6~~7

8 7 2

5~~6~~

5~~6~~7 8

8 2

1

5~~6~~7 2

1

2 8

8 2

4

8

3

3

1

1

1

1

Search Nodes and States

3 4

5 6 7

3 4

5 6 3 4

3 4

4 7 2

5 6 7 8 If states are allowed to be revisited,

5 6 7

5 6

8 2

5 6 7

1

8 7 2

the search tree may be infinite even 1

when the state space is finite

2
2 8
8 2
4
8
3
3
1
1
1
1
5
3 4
5 6 7

Data Structure of a Node

8 STATE
2

PARENT-NODE

1

BOOKKEEPING

CHILDREN

Action Right

Expanded yes

...

Depth 5 Path-Cost 5

Depth of a node N = length of path from root to N (depth of the root = 0)

6

3 4

5 6 7 8

Node Expansion

3 4

4 7 2 2

5 6

5 6 7 8

5 6 7

The expansion of a node N of the search tree consists of: 1) Evaluating the successor function on STATE(N) N

1

2) Generating a child of N for each state returned by the function

2

2 8 4

8

3

3

1

1

1

7

The fringe (frontier) is the set of all nodes that haven't been 3 4

5 6 7
3 ~~4~~
5 ~~6~~

Fringe/Frontier of Search Tree

3 ~~4~~
3 ~~4~~

4 ~~7~~ 2

expanded yet (they are the leaves of all visited nodes that

haven't 5 ~~6~~

5 ~~6~~ 7 2 8

5 ~~6~~ 7

been expanded yet

8

2

1

8 7

2

1

2

8

4

8

8

2

7

3

3

8

1 561

1 1

Search Strategy

- The fringe (frontier) is the set of all search nodes that haven’t been expanded yet
- The fringe (frontier) is implemented as a priority queue, say FRONTIER
  - INSERT(node, FRONTIER)

- REMOVE(FRONTIER)

The ordering of the nodes in FRONTIER defines the search strategy

Search Algorithm #1

iii. INSERT(N', FRONTIER) Expansion of N

SEARCH#1

1. If GOAL?(initial-state) then return initial-state 2. INSERT(initial-node, FRONTIER)

3. Repeat:

   a. If empty(FRONTIER) then return failure b. N ≈ REMOVE(FRONTIER)

c. s ← STATE(N)

d. For every state s’ in SUCCESSORS(s)

i. Create a new node N’ as a child of N ii. If GOAL?(s’) then return path or goal state

Performance Measures

- Completeness

  A search algorithm is complete if it finds a solution whenever one exists

- Optimality

  A search algorithm is optimal if it returns a minimum-cost path whenever a solution exists

- Complexity

It measures the time and amount of memory required by the

a

l

g

o

r

i

t

h

m

1

1

Blind vs. Heuristic Strategies

- Blind (or un-informed) strategies do not exploit state descriptions to order FRONTIER. They only exploit the positions of the nodes in the search tree

- Heuristic (or informed) strategies exploit state descriptions to order FRONTIER (the most “promising” nodes are placed at the beginning of FRONTIER)

12
3 4

5 6 7

7 ~~8~~ 6 1 2 3
1 ~~2 3~~

Example

4 ~~5~~
4 5 6 7 ~~8~~

8
For a blind strategy, N_1 and N_2 are just two nodes (at some

position in the search

2

tree)

STATE

N_1

1

STATE

N_2

Goal state 13

3 4

5 6 7

7 ~~8~~ 6 1 2 3

1 ~~2 3~~

Example

4 ~~5~~

4 5 6 7 ~~8~~

8

For a heuristic strategy counting the number of misplaced tiles, N_2 is more promising than N_1

2

STATE N_2

STATE N_1

1

Goal state

Remark

- Some search problems, such as the (n^2-1)-puzzle, are

NP-hard One can't expect to solve all instances of such problems in less than exponential time (in n)

One may still strive to solve each instance as efficiently as possible This is the purpose of the search strategy

Blind Strategies

Breadth-first

- Bidirectional

Depth-first

- Depth-limited
- Iterative deepening

Uniform-Cost

(variant of breadth-first)

16

= c(action) AA 0

Arc cost = 1

Arc cost

1

4 ~~5~~ 6 7 FRONTIER = (1)

2 3

Breadth-First Strategy

New nodes are inserted at the end of FRONTIER

2 3 FRONTIER = (2, 3)

1

4 5 6 7

Breadth-First Strategy

New nodes are inserted at the end of FRONTIER 18

2 3 FRONTIER = (3, 4, 5)

1

4 5 6 7

Breadth-First Strategy

New nodes are inserted at the end of FRONTIER

2 3 FRONTIER = (4, 5, 6, 7)

1

4 5 6 7

Breadth-First Strategy

New nodes are inserted at the end of FRONTIER

Important Parameters

1) Maximum number of successors of any state ☾

branching factor b of the search tree

2) Minimal length (≠ cost) of a path between the initial and a goal

state ☾ depth d of the shallowest goal node in the search tree

Evaluation

b: branching factor

d: depth of shallowest goal node

Breadth-first search is:

- Complete? Not complete?
- Optimal? Not optimal?

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:
  $1 + b + b^2 + \ldots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$
- ☾ Time and space complexity is $O(b^d)$

Remark

If a problem has no solution, breadth-first may run for ever (if the state space is infinite or states can be revisited arbitrary many times)

1 2 3 4

1 2 3 4

5 6 7 8

?

5 6 7 8

9

10

11

12

9

10

11

12

13

14

15

13

15

14

2 3 4 5 FRONTIER = (1)

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

25

2 3 4 5 FRONTIER = (2, 3)

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

26

2 3 4 5 FRONTIER = (4, 5, 3)

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

28

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

29

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

30

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

31

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

32

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

33

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

2 3 4 ~~5~~

Depth-First Strategy

New nodes are inserted at the front of FRONTIER 1

35

Evaluation

- b: branching factor
- d: depth of shallowest goal node

- m: maximal depth of a leaf node

Depth-first search is:

  - Complete?
  - Optimal?

- Time complexity is $O(b^m)$

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- m: maximal depth of a leaf node

Reminder: Breadth-first requires $O(b^d)$ time and space

- Depth-first search is:
  - Complete only for finite search tree
  - Not optimal
- Number of nodes generated (worst case):
  $1 + b + b^2 + \ldots + b^m = O(b^m)$
- Space complexity is $O(bm)$ [or $O(m)$]

Depth-Limited Search

- Depth-first with depth cutoff k (depth at which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Iterative Deepening Search Provides

the best of both breadth-first and depth-first search

Main idea:

Totally horrifying !

IDS

For k = 0, 1, 2, … do:

Perform depth-first search with depth cutoff k (i.e., only generate nodes with depth $\leq$ k)

Iterative Deepening

Iterative Deepening

Iterative Deepening

$$(d+1)(1) + db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$$

Performance

- Iterative deepening search is:
  - Complete
  - Optimal if step cost =1
- Time complexity is:
- Space complexity is: O(bd) or O(d)

$$db + (d-1)b^2 + \ldots + (1)\, b^d$$
$$= b^d + 2b^{d-1} + 3b^{d-2} + \ldots + db$$

Calculation

$$= (1 + 2b^{-1} + 3b^{-2} + \ldots + db^{-d}) \times b^d \leq \left(\Sigma_{i=1,\ldots,\infty} ib^{(1-i)}\right) \times b^d = b^d (b/(b-1))^2$$

Number of Generated Nodes

(Breadth-First & Iterative Deepening)

d = 5 and b = 2

BF	ID
1	1 x 6 = 6
2	2 x 5 = 10
4	4 x 4 = 16
8	8 x 3 = 24
16	16 x 2 = 32
32	32 x 1 = 32
63	120

120/63 ~ 2

Number of Generated Nodes

(Breadth-First & Iterative Deepening) d = 5 and b = 10

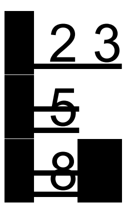
BF	ID
1	6
10	50
100	400
1,000	3,000
10,000	20,000
100,000	100,000
111,111	123,456

123,456/111,111 ~ 1.111

Comparison of Strategies

- Breadth-first is complete and optimal, but has high space complexity

- Depth-first is space efficient, but is neither complete, nor optimal
- Iterative deepening is complete and optimal, with the same space complexity as depth-first and almost the same time complexity as breadth-first



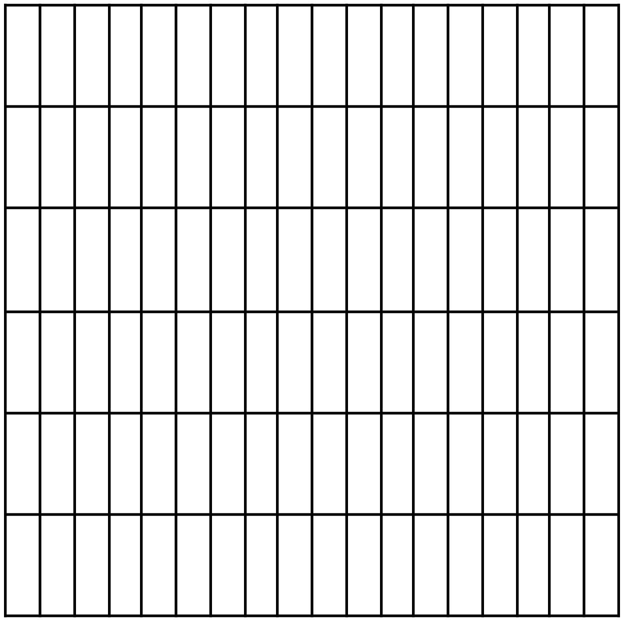
Revisited States

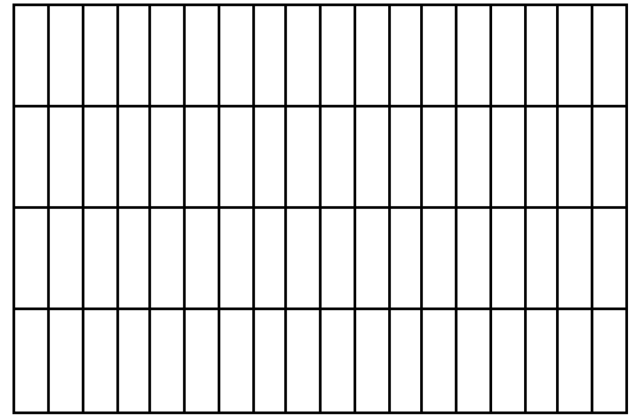
No

Few

Many

search tree is [redacted]ite [redacted]arch tree is[redacted]te





assembly

8-queens

8-puzzle and robot navigation

planning

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:

• Store all states associated with generated nodes in VISITED• If the state of a new node is in VISITED, then discard the node

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:

- Store all states associated with generated nodes in VISITED
- If the state of a new node is in VISITED, then discard the node

Implemented as hash-table
or as explicit data structure with flags

Avoiding Revisited States

- Depth-first search:

Solution 1:

– Store all states associated with nodes in current path in VISITED

– If the state of a new node is in VISITED, then discard the node Only avoids loops

Solution 2:

– Store all generated states in VISITED

– If the state of a new node is in VISITED, then discard the node Same space complexity as breadth-first !

C S G

Uniform-Cost Search

- Each arc has some cost c > 0
- The cost of the path to each node N is

g(N) = costs of arcs

- The goal is to generate a solution path of minimal cost
- The

nodes N in the queue FRINGE are sorted in increasing g(N)

A 1

5

5

10 S 0

B

A_5

B_{15}

1

5

C

15

G

G_{10}

11

Need to modify search algorithm

Search Algorithm #2

ii. INSERT(N', FRONTIER)

SEARCH#2

1. INSERT(initial-node,FRONTIER) 2.

Repeat:

a. If empty(FRONTIER) then return failure b. N ←
REMOVE(FRONTIER)

c. s ← STATE(N)

d. If GOAL?(s) then return path or goal state e. For every
state s' in SUCCESSORS(s)

i. Create a node N' as a successor of N 53

Avoiding Revisited States in Uniform-Cost Search

- For any state S, when the first node N such that STATE(N) = S is expanded, the path to N is the best path from the initial state to S

N

N"

N'

g(N) ⌧ g(N')

g(N) ⌧ g(N")

– If there exits a node N' in the fringe such that STATE(N') = STATE(N), discard the node -- N or N' -- with the highest-cost path

Avoiding Revisited States in Uniform-Cost Search

- For any state S, when the first node N such that STATE(N) = Sis expanded, the path to N is the best path from the initial state to S
- So:
  - When a node is expanded, store its state into CLOSED • When a new node N is generated:
    - If STATE(N) is in CLOSED, discard N