

CS4618 Artificial Intelligence I

Today: More Uninformed Search

Thomas Jansen

October 3rd

Announcement: Practicals

Remember weekly assignments
discussed in 'practicals'

Announcement: Practicals

Remember weekly assignments
discussed in 'practicals'

Fact weekly 'practicals'
each **Wednesday, 10–11am**
in **WGB G21**
starting next week

Announcement: Practicals

Remember weekly assignments
discussed in 'practicals'

Fact weekly 'practicals'
each **Wednesday, 10–11am**
in **WGB G21**
starting next week

Weekly assignments handed out each Friday
due the next Wednesday
discussed one week later in 'practicals'

Announcement: Practicals

Remember weekly assignments
discussed in 'practicals'

Fact weekly 'practicals'
each **Wednesday, 10–11am**
in **WGB G21**
starting next week

Weekly assignments handed out each Friday
due the next Wednesday
discussed one week later in 'practicals'

Exception today
weekly assignment **not to be returned**
but to be discussed next Wednesday in 'practicals'

Plans for Today

- 1 Introduction
Overview
- 2 Uninformed Search
Introduction
Breadth-First Search
- 3 Depth-First Search
Depth-First Search
Variants of Depth-First Search
- 4 Summary
Summary & Take Home Message

Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



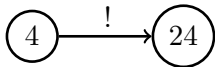
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each

4

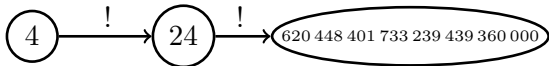
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



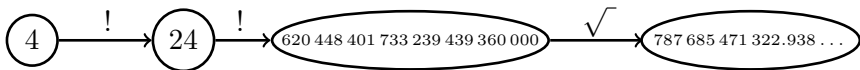
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



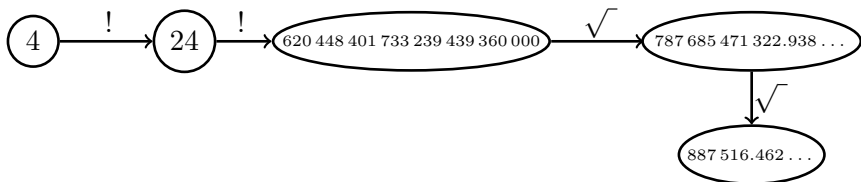
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



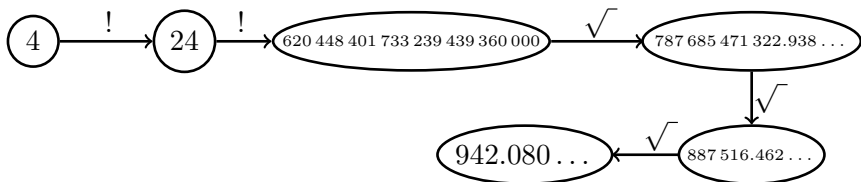
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



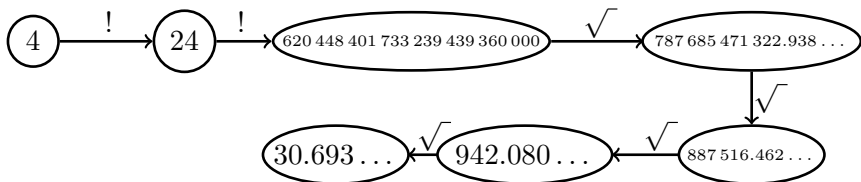
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



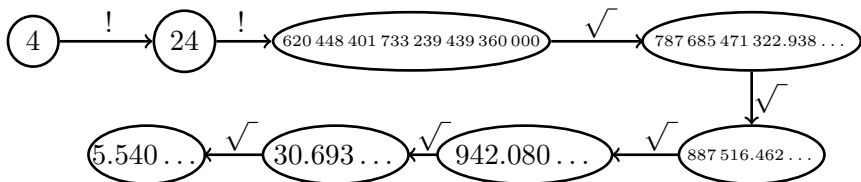
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



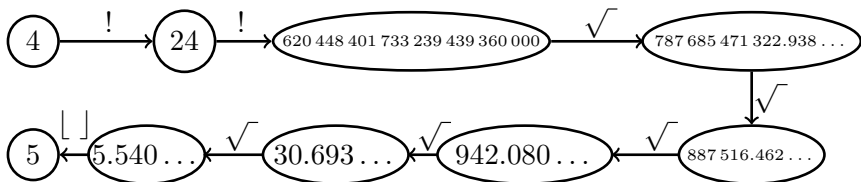
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $[\]$; cost 1 each



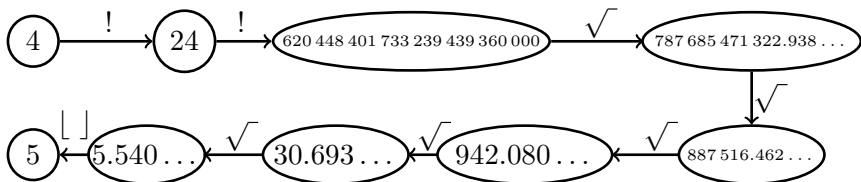
Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $\lfloor \rfloor$; cost 1 each



Knuth's Number Game

Remember initial state 4, goal state 5
actions $!$, $\sqrt{\quad}$, $\lfloor \rfloor$; cost 1 each



Cost 8

Uninformed Search

Assume the search algorithm knows

- the initial state,
- possible actions,
- how to find out which actions are applicable,
- how to apply the transition model,
- how to identify a goal state.

Uninformed Search

Assume the search algorithm knows

- the initial state,
- possible actions,
- how to find out which actions are applicable,
- how to apply the transition model,
- how to identify a goal state.

Notion blind search or uninformed search

Uninformed Search

Assume the search algorithm knows

- the initial state,
- possible actions,
- how to find out which actions are applicable,
- how to apply the transition model,
- how to identify a goal state.

Notion blind search or uninformed search

Observation search algorithms can **only** determine the order in which states are generated

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

```
function Breadth-First-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = queue with node as only element
  explored = ∅
```

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

function Breadth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = queue with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

function Breadth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = queue with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Remove(frontier)

 insert(node, explored)

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

function Breadth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = queue with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Remove(frontier)

 insert(node, explored)

 for each action in problem.actions(node.state) do

 child = problem.transition-model(node.state, action)

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

function Breadth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = queue with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Remove(frontier)

 insert(node, explored)

 for each action in problem.actions(node.state) do

 child = problem.transition-model(node.state, action)

 if problem.is-goal(child.state) then return child.state

Breadth-First Search

Idea Expand shallowest node in frontier first.

Breadth-First Search

function Breadth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = queue with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Remove(frontier)

 insert(node, explored)

 for each action in problem.actions(node.state) do

 child = problem.transition-model(node.state, action)

 if problem.is-goal(child.state) then return child.state

 if child \notin explored and child \notin frontier

 then insert(child, frontier)

Measuring the Performance of Search Algorithms

Measuring the Performance of Search Algorithms

- Completeness

Is the algorithm guaranteed to find a solution if one exists?

Measuring the Performance of Search Algorithms

- Completeness

Is the algorithm guaranteed to find a solution if one exists?

- Optimality

Is the algorithm guaranteed to find a solution with minimal path cost if one exists?

Measuring the Performance of Search Algorithms

- Completeness

Is the algorithm guaranteed to find a solution if one exists?

- Optimality

Is the algorithm guaranteed to find a solution with minimal path cost if one exists?

- Time Complexity

How long does the algorithm take to find a solution?

Measuring the Performance of Search Algorithms

- Completeness

Is the algorithm guaranteed to find a solution if one exists?

- Optimality

Is the algorithm guaranteed to find a solution with minimal path cost if one exists?

- Time Complexity

How long does the algorithm take to find a solution?

Remark Often proportional to the number of states generated.

Measuring the Performance of Search Algorithms

- **Completeness**
Is the algorithm guaranteed to find a solution if one exists?
- **Optimality**
Is the algorithm guaranteed to find a solution with minimal path cost if one exists?
- **Time Complexity**
How long does the algorithm take to find a solution?
Remark Often proportional to the number of states generated.
- **Space Complexity**
How much memory does the algorithm need to find a solution?

Measuring the Performance of Search Algorithms

- **Completeness**
Is the algorithm guaranteed to find a solution if one exists?
- **Optimality**
Is the algorithm guaranteed to find a solution with minimal path cost if one exists?
- **Time Complexity**
How long does the algorithm take to find a solution?
Remark Often proportional to the number of states generated.
- **Space Complexity**
How much memory does the algorithm need to find a solution?

Measuring with respect to what?

Measuring the Performance of Search Algorithms

- **Completeness**
Is the algorithm guaranteed to find a solution if one exists?
- **Optimality**
Is the algorithm guaranteed to find a solution with minimal path cost if one exists?
- **Time Complexity**
How long does the algorithm take to find a solution?
Remark Often proportional to the number of states generated.
- **Space Complexity**
How much memory does the algorithm need to find a solution?

Measuring with respect to what?

- **branching factor** $\hat{=}$ maximum number of applicable actions

Measuring the Performance of Search Algorithms

- **Completeness**
Is the algorithm guaranteed to find a solution if one exists?
- **Optimality**
Is the algorithm guaranteed to find a solution with minimal path cost if one exists?
- **Time Complexity**
How long does the algorithm take to find a solution?
Remark Often proportional to the number of states generated.
- **Space Complexity**
How much memory does the algorithm need to find a solution?

Measuring with respect to what?

- **branching factor** $\hat{=}$ maximum number of applicable actions
- **depth** $\hat{=}$ minimal number of steps to a goal state

Measuring the Performance of Search Algorithms

- **Completeness**

Is the algorithm guaranteed to find a solution if one exists?

- **Optimality**

Is the algorithm guaranteed to find a solution with minimal path cost if one exists?

- **Time Complexity**

How long does the algorithm take to find a solution?

Remark Often proportional to the number of states generated.

- **Space Complexity**

How much memory does the algorithm need to find a solution?

Measuring with respect to what?

- **branching factor** $\hat{=}$ maximum number of applicable actions
- **depth** $\hat{=}$ minimal number of steps to a goal state
- m $\hat{=}$ maximum length of path in the state space

Properties of Breadth-First Search

Properties of Breadth-First Search

- Completeness?

Properties of Breadth-First Search

- Completeness?

Observation complete ✓

Properties of Breadth-First Search

- Completeness?

Observation complete ✓

- Optimality?

Properties of Breadth-First Search

- Completeness?

Observation complete ✓

- Optimality?

Observation not necessarily

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

Properties of Breadth-First Search

- Completeness?

Observation complete ✓

- Optimality?

Observation not necessarily

Observation optimal if cost is non-decreasing function of depth (in particular for uniform costs)

- Time Complexity?

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Observe

$$1 + b + b^2 + \dots + b^{d-1} < N \leq 1 + b + b^2 + \dots + b^{d-1} + b^d$$

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Observe

$$1 + b + b^2 + \dots + b^{d-1} < N \leq 1 + b + b^2 + \dots + b^{d-1} + b^d$$

$$(b^d - 1)/(b - 1) < N < b^{d+1}/(b - 1)$$

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Observe

$$1 + b + b^2 + \dots + b^{d-1} < N \leq 1 + b + b^2 + \dots + b^{d-1} + b^d$$

$$(b^d - 1)/(b - 1) < N < b^{d+1}/(b - 1)$$

- **Space Complexity?**

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Observe

$$1 + b + b^2 + \dots + b^{d-1} < N \leq 1 + b + b^2 + \dots + b^{d-1} + b^d$$

$$(b^d - 1)/(b - 1) < N < b^{d+1}/(b - 1)$$

- **Space Complexity?**

Assume as for time complexity

Properties of Breadth-First Search

- **Completeness?**

Observation complete ✓

- **Optimality?**

Observation **not necessarily**

Observation **optimal** if cost is non-decreasing function of depth (in particular for uniform costs)

- **Time Complexity?**

Assume uniform number of applicable actions b and solution at depth d

Let N = number of nodes created

Observe

$$1 + b + b^2 + \dots + b^{d-1} < N \leq 1 + b + b^2 + \dots + b^{d-1} + b^d$$

$$(b^d - 1)/(b - 1) < N < b^{d+1}/(b - 1)$$

- **Space Complexity?**

Assume as for time complexity

Observe $(b^{d-1} - 1)/(b - 1) < S \leq b^d/(b - 1)$

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| <u>depth</u> | <u>nodes</u> | <u>time</u> | <u>memory</u> |
|--------------|--------------|-------------|---------------|
|--------------|--------------|-------------|---------------|

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|-------|----------|--------|
| 2 | 110 | .000011s | 107KB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|-------|----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|--------|----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|--------|----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|-----------|----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |
| 10 | 10^{10} | 3h | 10TB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node,
0.000001s per node

| depth | nodes | time | memory |
|-------|-----------|----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |
| 10 | 10^{10} | 3h | 10TB |
| 12 | 10^{12} | 13 days | 1PB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node, 0.000001s per node

| depth | nodes | time | memory |
|-------|-----------|-----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |
| 10 | 10^{10} | 3h | 10TB |
| 12 | 10^{12} | 13 days | 1PB |
| 14 | 10^{14} | 3.5 years | 99PB |

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node, 0.000001s per node

| depth | nodes | time | memory |
|-------|-----------|-----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |
| 10 | 10^{10} | 3h | 10TB |
| 12 | 10^{12} | 13 days | 1PB |
| 14 | 10^{14} | 3.5 years | 99PB |

Observations

- memory **more critical** than time

Time and Space Complexity for Breadth-First Search

Example with branching factor 10, 1000 bytes per node, 0.000001s per node

| depth | nodes | time | memory |
|-------|-----------|-----------|--------|
| 2 | 110 | .000011s | 107KB |
| 4 | 11110 | .0011s | 10.6MB |
| 6 | 10^6 | 1.1s | 1GB |
| 8 | 10^8 | 2min | 103GB |
| 10 | 10^{10} | 3h | 10TB |
| 12 | 10^{12} | 13 days | 1PB |
| 14 | 10^{14} | 3.5 years | 99PB |

Observations

- memory **more critical** than time
- only problems with solutions at small depth feasible

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Observation breadth-first search **optimal**

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Observation breadth-first search **optimal**
since it always expands shallowest node first

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Observation breadth-first search **optimal**
since it always expands shallowest node first

Can we have optimal performance for arbitrary cost functions?

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Observation breadth-first search **optimal**
since it always expands shallowest node first

Can we have optimal performance for arbitrary cost functions?

Yes if we take cost into account in deciding about ordering

Uniform-Cost Search

Consider breadth-first search for problem with uniform costs

Observation breadth-first search **optimal**
since it always expands shallowest node first

Can we have optimal performance for arbitrary cost functions?

Yes if we take cost into account in deciding about ordering
 \rightsquigarrow order in queue according to cost (\rightsquigarrow priority queue)
 and be careful when re-discovering nodes

Uniform-Cost Search Algorithm

Uniform-Cost Search Algorithm

Uniform-Cost Search

```
function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
```

Uniform-Cost Search Algorithm

Uniform-Cost Search

```
function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
```

Uniform-Cost Search Algorithm

Uniform-Cost Search

```

function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
    node = Remove(frontier)
    insert(node, explored)
  
```

Uniform-Cost Search Algorithm

Uniform-Cost Search

```

function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
    node = Remove(frontier)
    insert(node, explored)
    for each action in problem.actions(node.state) do
      child = problem.transition-model(node.state, action)
  
```

Uniform-Cost Search Algorithm

Uniform-Cost Search

```

function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
    node = Remove(frontier)
    insert(node, explored)
    for each action in problem.actions(node.state) do
      child = problem.transition-model(node.state, action)
      if problem.is-goal(child.state) then return child.state
  
```

Uniform-Cost Search Algorithm

Uniform-Cost Search

```

function Uniform-Cost-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = priority queue with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
    node = Remove(frontier)
    insert(node, explored)
    for each action in problem.actions(node.state) do
      child = problem.transition-model(node.state, action)
      if problem.is-goal(child.state) then return child.state
      if child ∉ explored und child ∉ frontier
        then insert(child, frontier) (updating cost if necessary)
  
```

Optimality of Uniform-Cost Search

Optimality of Uniform-Cost Search

Key Observation node chosen for expansion has 'correct' cost

Optimality of Uniform-Cost Search

Key Observation node chosen for expansion has 'correct' cost

Consequence minimal cost solution found first

Optimality of Uniform-Cost Search

Key Observation node chosen for expansion has 'correct' cost

Consequence minimal cost solution found first

Observation time and memory requirement **unbounded**
in comparison to breadth-first search

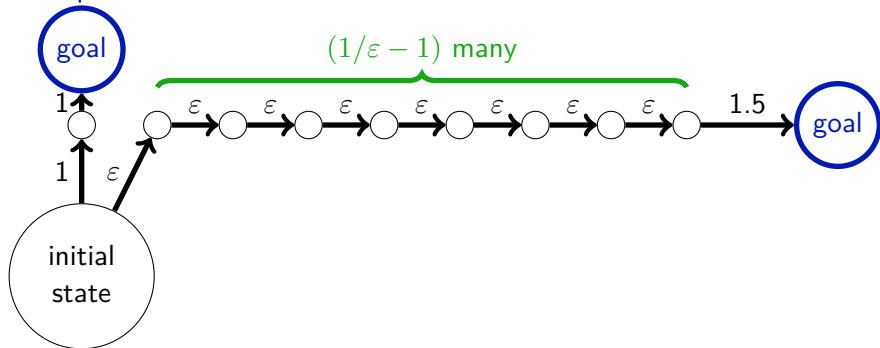
Optimality of Uniform-Cost Search

Key Observation node chosen for expansion has 'correct' cost

Consequence minimal cost solution found first

Observation time and memory requirement **unbounded**
in comparison to breadth-first search

Example



Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

```
function Depth-First-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = stack with node as only element
  explored = ∅
```

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

function Depth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = stack with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

function Depth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = stack with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Pop(frontier)

 insert(node, explored)

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

```

function Depth-First-Search(problem) returns solution or failure
  node = problem.initial-state
  if problem.is-goal(node.state) then return node.state
  frontier = stack with node as only element
  explored = ∅
  Repeat
    If Empty?(frontier) then return failure
    node = Pop(frontier)
    insert(node, explored)
    for each action in problem.actions(node.state) do
      child = problem.transition-model(node.state, action)
  
```

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

function Depth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = stack with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Pop(frontier)

 insert(node, explored)

 for each action in problem.actions(node.state) do

 child = problem.transition-model(node.state, action)

 if problem.is-goal(child.state) then return child.state

Depth-First Search

Idea Expand deepest node in frontier first.

Depth-First Search

function Depth-First-Search(**problem**) returns solution or failure

node = problem.initial-state

if problem.is-goal(node.state) then return node.state

frontier = stack with node as only element

explored = \emptyset

Repeat

 If Empty?(frontier) then return failure

 node = Pop(frontier)

 insert(node, explored)

 for each action in problem.actions(node.state) do

 child = problem.transition-model(node.state, action)

 if problem.is-goal(child.state) then return child.state

 if child \notin explored und child \notin frontier

 then Push(child, frontier)

Comparing Depth-First and Breadth-First Search

Comparing Depth-First and Breadth-First Search

- Completeness?

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

Depth-First

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no

Depth-First

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no

Depth-First

in general no

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no

with uniform cost yes

Depth-First

in general no

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no

with uniform cost yes

Depth-First

in general no

with uniform cost no

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- **Time Complexity?**

Comparing Depth-First and Breadth-First Search

- **Completeness?**

Breadth-First

yes

Depth-First

no (see Knuth's game)

- **Optimality?**

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- **Time Complexity?**

Breadth-First

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

$$d \leq N < b^{d+1}$$

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

$$d \leq N < b^{d+1}$$

- Space Complexity?

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

$$d \leq N < b^{d+1}$$

- Space Complexity?

Breadth-First

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

$$d \leq N < b^{d+1}$$

- Space Complexity?

Breadth-First

$$\frac{b^{d-1} - 1}{b - 1} < S < \frac{b^d}{b - 1}$$

Depth-First

Comparing Depth-First and Breadth-First Search

- Completeness?

Breadth-First

yes

Depth-First

no (see Knuth's game)

- Optimality?

Breadth-First

in general no
with uniform cost yes

Depth-First

in general no
with uniform cost no

- Time Complexity?

Breadth-First

$$\frac{b^d - 1}{b - 1} < N < \frac{b^{d+1}}{b - 1}$$

Depth-First

$$d \leq N < b^{d+1}$$

- Space Complexity?

Breadth-First

$$\frac{b^{d-1} - 1}{b - 1} < S < \frac{b^d}{b - 1}$$

Depth-First

$$b \cdot d \leq S < b \cdot m$$

Backtracking

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

- generate new states and explore
instead of generating all and then exploring

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

- generate new states and explore
instead of generating all and then exploring
- create next state as modification of current state and un-do
instead of copying or creating new from scratch

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

- generate new states and explore instead of generating all and then exploring
- create next state as modification of current state and un-do instead of copying or creating new from scratch

Observation

- reduces space complexity to m

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

- generate new states and explore instead of generating all and then exploring
- create next state as modification of current state and un-do instead of copying or creating new from scratch

Observation

- reduces space complexity to m
- requires capability of modifying and restoring states

Backtracking

Observation backtracking $\hat{=}$ modified depth-first search
(made more space efficient)

- generate new states and explore instead of generating all and then exploring
- create next state as modification of current state and un-do instead of copying or creating new from scratch

Observation

- reduces space complexity to m
- requires capability of modifying and restoring states
- requires capability of creating 'next' state

Depth-Limited Search

Idea limit search to depth l

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

```
function Depth-Limited-Search(problem, limit) returns solution or failure/cutoff
  return Recursive-DLS(make-node(problem.initial-state), problem, limit)
```

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
 return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

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

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
 return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

function Recursive-DLS(**node**, **problem**, **limit**) returns solution or failure/cutoff
 if **problem**.is-goal(**node**.state) then return **node**.state

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
 return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

function Recursive-DLS(**node**, **problem**, **limit**) returns solution or failure/cutoff
 if **problem**.is-goal(**node**.state) then return **node**.state
 else if **limit**=0 then return cutoff

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

```
function Depth-Limited-Search(problem, limit) returns solution or failure/cutoff
  return Recursive-DLS(make-node(problem.initial-state), problem, limit)
```

```
function Recursive-DLS(node, problem, limit) returns solution or failure/cutoff
  if problem.is-goal(node.state) then return node.state
  else if limit=0 then return cutoff
  else
    cutoff-occured = false
    foreach action in problem.actions(node.state) do
      child = problem.transition-model(node.state, action)
```

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
 return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

function Recursive-DLS(**node**, **problem**, **limit**) returns solution or failure/cutoff
 if **problem**.is-goal(**node**.state) then return **node**.state
 else if **limit**=0 then return cutoff
 else

 cutoff-occured = false

 foreach action in **problem**.actions(**node**.state) do

 child = **problem**.transition-model(**node**.state, action)

 result = Recursive-DLS(child, **problem**, limit - 1)

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

function Recursive-DLS(**node**, **problem**, **limit**) returns solution or failure/cutoff
if **problem**.is-goal(**node**.state) then return **node**.state
else if **limit**=0 then return cutoff
else

 cutoff-occured = false

 foreach action in **problem**.actions(**node**.state) do

 child = **problem**.transition-model(**node**.state, action)

 result = Recursive-DLS(child, **problem**, limit - 1)

 if result=cutoff then cutoff-occured=true

 else if result \neq failure then return result

Depth-Limited Search

Idea limit search to depth l

Remark general Depth-First Search $\hat{=} l = \infty$

Depth-Limited Search

function Depth-Limited-Search(**problem**, **limit**) returns solution or failure/cutoff
 return Recursive-DLS(make-node(**problem**.initial-state), **problem**, **limit**)

function Recursive-DLS(**node**, **problem**, **limit**) returns solution or failure/cutoff
 if **problem**.is-goal(**node**.state) then return **node**.state
 else if **limit**=0 then return cutoff
 else

 cutoff-occured = false

 foreach action in **problem**.actions(**node**.state) do

 child = **problem**.transition-model(**node**.state, action)

 result = Recursive-DLS(child, **problem**, limit - 1)

 if result=cutoff then cutoff-occured=true

 else if result \neq failure then return result

 if cutoff-occured then return cutoff else return failure

On Depth-Limited Search

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

- Time Complexity

$$O(b^l)$$

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

- Time Complexity

$$O(b^l)$$

Observation cutoff due to reaching the limit
 additional source of **incompleteness**

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

- Time Complexity

$$O(b^l)$$

Observation cutoff due to reaching the limit
 additional source of **incompleteness**

How can we set the depth limit?

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

- Time Complexity

$$O(b^l)$$

Observation cutoff due to reaching the limit
 additional source of **incompleteness**

How can we set the depth limit?

- problem properties need to be taken into account

On Depth-Limited Search

- Space Complexity

$$\leq b \cdot l$$

- Time Complexity

$$O(b^l)$$

Observation cutoff due to reaching the limit
 additional source of **incompleteness**

How can we set the depth limit?

- problem properties need to be taken into account
- **useful diameter** $\hat{=}$ largest distance between two states in state space

Iterative Deepening Depth-First Search

- Idea** apply Depth-Limited Search with minimal feasible depth limit increasing the limit as needed

Iterative Deepening Depth-First Search

Idea apply Depth-Limited Search with minimal feasible depth limit increasing the limit as needed

Iterative Deepening Depth-First Search

function Iterative-Deepening-Search(**problem**)
returns solution or failure/cutoff

Iterative Deepening Depth-First Search

Idea apply Depth-Limited Search with minimal feasible depth limit increasing the limit as needed

Iterative Deepening Depth-First Search

function Iterative-Deepening-Search(**problem**)

returns solution or failure/cutoff

depth = 0

Repeat

result = Depth-Limited-Search(**problem**, depth)

if result \neq cutoff then return result

Iterative Deepening Depth-First Search

Idea apply Depth-Limited Search with minimal feasible depth limit increasing the limit as needed

Iterative Deepening Depth-First Search

function Iterative-Deepening-Search(**problem**)

returns solution or failure/cutoff

depth = 0

Repeat

result = Depth-Limited-Search(**problem**, depth)

if result \neq cutoff then return result

depth = depth + 1

About Iterative Deepening Depth-First Search

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

1
⏟

level 0 at limit 0

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

$$\begin{array}{ccccccc}
 & & \underbrace{1} & & & & \\
 & & \text{level 0 at limit 0} & & & & \\
 + & & & + & & & \\
 & & \underbrace{1} & & \underbrace{b} & & \\
 & & \text{level 0 at limit 1} & & \text{level 1 at limit 1} & &
 \end{array}$$

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

$$\begin{array}{ccccccc}
 & \underbrace{1} & & & & & \\
 \text{level 0 at limit 0} & & & & & & \\
 + & \underbrace{1} & + & \underbrace{b} & & & \\
 \text{level 0 at limit 1} & \text{level 1 at limit 1} & & & & & \\
 + & \underbrace{1} & + & \underbrace{b} & + & \underbrace{b^2} & \\
 \text{level 0 at limit 2} & \text{level 1 at limit 2} & \text{level 2 at limit 2} & & & &
 \end{array}$$

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

$$\begin{array}{ccccccc}
 & \underbrace{1} & & & & & \\
 \text{level 0 at limit 0} & & & & & & \\
 + & \underbrace{1} & + & \underbrace{b} & & & \\
 \text{level 0 at limit 1} & \text{level 1 at limit 1} & & & & & \\
 + & \underbrace{1} & + & \underbrace{b} & + & \underbrace{b^2} & + \dots \\
 \text{level 0 at limit 2} & \text{level 1 at limit 2} & \text{level 2 at limit 2} & & & &
 \end{array}$$

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

$$\begin{aligned}
 & \underbrace{1}_{\text{level 0 at limit 0}} \\
 + & \underbrace{1}_{\text{level 0 at limit 1}} + \underbrace{b}_{\text{level 1 at limit 1}} \\
 + & \underbrace{1}_{\text{level 0 at limit 2}} + \underbrace{b}_{\text{level 1 at limit 2}} + \underbrace{b^2}_{\text{level 2 at limit 2}} + \dots \\
 = & \sum_{l=0}^{d-1} \underbrace{(d-l)}_{\text{repetitions}} \cdot b^l
 \end{aligned}$$

About Iterative Deepening Depth-First Search

Observation 'shallow' levels are explored **multiple** times

How costly is this obvious wastefulness?

Consider problem with solution at depth d
and uniform branching factor b

Remember BFS and DFS $O(b^{d+1})$

$$\begin{aligned}
 & \underbrace{1}_{\text{level 0 at limit 0}} \\
 + & \underbrace{1}_{\text{level 0 at limit 1}} + \underbrace{b}_{\text{level 1 at limit 1}} \\
 + & \underbrace{1}_{\text{level 0 at limit 2}} + \underbrace{b}_{\text{level 1 at limit 2}} + \underbrace{b^2}_{\text{level 2 at limit 2}} + \dots \\
 = & \sum_{l=0}^{d-1} \underbrace{(d-l)}_{\text{repetitions}} \cdot b^l = \dots = O(b^{d+1})
 \end{aligned}$$

Bidirectional Search

- Idea** start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Remember time complexity $\approx b^d$ for DFS (and BFS)

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Remember time complexity $\approx b^d$ for DFS (and BFS)

Observe both searches stop at depth $d/2$
and $b^{d/2} + b^{d/2} \ll b^d$

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Remember time complexity $\approx b^d$ for DFS (and BFS)

Observe both searches stop at depth $d/2$
 and $b^{d/2} + b^{d/2} \ll b^d$

Requirements

- goal known

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Remember time complexity $\approx b^d$ for DFS (and BFS)

Observe both searches stop at depth $d/2$
and $b^{d/2} + b^{d/2} \ll b^d$

Requirements

- goal known
- backwards search feasible (need to be able to 'reverse actions')

Bidirectional Search

Idea start **two** searches,
one from the initial state and
one from a goal state and **search** until the frontiers intersect

Why does this make sense?

Remember time complexity $\approx b^d$ for DFS (and BFS)

Observe both searches stop at depth $d/2$
 and $b^{d/2} + b^{d/2} \ll b^d$

Requirements

- goal known
- backwards search feasible (need to be able to 'reverse actions')

Remark 'dummy goal' **useful** in case of multiple goals

Comparing Algorithms for Uninformed Search

Comparing Algorithms for Uninformed Search

| Algorithm | | | | |
|---------------------|--|--|--|--|
| Breadth-First | | | | |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|---------------------|--------------|------------|------|-------|
| Breadth-First | | | | |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|---------------------|--------------------------|------------|------|-------|
| Breadth-First | yes (for finite b) | | | |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|---------------------|--------------------------|-------------------------|------|-------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | | |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--------------------------|-------------------------|---------------|-------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--------------------------|-------------------------|---------------|---------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|---------------|---------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|---------------|---------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|---------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | $O(b \cdot d)$ |
| Bidirectional | | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|-------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | $O(b \cdot d)$ |
| Bidirectional | yes (for finite b , with BFS) | | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|--------------------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | $O(b \cdot d)$ |
| Bidirectional | yes (for finite b , with BFS) | yes (for uni. costs, with BFS) | | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|--------------------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | $O(b \cdot d)$ |
| Bidirectional | yes (for finite b , with BFS) | yes (for uni. costs, with BFS) | $O(b^{d/2})$ | |

Comparing Algorithms for Uninformed Search

| Algorithm | Completeness | Optimality | Time | Space |
|------------------------|--|--------------------------------------|----------------------|----------------------|
| Breadth-First | yes (for finite b) | yes (for uni. costs) | $\Theta(b^d)$ | $\Theta(b^d)$ |
| Uniform-Cost | yes (for finite b , cost > 0) | yes | (depends on cost) | (depends on cost) |
| Depth-First | no | no | $O(b^m)$ | $O(b \cdot m)$ |
| Depth-Limited | no | no | $O(b^l)$ | $O(b \cdot l)$ |
| Iterative Deepening | yes (for finite b) | yes (for uni. costs) | $O(b^d)$ | $O(b \cdot d)$ |
| Bidirectional | yes (for finite b , with BFS) | yes (for uni. costs, with BFS) | $O(b^{d/2})$ | $O(b^{d/2})$ |

Summary & Take Home Message

Things to remember

Summary & Take Home Message

Things to remember

- breadth-first search

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search
- depth-first search and variants
 - depth-limited search
 - iterative deepening depth-first search
 - bidirectional search

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search
- depth-first search and variants
 - depth-limited search
 - iterative deepening depth-first search
 - bidirectional search

Take Home Message

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search
- depth-first search and variants
 - depth-limited search
 - iterative deepening depth-first search
 - bidirectional search

Take Home Message

- Trade-offs between space, time, and other properties make it worthwhile knowing (and applying) different kinds of algorithms for uninformed search.

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search
- depth-first search and variants
 - depth-limited search
 - iterative deepening depth-first search
 - bidirectional search

Take Home Message

- Trade-offs between space, time, and other properties make it worthwhile knowing (and applying) different kinds of algorithms for uninformed search.
- DFS is vastly more space-efficient than BFS.

Summary & Take Home Message

Things to remember

- breadth-first search
- uniform-cost search
- depth-first search and variants
 - depth-limited search
 - iterative deepening depth-first search
 - bidirectional search

Take Home Message

- Trade-offs between space, time, and other properties make it worthwhile knowing (and applying) different kinds of algorithms for uninformed search.
- DFS is vastly more space-efficient than BFS.
- Even in the restricted scenario of uninformed search clever search algorithms can be implemented.