Basics of AI and Machine Learning

State-Space Search: Breadth-first Search

Jendrik Seipp

Linköping University

Slides modified from Basel AI group, with permission

State-Space Search: Overview

Chapter overview: state-space search

- Foundations
- Basic Algorithms
  - Data Structures for Search Algorithms
  - Tree Search and Graph Search
  - Breadth-first Search
  - Uniform Cost Search
  - Depth-first Search
- Heuristic Algorithms

Blind Search

Blind Search

In the next three chapters we consider blind search algorithms:

Blind Search Algorithms

Blind search algorithms use no information about state spaces apart from the black box interface.

They are also called uninformed search algorithms.

contrast: heuristic search algorithms (subsequent chapters)

Blind Search Algorithms: Examples

examples of blind search algorithms:

- breadth-first search
- uniform cost search
- depth-first search
- depth-limited search
- iterative deepening search

Blind Search Algorithms: Examples

examples of blind search algorithms:

- breadth-first search ($\rightsquigarrow$ this chapter)
- uniform cost search
- depth-first search
- depth-limited search
- iterative deepening search

Breadth-first Search: Introduction

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

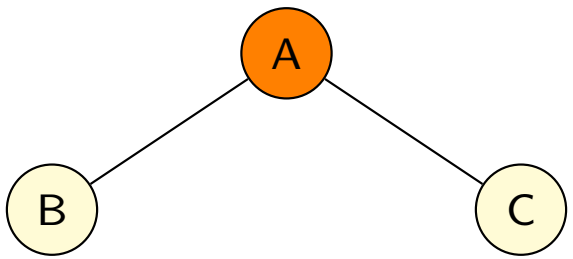
$\rightsquigarrow$ e.g., open list as linked list or deque

open: A

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

$\rightsquigarrow$ e.g., open list as linked list or deque

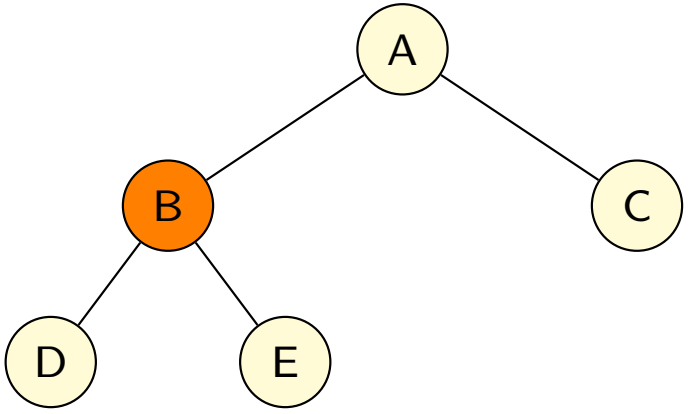


open: B, C

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

$\rightsquigarrow$ e.g., open list as linked list or deque

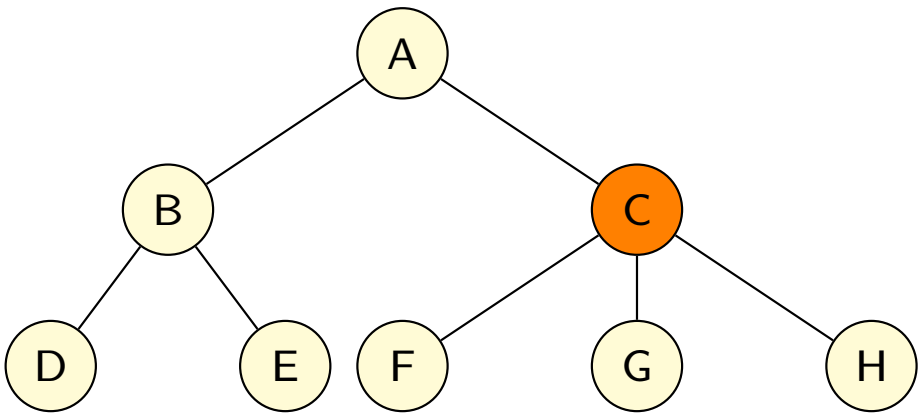


open: C, D, E

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

$\rightsquigarrow$ e.g., open list as linked list or deque

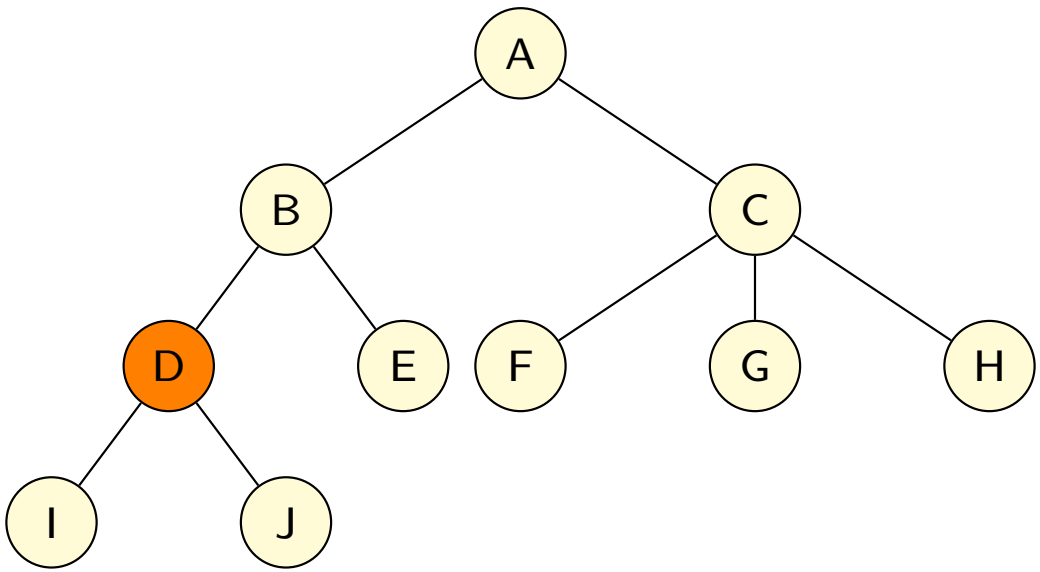


open: D, E, F, G, H

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

$\rightsquigarrow$ e.g., open list as linked list or deque

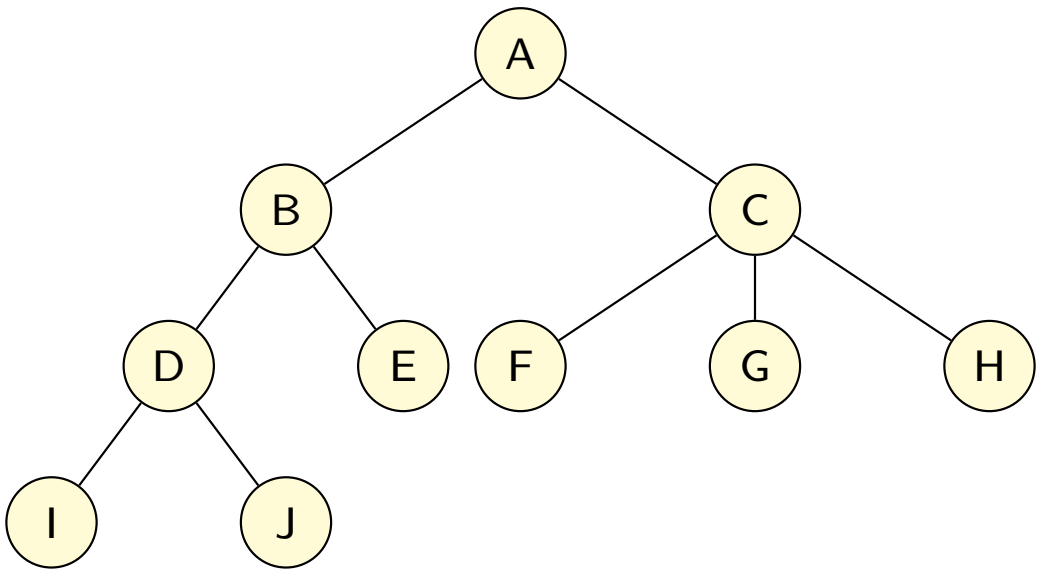


open: E, F, G, H, I, J

Breadth-first Search

Breadth-first search expands nodes in order of generation (FIFO).

$\rightsquigarrow$ e.g., open list as linked list or deque



- searches state space layer by layer
- always finds shallowest goal state first

Breadth-first Search: Tree Search or Graph Search?

Breadth-first search can be performed

- without duplicate elimination (as a tree search)
  $\rightsquigarrow$ BFS-Tree
- or with duplicate elimination (as a graph search)
  $\rightsquigarrow$ BFS-Graph

(BFS = breadth-first search).

$\rightsquigarrow$ We consider both variants.

BFS-Tree

BFS-Tree

breadth-first search without duplicate elimination:

BFS-Tree

```
if is_goal(init()):
    return ⟨⟩
open := new Deque
open.push_back(make_root_node())
while not open.is_empty():
    n := open.pop_front()
    for each ⟨a, s'⟩ ∈ succ(n.state):
        n' := make_node(n, a, s')
        if is_goal(s'):
            return extract_path(n')
        open.push_back(n')
return unsolvable
```

BFS-Graph

BFS-Graph (Breadth-First Search with Duplicate Elim.)

BFS-Graph

```
if is_goal(init()):
    return ⟨⟩
open := new Deque
open.push_back(make_root_node())
closed := new HashSet
closed.insert(init())
while not open.is_empty():
    n := open.pop_front()
    for each ⟨a, s'⟩ ∈ succ(n.state):
        n' := make_node(n, a, s')
        if is_goal(s'):
            return extract_path(n')
        if s' ∉ closed:
            closed.insert(s')
            open.push_back(n')
return unsolvable
```

Properties of Breadth-first Search

Properties of Breadth-first Search

Properties of Breadth-first Search:

- BFS-Tree is semi-complete, but not complete (cycles)
- BFS-Graph is complete. (avoids cycles)
- BFS (both variants) is optimal
  if all actions have the same cost (BFS incrementally checks longer solution paths),
  but not in general (BFS ignores costs).

BFS-Tree or BFS-Graph?

What is better, BFS-Tree or BFS-Graph?

BFS-Tree or BFS-Graph?

What is better, BFS-Tree or BFS-Graph?

advantages of BFS-Graph:

- complete
- much (!) more efficient if there are many duplicates

BFS-Tree or BFS-Graph?

What is better, BFS-Tree or BFS-Graph?

advantages of BFS-Graph:

- complete
- much (!) more efficient if there are many duplicates

advantages of BFS-Tree:

- simpler
- less overhead (time/space) if there are few duplicates

BFS-Tree or BFS-Graph?

What is better, BFS-Tree or BFS-Graph?

advantages of BFS-Graph:

- complete
- much (!) more efficient if there are many duplicates

advantages of BFS-Tree:

- simpler
- less overhead (time/space) if there are few duplicates

Conclusion

BFS-Graph is usually preferable, unless we know that there is a negligible number of duplicates in the given state space.

Summary

Summary

- blind search algorithm: use no information
  except black box interface of state space
- breadth-first search: expand nodes in order of generation
  - search state space layer by layer
  - can be tree search or graph search
  - complete as a graph search; semi-complete as a tree search
  - optimal with uniform action costs