

Most materials in this chapter are based on materials from <https://ai.berkeley.edu>



C1

Search

COMP3270
Introduction to Artificial Intelligence



1

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 1: 01_search.pptx.
- Canonical visual layer: 01_search.pdf.

C1: Search

Slide 1 of 193 · Initial

Chapter 1: Search Roadmap

2

Chapter Roadmap Define the Problem Abstraction, states, actions, goals, and costs Find a Solution Uninformed, informed, local, and constraint-based search

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 2: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Chapter 1: Search Roadmap

Define the Problem

Abstraction, states, actions, goals, and costs

2

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 2:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Chapter Roadmap Define the Problem Abstraction, states, actions, goals, and costs Find a Solution Uninformed, informed, local, and constraint-based search

Slide 2 of 193 · Reveal 1

Compe

Chapter 1: Search Roadmap

Define the Problem

Abstraction, states, actions, goals, and costs

Find a Solution

Uninformed, informed, local, and constraint-based search

2

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 2:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Chapter Roadmap Define the Problem Abstraction, states, actions, goals, and costs Find a Solution Uninformed, informed, local, and constraint-based search

Slide 2 of 193 · Reveal 2

Compe

Chapter 1: Search Roadmap

Define the Problem

Abstraction, states, actions, goals, and costs

Find a Solution

Uninformed, informed, local, and constraint-based search

Compete Against an Opponent

Minimax, pruning, and uncertainty

2

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 2:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Chapter Roadmap Define the Problem Abstraction, states, actions, goals, and costs Find a Solution Uninformed, informed, local, and constraint-based search Compe

Slide 2 of 193 · Reveal 3

Search Looks Ahead: Which Actions Reach the Goal?



3

Search Looks Ahead: Which Actions Reach the Goal?

Slide 3 of 193 · Initial

Speaker notes

Instead of trying actions blindly in the real world, an agent can explore possible sequences first.

It asks: which sequence of actions reaches the goal?

But to look ahead, the agent needs a simpler model of the world. That is abstraction.

[Sources] - Recovered COMP3270 2024–25 source slide 3:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

To Look Ahead, Build a Simpler Model

Real World → Search Model



4

Speaker notes

The real world contains far more detail than an agent can usefully search.

To look ahead, we build a simpler model.

To Look Ahead, Build a Simpler Model

Slide 4 of 193 · Initial

To Look Ahead, Build a Simpler Model

Real World → Search Model



Abstraction keeps only what the decision needs

4

Speaker notes

This simplification is abstraction. It keeps only the information needed for the decision.

To Look Ahead, Build a Simpler Model

Slide 4 of 193 · Reveal 1

To Look Ahead, Build a Simpler Model

Real World → Search Model



Abstraction keeps only what the decision needs

States · Actions · Goal · Costs

4

Speaker notes

For a search problem, that normally means the states, the available actions, the goal, and the costs.

Once we have the model, the next question is how to search it.

To Look Ahead, Build a Simpler Model

Slide 4 of 193 · Reveal 2

Choose the Search Strategy to Match the Problem

5

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Initial

Speaker notes

We have a model. Now how should we search it?

The answer depends on what the problem gives us.

Choose the Search Strategy to Match the Problem

No Guidance → Uninformed Search

5

Speaker notes

With no guidance about where the goal lies, uninformed methods such as BFS, DFS, and uniform-cost search explore systematically.

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Reveal 1

Choose the Search Strategy to Match the Problem

No Guidance → Uninformed Search

A Helpful Estimate → Informed Search

5

Speaker notes

If we have a helpful estimate of how close a state is to the goal, informed methods such as greedy search and A* use that estimate.

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Reveal 2

Choose the Search Strategy to Match the Problem

No Guidance → Uninformed Search

A Helpful Estimate → Informed Search

Improve One Current State → Local Search

5

Speaker notes

If we only need to improve one current state, local search changes that state to make it better. Gradient descent is one example.

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Reveal 3

Choose the Search Strategy to Match the Problem

No Guidance → Uninformed Search

A Helpful Estimate → Informed Search

Improve One Current State → Local Search

Hard Constraints → Constraint Satisfaction

5

Speaker notes

If choices must satisfy hard rules, constraint satisfaction methods use those constraints to eliminate impossible choices. AC-3 is one example.

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Reveal 4

Choose the Search Strategy to Match the Problem

No Guidance → Uninformed Search

A Helpful Estimate → Informed Search

Improve One Current State → Local Search

Hard Constraints → Constraint Satisfaction

An Opponent → Adversarial Search

5

Speaker notes

If another agent is working against us, adversarial search plans for that opponent. Minimax and alpha–beta pruning are examples.

We will begin by defining a search problem precisely, then study these strategies in turn.

Choose the Search Strategy to Match the Problem

Slide 5 of 193 · Reveal 5

Defining a Search Problem

6

Defining a Search Problem

Slide 6 of 193 · Initial

Speaker notes

We now know there are several ways to search.

But none of them can begin until we define the problem precisely.

Defining a Search Problem

State Representation What information describes a situation?

6

Defining a Search Problem

Slide 6 of 193 · Reveal 1

Speaker notes

First, what information do we need to describe one situation? That information defines the state representation.

Defining a Search Problem

State Representation What information describes a situation?

Initial State Where does the agent begin?

6

Defining a Search Problem

Slide 6 of 193 · Reveal 2

Speaker notes

Second, which state does the agent begin from? That is the initial state.

Defining a Search Problem

State Representation	What information describes a situation?
Initial State	Where does the agent begin?
Actions and Transition Model	What can the agent do, and what changes?

6

Speaker notes

Third, what actions are available, and what new state does each action produce? Together, these are the actions and transition model.

Defining a Search Problem

Slide 6 of 193 · Reveal 3

Defining a Search Problem

State Representation	What information describes a situation?
Initial State	Where does the agent begin?
Actions and Transition Model	What can the agent do, and what changes?
Goal Test	How do we recognise success?

6

Speaker notes

Fourth, how do we recognise success? The goal test tells us whether a state is a goal state.

Defining a Search Problem

Slide 6 of 193 · Reveal 4

Defining a Search Problem

State Representation	What information describes a situation?
Initial State	Where does the agent begin?
Actions and Transition Model	What can the agent do, and what changes?
Goal Test	How do we recognise success?
Path Cost	How do we compare solutions?

6

Speaker notes

Fifth, how expensive is a route? Path cost lets us compare different sequences of actions.

When every action has the same cost and we only need any solution, the path cost may be left implicit. That is why the next N-Queens examples do not list it separately.

Defining a Search Problem

Slide 6 of 193 · Reveal 5

Defining a Search Problem

State Representation	What information describes a situation?
Initial State	Where does the agent begin?
Actions and Transition Model	What can the agent do, and what changes?
Goal Test	How do we recognise success?
Path Cost	How do we compare solutions?

A **solution** is a sequence of actions from the initial state to a goal state

6

Speaker notes

A solution is a sequence of actions that takes us from the start state to a goal state.

Next, we will define the same N-Queens puzzle in two different ways. Both definitions are valid, but they produce very different search spaces.

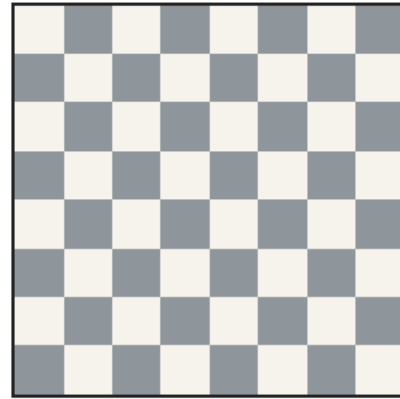
Defining a Search Problem

Slide 6 of 193 · Reveal 6

N-Queens Definition A: Place Anywhere, Check at the End

Start

Empty board



7

N-Queens Definition A: Place Anywhere, Check at the End

Slide 7 of 193 · Initial

Speaker notes

Let us apply the five-question checklist to N-Queens.

In Definition A, we begin with an empty board.

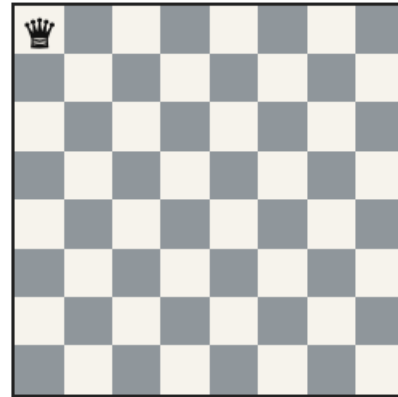
N-Queens Definition A: Place Anywhere, Check at the End

Start

Empty board

Action

Add a queen to any empty square



7

Speaker notes

An action adds one queen to any empty square. We do not check yet whether that queen attacks another queen.

N-Queens Definition A: Place Anywhere, Check at the End

Slide 7 of 193 · Reveal 1

N-Queens Definition A: Place Anywhere, Check at the End

Start

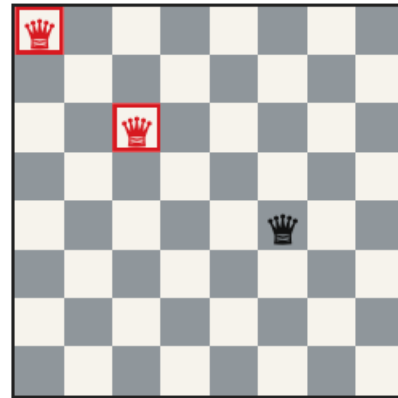
Empty board

Action

Add a queen to any empty square

State

Any arrangement of 0–n queens



7

Speaker notes

Continue applying that action. A state can therefore be any arrangement of zero to n queens, including arrangements in which queens attack each other.

Because our actions only add queens, an attacking partial board cannot later become a valid goal. It is already a dead end.

N-Queens Definition A: Place Anywhere, Check at the End

Slide 7 of 193 · Reveal 2

N-Queens Definition A: Place Anywhere, Check at the End

Start

Empty board

Action

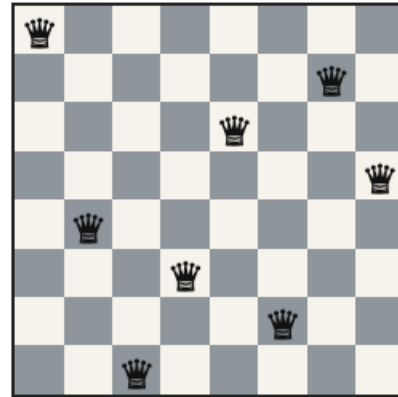
Add a queen to any empty square

State

Any arrangement of 0– n queens

Goal

n queens, none attacking



7

Speaker notes

The goal requires n queens on the board with no queen attacking another.

N-Queens Definition A: Place Anywhere, Check at the End

Slide 7 of 193 · Reveal 3

N-Queens Definition A: Place Anywhere, Check at the End

Start

Empty board

Action

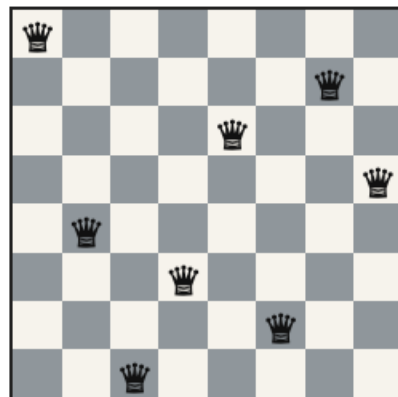
Add a queen to any empty square

State

Any arrangement of 0–n queens

Goal

n queens, none attacking



Attacking partial boards are still states

7

Speaker notes

The important consequence is that attacking partial boards are still states in Definition A.

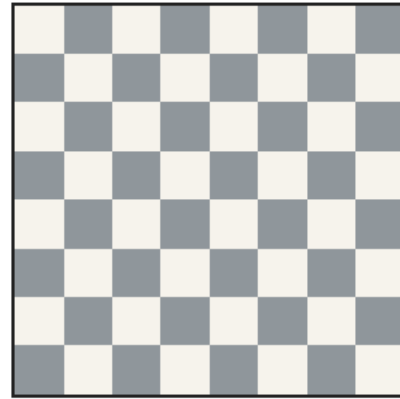
This definition makes the search consider many states that cannot lead to a solution. Can we rule those states out earlier?

N-Queens Definition A: Place Anywhere, Check at the End

Slide 7 of 193 · Reveal 4

N-Queens Definition B: Rule Out Bad States Early

Start
Empty board



8

Speaker notes

Definition B starts from the same empty board, but changes what counts as a state and which actions are allowed.

N-Queens Definition B: Rule Out Bad States Early

Slide 8 of 193 · Initial

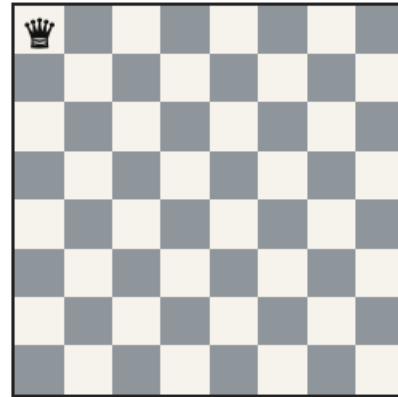
N-Queens Definition B: Rule Out Bad States Early

Start

Empty board

Action

Add one non-attacking queen to an empty column



8

Speaker notes

An action adds one queen to an empty column only when it does not attack any queen already on the board.

N-Queens Definition B: Rule Out Bad States Early

Slide 8 of 193 · Reveal 1

N-Queens Definition B: Rule Out Bad States Early

Start

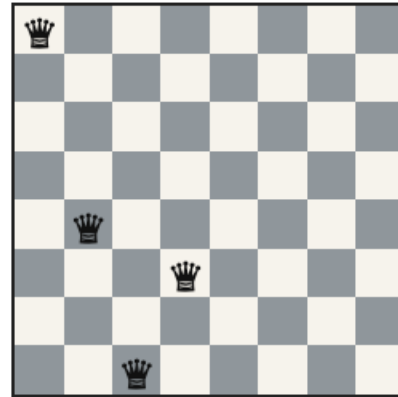
Empty board

Action

Add one non-attacking queen to an empty column

State

One queen per column, none attacking



8

Speaker notes

Every state has at most one queen per column, and no queen attacks another.

An invalid partial board is rejected immediately instead of becoming another state to search.

N-Queens Definition B: Rule Out Bad States Early

Slide 8 of 193 · Reveal 2

N-Queens Definition B: Rule Out Bad States Early

Start

Empty board

Action

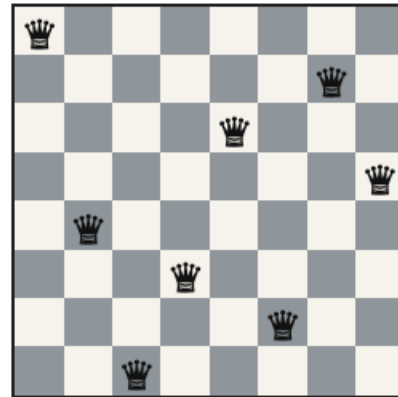
Add one non-attacking queen to an empty column

State

One queen per column, none attacking

Goal

n queens on the board



8

Speaker notes

The goal is reached when n queens are on the board. The non-attacking requirement is already guaranteed by the state and action definitions.

N-Queens Definition B: Rule Out Bad States Early

Slide 8 of 193 · Reveal 3

N-Queens Definition B: Rule Out Bad States Early

Start

Empty board

Action

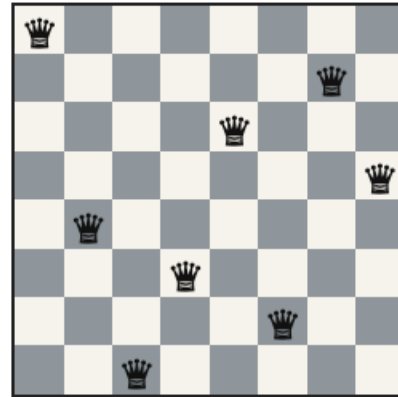
Add one non-attacking queen to an empty column

State

One queen per column, none attacking

Goal

n queens on the board



Invalid partial boards never enter the state space

8

Speaker notes

Invalid partial boards never enter the state space in Definition B.

The physical puzzle is unchanged. The abstraction changed, so the size of the search problem changed.

Which definition has the larger state space?

N-Queens Definition B: Rule Out Bad States Early

Slide 8 of 193 · Reveal 4

A State Space Contains Every Reachable State

Different problem definitions produce different state spaces

9

A State Space Contains Every Reachable State

Slide 9 of 193 · Initial

Speaker notes

A state space contains every distinct state that can be reached from the start by some sequence of actions.

We now have two abstractions of the same physical N-Queens puzzle.

A State Space Contains Every Reachable State

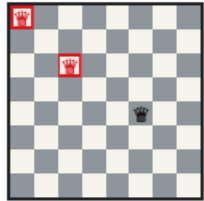
Different problem definitions produce different state spaces

TASK

Which definition has the larger state space?

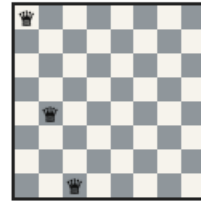
Definition A

Place anywhere. Check at the end.



Definition B

Rule out bad states early.



9

Speaker notes

Ask students which definition creates more states and to justify their answer before doing any calculation.

A State Space Contains Every Reachable State

Slide 9 of 193 · Reveal 1

Same Puzzle, Very Different State Spaces

SOLUTION

10

Same Puzzle, Very Different State Spaces

Slide 10 of 193 · Initial

Speaker notes

We will count the two state spaces separately before comparing them.

Same Puzzle, Very Different State Spaces

SOLUTION

Definition A · Count Every Placement

$$|S_A| = \sum_{k=0}^8 \binom{64}{k}$$

```
main.py
1 from math import comb
2
3 states = sum(
4     comb(64, k) for k in range(
5 )
6 print(states)
7
```

Download: comp3270.dirk.hi/h_queens_definition_a.py

10

Speaker notes

Definition A permits any arrangement of zero to eight queens on the sixty-four squares. For exactly k queens, we choose k squares, so the formula sums sixty-four choose k from zero to eight.

Run the program on the left. Python's comb function performs each combination calculation and produces 5,130,659,561 states.

Same Puzzle, Very Different State Spaces

Slide 10 of 193 · Reveal 1

Same Puzzle, Very Different State Spaces

SOLUTION

Definition A · Count Every Placement

$$|S_A| = \sum_{k=0}^8 \binom{64}{k}$$

```
main.py
1 from math import comb
2
3 states = sum(
4     comb(64, k) for k in range(
5 )
6 print(states)
7
```

Download: comp3270.dirk.hkh/queens_definition_a.py

Definition B · Keep Only Safe Placements

$$|S_B| = ?$$

```
main.py
1 def count(start=0, qs=()):
2     return 1 + sum(count(c + 1, qs +
3         for c in range(start, 8)
4         for r in range(8)
5         if all(r != y and
6             abs(c - x) != abs(r - y)
7             for x, y in qs))
8 print(count())
9
```

Download: comp3270.dirk.hkh/queens_definition_b.py

10

Speaker notes

Definition B counts only safe partial boards. Run the program on the right. It counts the current board, then recursively tries each safe queen in a later column.

Using later columns ensures that every distinct board is counted once. The result is 118,969 states.

Same Puzzle, Very Different State Spaces

Slide 10 of 193 · Reveal 2

Same Puzzle, Very Different State Spaces

SOLUTION

Definition A · Count Every Placement

$$|S_A| = \sum_{k=0}^8 \binom{64}{k}$$

```
main.py
1 from math import comb
2
3 states = sum(
4     comb(64, k) for k in range(
5 )
6 print(states)
7
```

Download: comp3270.dirk.hkh_queens_definition_a.py

Definition B · Keep Only Safe Placements

$$|S_B| = ?$$

```
main.py
1 def count(start=0, qs=()):
2     return 1 + sum(count(c + 1, qs +
3         for c in range(start, 8)
4         for r in range(8)
5         if all(r != y and
6             abs(c - x) != abs(r - y)
7             for x, y in qs))
8 print(count())
9
```

Download: comp3270.dirk.hkh_queens_definition_b.py

$$|S_A| \approx 43,126 \times |S_B|$$

10

Speaker notes

The comparison says that $|S_A|$ is about 43,126 times $|S_B|$.

The physical puzzle did not change. The abstraction changed, and that dramatically changed how much there is to search.

Same Puzzle, Very Different State Spaces

Slide 10 of 193 · Reveal 3

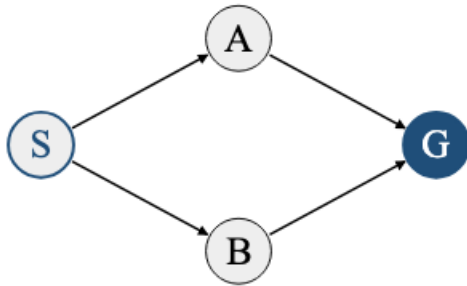
A Search Tree Records Action Sequences

State Space Graph

Each distinct state appears once

Search Tree

Each action sequence creates a tree node



11

Speaker notes

A state space graph contains each distinct state once. The graph on the left therefore contains one node for G.

A search tree records action sequences from the initial state. We will unfold this finite graph into a tree.

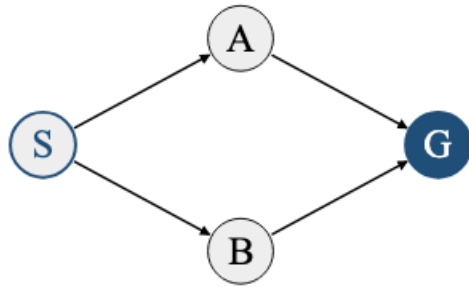
A Search Tree Records Action Sequences

Slide 11 of 193 · Initial

A Search Tree Records Action Sequences

State Space Graph

Each distinct state appears once



Search Tree

Each action sequence creates a tree node



11

Speaker notes

The root of the search tree represents the empty action sequence, so it contains the initial state S.

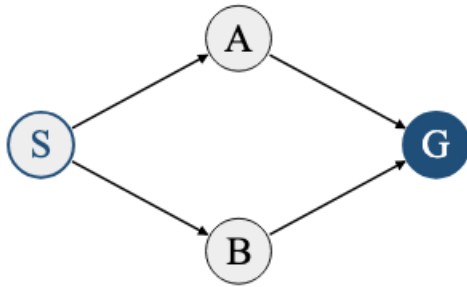
A Search Tree Records Action Sequences

Slide 11 of 193 · Reveal 1

A Search Tree Records Action Sequences

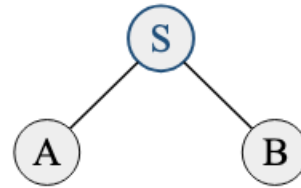
State Space Graph

Each distinct state appears once



Search Tree

Each action sequence creates a tree node



11

Speaker notes

Expanding S creates one child for the sequence leading to A and another for the sequence leading to B.

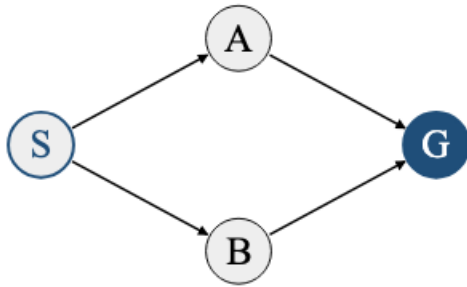
A Search Tree Records Action Sequences

Slide 11 of 193 · Reveal 2

A Search Tree Records Action Sequences

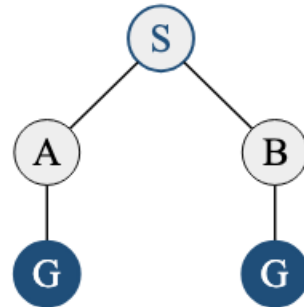
State Space Graph

Each distinct state appears once



Search Tree

Each action sequence creates a tree node



11

Speaker notes

Expanding A and B reaches G through two different action sequences. The state space graph contains one G, but the search tree contains two G nodes.

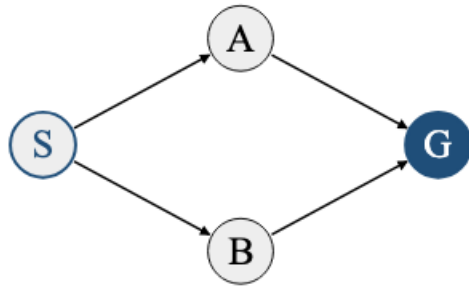
A Search Tree Records Action Sequences

Slide 11 of 193 · Reveal 3

A Search Tree Records Action Sequences

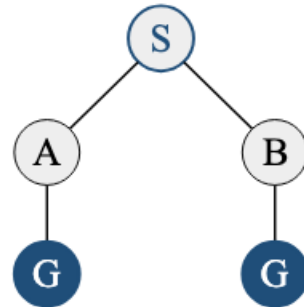
State Space Graph

Each distinct state appears once



Search Tree

Each action sequence creates a tree node



Different paths to G create different search-tree nodes

11

Speaker notes

This is the key distinction: tree nodes represent paths, not merely distinct states.

Next, students will construct a complete search tree for another finite graph.

A Search Tree Records Action Sequences

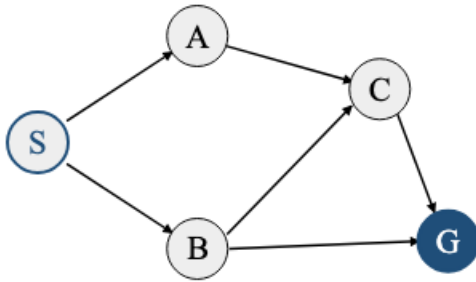
Slide 11 of 193 · Reveal 4

Draw the Search Tree

Starting at S, draw the complete search tree. How many times does G appear?

TASK

State Space Graph



12

Speaker notes

Give students the finite state space graph and ask them to draw the complete search tree from S.

Ask them to count how many times G appears. The graph has no cycles, so the complete tree is finite.

Draw the Search Tree

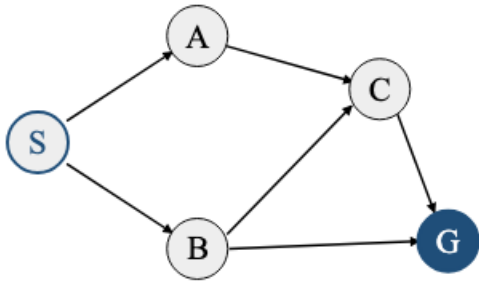
Slide 12 of 193 · Initial

Draw the Search Tree

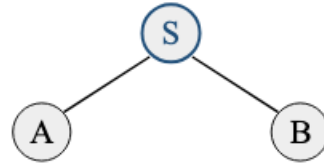
Starting at S, draw the complete search tree. How many times does G appear?

TASK

State Space Graph

**SOLUTION**

Search Tree



12

Draw the Search Tree

Slide 12 of 193 · Reveal 1

Speaker notes

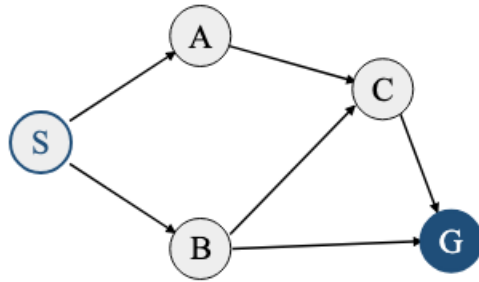
Start the tree at S and expand it to A and B.

Draw the Search Tree

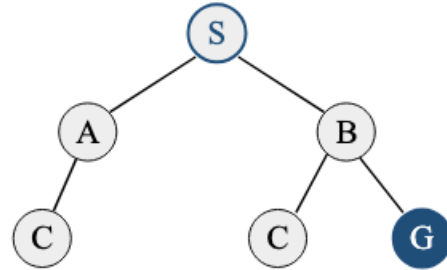
Starting at S, draw the complete search tree. How many times does G appear?

TASK**SOLUTION**

State Space Graph



Search Tree



12

Speaker notes

A reaches C. B reaches another C node and also reaches G directly. The same state C now appears twice because two action sequences reach it.

Draw the Search Tree

Slide 12 of 193 · Reveal 2

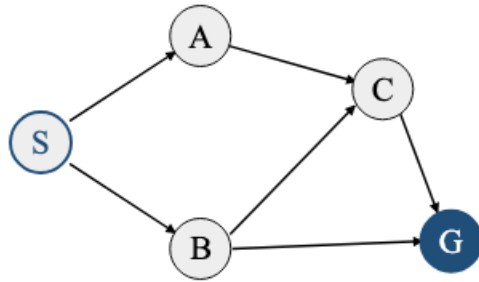
Draw the Search Tree

Starting at S, draw the complete search tree. How many times does G appear?

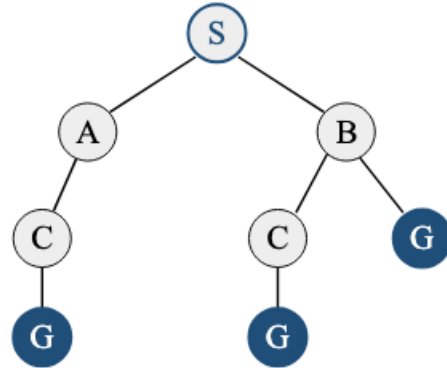
TASK

SOLUTION

State Space Graph



Search Tree



12

Speaker notes

Each C node reaches its own G node. Together with the direct path from B, G appears three times in the search tree.

Draw the Search Tree

Slide 12 of 193 · Reveal 3

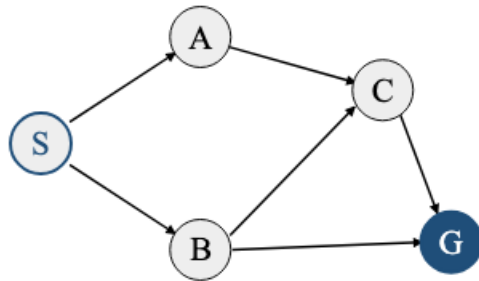
Draw the Search Tree

Starting at S, draw the complete search tree. How many times does G appear?

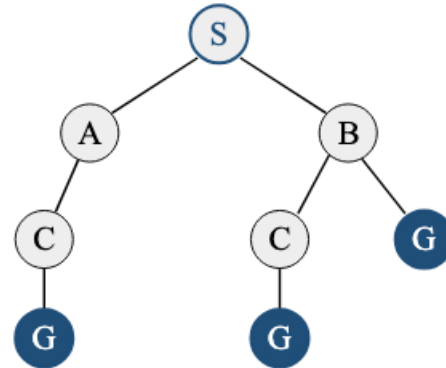
TASK

SOLUTION

State Space Graph



Search Tree



G appears three times because three action sequences reach it

12

Speaker notes

The state space graph has one G state, but the search tree has three G nodes because it records three distinct action sequences.

Draw the Search Tree

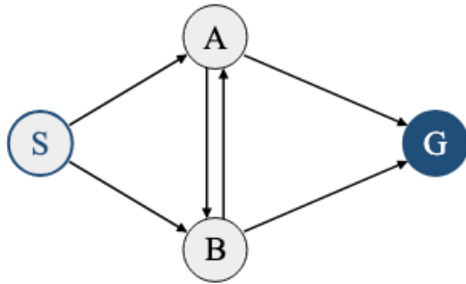
Slide 12 of 193 · Reveal 4

Trace the Search Tree

How many times does G appear in the complete search tree?

TASK

Cyclic State Space Graph



12

Speaker notes

Return to the original cyclic example. Ask students how many times G appears in the complete search tree.

The important clue is that A and B can lead back to one another.

Draw the Search Tree

Slide 12 of 193 · Reveal 5

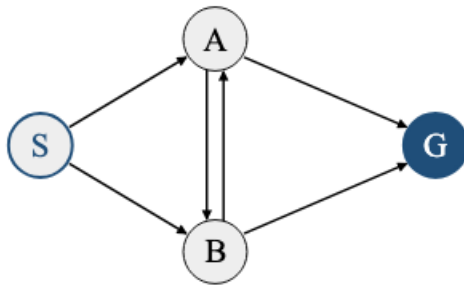
Trace the Search Tree

How many times does G appear in the complete search tree?

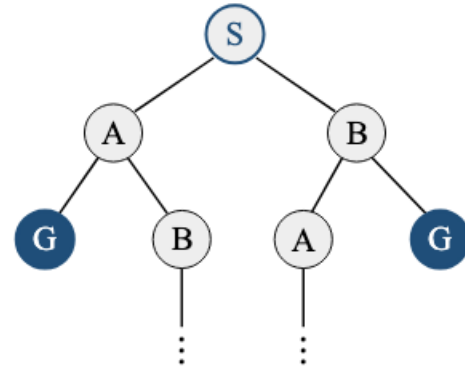
TASK

SOLUTION

Cyclic State Space Graph



Search Tree Prefix



Infinitely many: A and B can alternate any number of times before moving to G

12

Speaker notes

G appears infinitely many times. The search can alternate between A and B any number of times before taking an action to G.

The diagram shows only a finite prefix. The alternating branches continue without end, which motivates repeated-state checking in graph search.

Draw the Search Tree

Slide 12 of 193 · Reveal 6

Define the Romania Route Problem

Find a route from Arad to Bucharest

TASK

- State
- Start
- Actions and Transition
- Goal
- Path Cost



13

Speaker notes

We need a small running example for the search algorithms that follow.

Suppose we begin in Arad and want to reach Bucharest. The first column names the five parts of the search problem. Ask students to suggest each answer.

Define the Romania Route Problem

Define the Romania Route Problem

Find a route from Arad to Bucharest

TASK

- State
- Start
- Actions and Transition
- Goal
- Path Cost



13

Speaker notes

The A marker locates Arad in western Romania. This is our initial state.

Define the Romania Route Problem

Slide 13 of 193 · Reveal 1

Define the Romania Route Problem

Find a route from Arad to Bucharest

TASK

- State
- Start
- Actions and Transition
- Goal
- Path Cost



13

Speaker notes

The B marker locates Bucharest in the southeast. The search must connect these two locations through the road network.

Define the Romania Route Problem

Slide 13 of 193 · Reveal 2

Define the Romania Route Problem

Find a route from Arad to Bucharest

SOLUTION

State	Current city
Start	Arad
Actions and Transition	Choose a neighboring city and move there
Goal	Reach Bucharest
Path Cost	Sum of road distances



13

Speaker notes

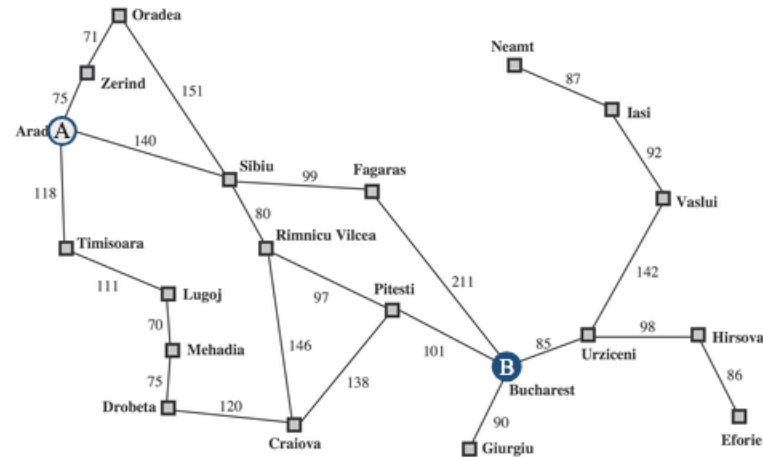
A state records the current city, and the initial state is Arad.

An action chooses a neighboring city. The transition model moves us there. Bucharest satisfies the goal test, and the path cost is the sum of the road distances.

Define the Romania Route Problem

Romania as a State-Space Graph

Cities are states · roads are transitions · distances are step costs (ignored for now)



14

Speaker notes

The geographic map is now abstracted into a weighted state space graph.

Each city is a state. Each road represents a legal transition, and its number is the step cost for taking that road.

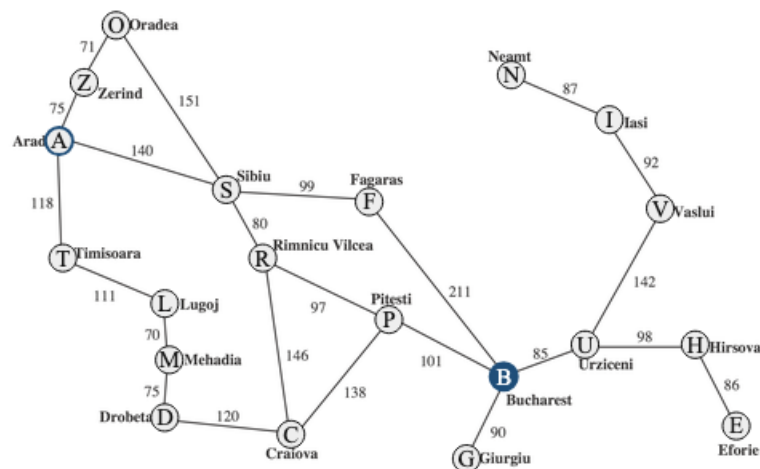
From this point onward, we use each city's initial as its state label: A for Arad and B for Bucharest.

Romania as a State-Space Graph

Slide 14 of 193 · Initial

Romania as a State-Space Graph

Cities are states · roads are transitions · distances are step costs (ignored for now)



14

Speaker notes

Reveal the initial used for every city. In this example, all city names have different first letters, so each initial can serve as a unique state identifier.

This is only a convenient shorthand for this graph. In general, state identifiers must be unique even when names share an initial.

Romania as a State-Space Graph

Slide 14 of 193 · Reveal 1

Expanding A Generates Three Successors



```

main.py
1  romania = {
2      'A': ['S', 'T', 'Z'],
3      'Z': ['A', 'O'], 'O': ['S', 'Z'],
4      'T': ['A', 'L'], 'L': ['M', 'T'],
5      'M': ['D', 'L'], 'D': ['C', 'M'],
6      'S': ['A', 'F', 'O', 'R'],
7      'R': ['C', 'P', 'S'],
8      'C': ['D', 'P', 'R'],
9      'F': ['B', 'S'],
10     'P': ['B', 'C', 'R'], 'B': []
11 }
12 print(romania['A'])
13

```

Download: comp3270.dirk.hk/roads_from_arad.py

15

Speaker notes

The dictionary stores connectivity using city initials. Each key is a state, and its value lists the neighboring states.

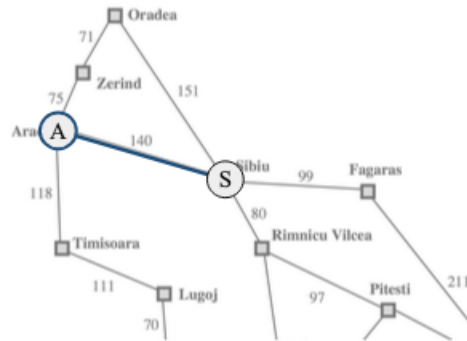
For now, ignore the road distances and focus on which successors can be generated from A.

The cropped graph keeps the Arad neighborhood visible beside the runnable code.

Expanding A Generates Three Successors

Slide 15 of 193 · Initial

Expanding A Generates Three Successors



```

1- main.py
2- romania = {
3-     'A': ['S', 'T', 'Z'],
4-     'Z': ['A', 'O'], 'O': ['S', 'Z'],
5-     'T': ['A', 'L'], 'L': ['M', 'T'],
6-     'M': ['D', 'L'], 'D': ['C', 'M'],
7-     'S': ['A', 'F', 'O', 'R'],
8-     'R': ['C', 'P', 'S'],
9-     'C': ['D', 'P', 'R'],
10-    'F': ['B', 'S'],
11-    'P': ['B', 'C', 'R'], 'B': []
12- }
13- print(romania['A'])

```

Download: comp3270.dirk.hk/roads_from_arad.py

15

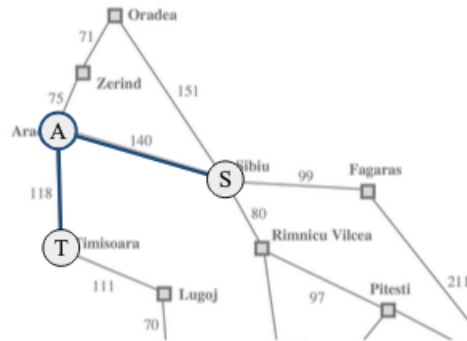
Speaker notes

The first entry in the list for A is S, so highlight the road from Arad to Sibiu and generate state S.

Expanding A Generates Three Successors

Slide 15 of 193 · Reveal 1

Expanding A Generates Three Successors



```

1- main.py
2- romania = {
3-     'A': ['S', 'T', 'Z'],
4-     'Z': ['A', 'O'], 'O': ['S', 'Z']
5-     'T': ['A', 'L'], 'L': ['M', 'T']
6-     'M': ['D', 'L'], 'D': ['C', 'M']
7-     'S': ['A', 'F', 'O', 'R'],
8-     'R': ['C', 'P', 'S'],
9-     'C': ['D', 'P', 'R'],
10    'F': ['B', 'S'],
11    'P': ['B', 'C', 'R'], 'B': []
12 }
13 print(romania['A'])

```

Download: comp3270.dirk.hk/roads_from_arad.py

15

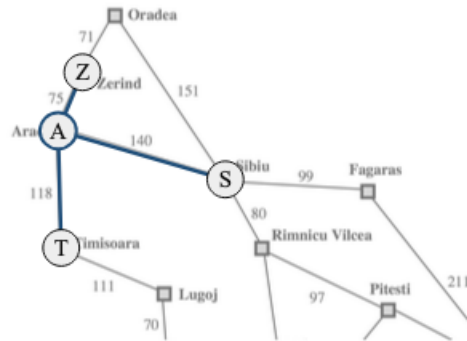
Speaker notes

The second successor is T for Timisoara.

Expanding A Generates Three Successors

Slide 15 of 193 · Reveal 2

Expanding A Generates Three Successors



```

1  romania = {
2      'A': ['S', 'T', 'Z'],
3      'Z': ['A', 'O'], 'O': ['S', 'Z'],
4      'T': ['A', 'L'], 'L': ['M', 'T'],
5      'M': ['D', 'L'], 'D': ['C', 'M'],
6      'S': ['A', 'F', 'O', 'R'],
7      'R': ['C', 'P', 'S'],
8      'C': ['D', 'P', 'R'],
9      'F': ['B', 'S'],
10     'P': ['B', 'C', 'R'], 'B': []
11 }
12 print(romania['A'])
13

```

Download: comp3270.dirk.hk/roads_from_arad.py

15

Speaker notes

The third successor is Z for Zerind. A, S, T, and Z use equal-sized state circles.

A is rendered after the road overlays, so the start state remains in the foreground. Run the example to confirm the output S, T, Z.

Expanding A Generates Three Successors

Slide 15 of 193 · Reveal 3

The Frontier Stores Nodes Waiting to Be Expanded

Frontier = generated nodes waiting to be expanded.

It is the search algorithm's current set of choices.

Initially: one generated node



16

Speaker notes

The frontier is the collection of generated nodes that have not yet been expanded. It is the set of choices currently available to the search algorithm.

Initially, only the root node has been generated. That node contains state A, so the frontier contains one node labelled A.

The Frontier Stores Nodes Waiting to Be Expanded

Slide 16 of 193 · Initial

The Frontier Stores Nodes Waiting to Be Expanded

Frontier = generated nodes waiting to be expanded.

It is the search algorithm's current set of choices.

Frontier after selection

Expand A



16

Speaker notes

The algorithm selects the node labelled A, removes it from the frontier, and expands it.

A selected node leaves the frontier because it is no longer waiting to be expanded.

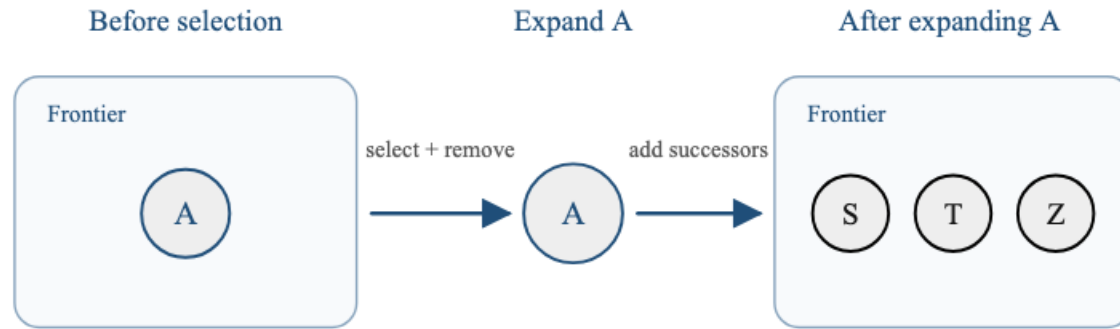
The Frontier Stores Nodes Waiting to Be Expanded

Slide 16 of 193 · Reveal 1

The Frontier Stores Nodes Waiting to Be Expanded

Frontier = generated nodes waiting to be expanded.

It is the search algorithm's current set of choices.



16

Speaker notes

The expansion generates the three children listed in the dictionary: S, T, and Z.

They are added to the frontier because each is now available for later expansion.

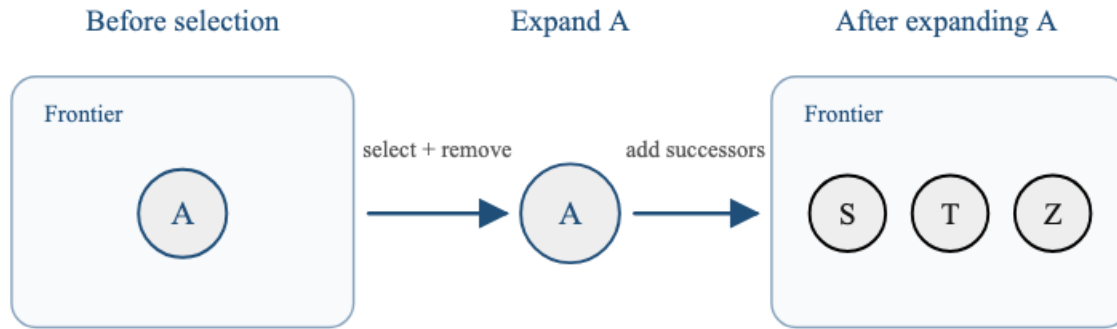
The Frontier Stores Nodes Waiting to Be Expanded

Slide 16 of 193 · Reveal 2

The Frontier Stores Nodes Waiting to Be Expanded

Frontier = generated nodes waiting to be expanded.

It is the search algorithm's current set of choices.



Which node should be expanded next: S, T, or Z?

16

Speaker notes

The frontier now contains three choices. Which node should be removed and expanded next?

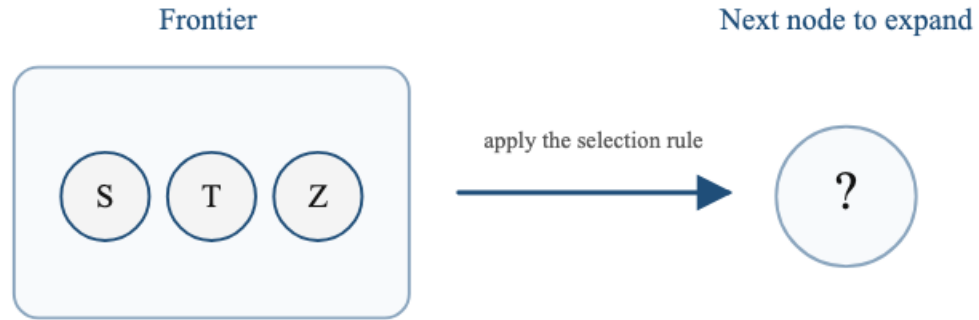
A search strategy answers that question by defining the order of node expansion.

The Frontier Stores Nodes Waiting to Be Expanded

Slide 16 of 193 · Reveal 3

A Search Strategy Chooses the Next Node to Expand

It is the rule that determines the order of node expansion.



17

Speaker notes

The frontier currently contains the nodes labelled S, T, and Z. A search strategy supplies the rule for choosing which frontier node to expand next.

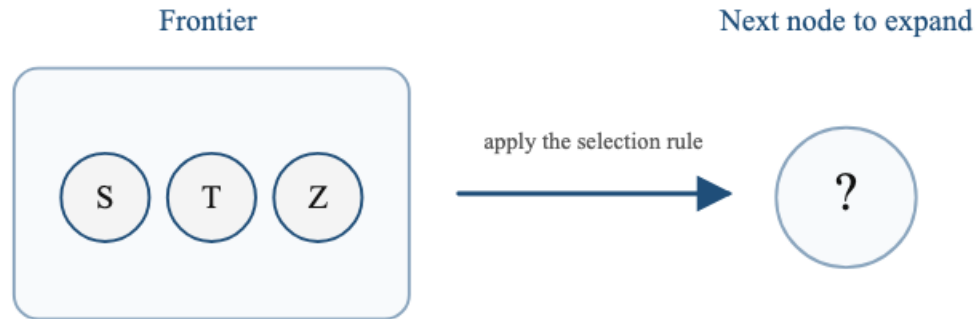
Because that rule determines the sequence of expansions, it also determines the order in which the search tree is explored.

A Search Strategy Chooses the Next Node to Expand

Slide 17 of 193 · Initial

A Search Strategy Chooses the Next Node to Expand

It is the rule that determines the order of node expansion.



How should we compare search strategies?

17

Speaker notes

Different strategies can make different choices from the same frontier. We therefore need criteria for comparing them.

A Search Strategy Chooses the Next Node to Expand

Slide 17 of 193 · Reveal 1

Four Questions Compare Search Strategies

We compare what a strategy guarantees and what resources it requires.

18

Four Questions Compare Search Strategies

Slide 18 of 193 · Initial

Speaker notes

We compare search strategies using four questions. Reveal each criterion separately so students can distinguish the guarantee or resource being measured.

Four Questions Compare Search Strategies

We compare what a strategy guarantees and what resources it requires.

Completeness

Will it find a solution whenever one exists?

18

Speaker notes

Completeness asks whether the strategy is guaranteed to find a solution whenever one exists.

Four Questions Compare Search Strategies

Slide 18 of 193 · Reveal 1

Four Questions Compare Search Strategies

We compare what a strategy guarantees and what resources it requires.

Completeness

Will it find a solution whenever one exists?

Optimality

Will it find a least-cost solution?

18

Speaker notes

Optimality asks whether the strategy is guaranteed to find a least-cost solution.

Four Questions Compare Search Strategies

Slide 18 of 193 · Reveal 2

Four Questions Compare Search Strategies

We compare what a strategy guarantees and what resources it requires.

Completeness

Will it find a solution whenever one exists?

Optimality

Will it find a least-cost solution?

Time

How many nodes are generated?

18

Speaker notes

Time is measured by the number of nodes generated during the search.

Four Questions Compare Search Strategies

Slide 18 of 193 · Reveal 3

Four Questions Compare Search Strategies

We compare what a strategy guarantees and what resources it requires.

Completeness

Will it find a solution whenever one exists?

Optimality

Will it find a least-cost solution?

Time

How many nodes are generated?

Space

What is the maximum number of nodes held in memory?

18

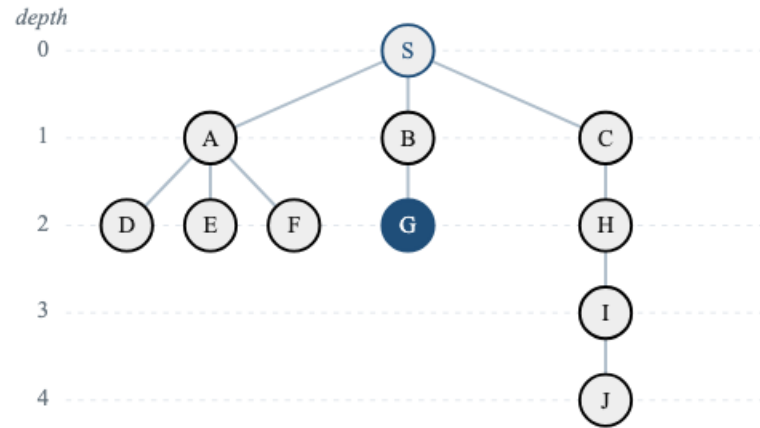
Speaker notes

Space is measured by the maximum number of nodes held in memory at any one time.

Four Questions Compare Search Strategies

Slide 18 of 193 · Reveal 4

Search Complexity Depends on b, d, and m



19

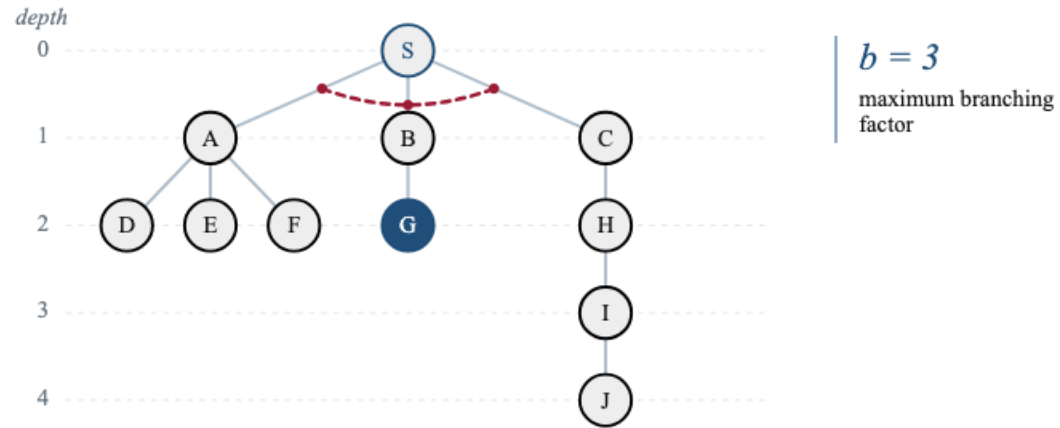
Search Complexity Depends on b, d, and m

Slide 19 of 193 · Initial

Speaker notes

Depth counts edges from the root. Use this one tree to make all three complexity parameters concrete.

Search Complexity Depends on b, d, and m



19

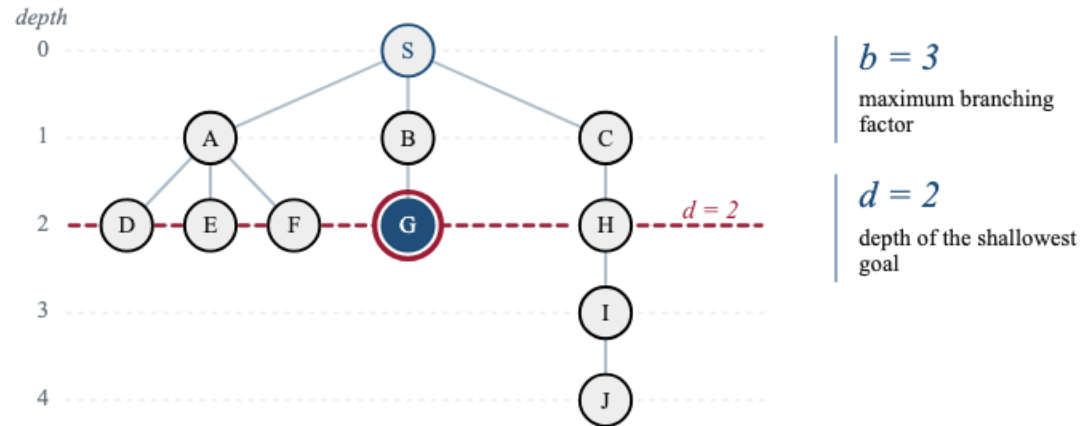
Speaker notes

The counting arc groups the root's three outgoing branches. Since no node has more than three children, the maximum branching factor is $b = 3$.

Search Complexity Depends on b, d, and m

Slide 19 of 193 · Reveal 1

Search Complexity Depends on b , d , and m



19

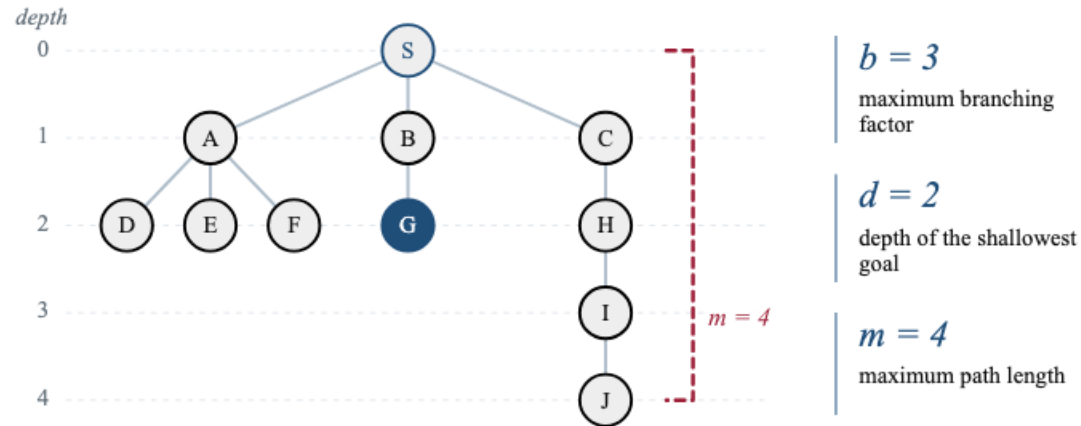
Search Complexity Depends on b , d , and m

Slide 19 of 193 · Reveal 2

Speaker notes

The branching-factor highlight disappears. The dashed line and goal ring now isolate the shallowest goal node G, which is two edges from the root, so $d = 2$.

Search Complexity Depends on b , d , and m



19

Speaker notes

The depth highlight disappears. The dashed bracket beside the deepest branch spans four edges, so $m = 4$. In an unbounded search tree, m may be infinite.

Search Complexity Depends on b , d , and m

Slide 19 of 193 · Reveal 3



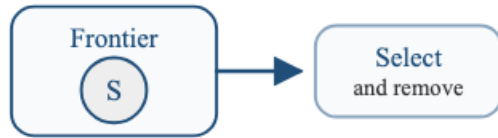
20

Search Repeats: Select, Test, Expand

Slide 20 of 193 · Initial

Speaker notes

The simplest search loop begins with a frontier containing a node for the initial state. Focus first on the shared select, test, and expand cycle; we will compare bookkeeping choices shortly.



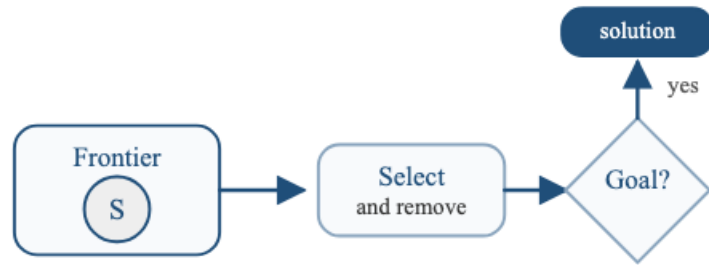
20

Search Repeats: Select, Test, Expand

Slide 20 of 193 · Reveal 1

Speaker notes

At each iteration, the algorithm selects one node and removes it from the frontier.



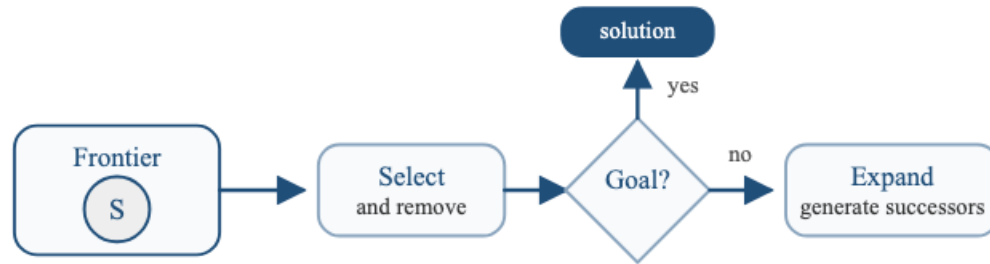
20

Search Repeats: Select, Test, Expand

Slide 20 of 193 · Reveal 2

Speaker notes

If the selected node contains a goal state, the algorithm returns the corresponding solution.



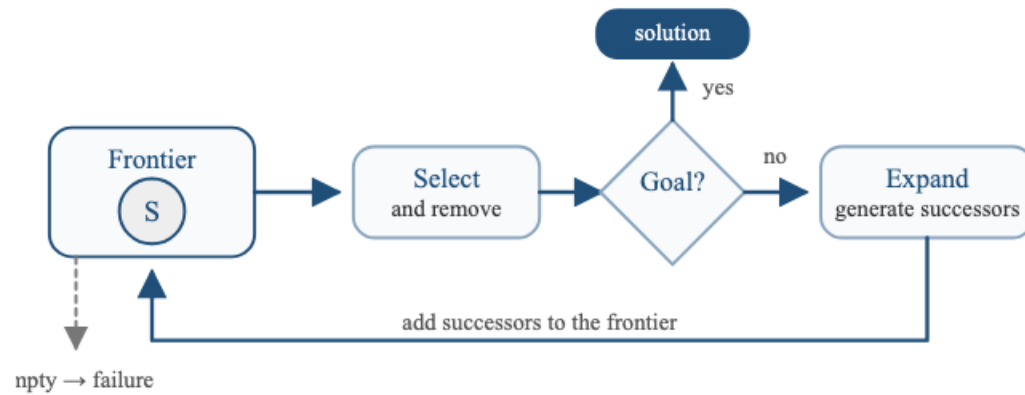
20

Search Repeats: Select, Test, Expand

Slide 20 of 193 · Reveal 3

Speaker notes

Otherwise, the algorithm expands the node and generates its successors.



The search strategy supplies the rule used at **Select**.

20

Speaker notes

The successors return to the frontier and the loop repeats. If the frontier becomes empty, no solution can be returned. The only part that changes between BFS, DFS, and UCS is the selection rule.

Search Repeats: Select, Test, Expand

Slide 20 of 193 · Reveal 4

Uninformed Search and Informed Search

Uninformed Search

Problem definition

No estimate to a goal

BFS · DFS · UCS

Informed Search

Problem definition

+ estimate to a goal

Greedy · A*

21

Speaker notes

An uninformed strategy uses the problem definition but has no estimate of the cost remaining to a goal. An informed strategy adds such an estimate to guide which node is selected next. We will study BFS, DFS, and UCS first, followed later by Greedy search and A*.

Uninformed Search and Informed Search

Slide 21 of 193 · Initial

Uninformed Search and Informed Search

Uninformed Search

Problem definition

No estimate to a goal

BFS · DFS · UCS

Informed Search

Problem definition

+ estimate to a goal

Greedy · A*

We begin with BFS.

21

Speaker notes

We begin with breadth-first search. Before examining how BFS orders the frontier, we separate the selection strategy from the bookkeeping used to avoid repeated work.

Uninformed Search and Informed Search

Slide 21 of 193 · Reveal 1

One Strategy, Two Search Algorithms

The strategy orders the frontier; TSA and GSA handle repeated states differently.

Tree Search Algorithm (TSA)

Keeps a new search node for every path

Graph Search Algorithm (GSA)

Remembers explored states and can skip repeats

22

Speaker notes

Separate two choices. A search strategy determines the order in which nodes leave the frontier. The tree-search and graph-search algorithms determine how repeated states are handled.

One Strategy, Two Search Algorithms

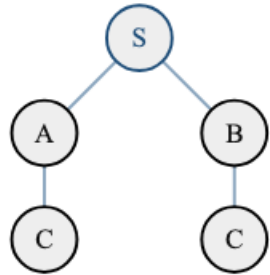
Slide 22 of 193 · Initial

One Strategy, Two Search Algorithms

The strategy orders the frontier; TSA and GSA handle repeated states differently.

Tree Search Algorithm (TSA)

Keeps a new search node for every path



State C appears in two search nodes

Graph Search Algorithm (GSA)

Remembers explored states and can skip repeats

22

Speaker notes

Tree search treats each path as a branch of the search tree. Because it does not keep a separate record of explored states, the two paths can create two different nodes containing state C.

One Strategy, Two Search Algorithms

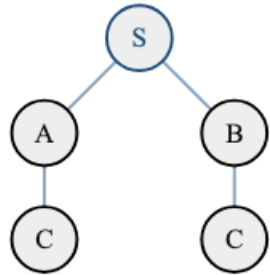
Slide 22 of 193 · Reveal 1

One Strategy, Two Search Algorithms

The strategy orders the frontier; TSA and GSA handle repeated states differently.

Tree Search Algorithm (TSA)

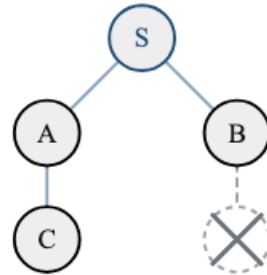
Keeps a new search node for every path



State C appears in two search nodes

Graph Search Algorithm (GSA)

Remembers explored states and can skip repeats



A node with explored state C can be skipped

22

Speaker notes

Graph search remembers explored states across paths. When a node contains a state that has already been explored, the algorithm can skip expanding it again. The precise duplicate-checking rule will be introduced later.

One Strategy, Two Search Algorithms

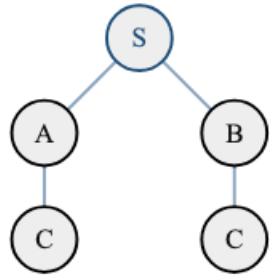
Slide 22 of 193 · Reveal 2

One Strategy, Two Search Algorithms

The strategy orders the frontier; TSA and GSA handle repeated states differently.

Tree Search Algorithm (TSA)

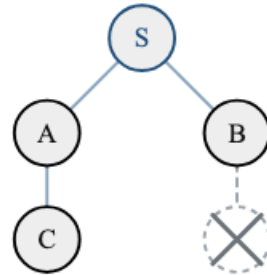
Keeps a new search node for every path



State C appears in two search nodes

Graph Search Algorithm (GSA)

Remembers explored states and can skip repeats



A node with explored state C can be skipped

We begin with BFS-TSA.

22

Speaker notes

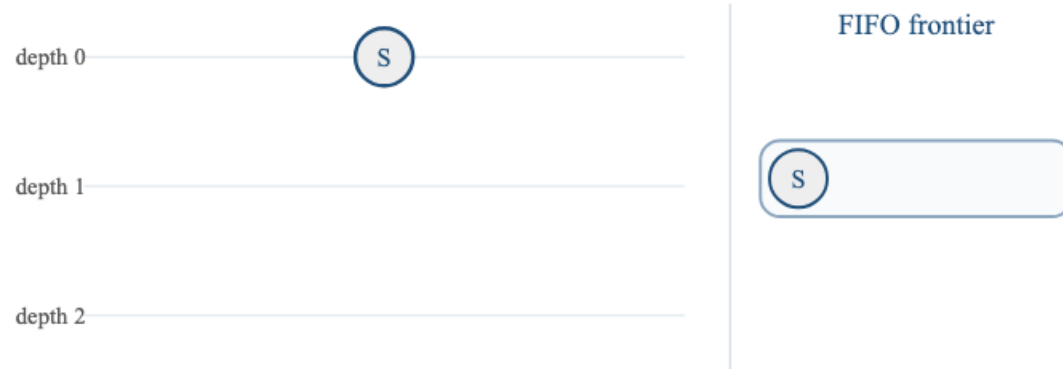
We begin with the tree-search form of breadth-first search: BFS-TSA. The next slide explains how its FIFO frontier orders the search nodes.

One Strategy, Two Search Algorithms

Slide 22 of 193 · Reveal 3

BFS Explores One Depth at a Time

BFS uses a FIFO frontier: earlier nodes stay ahead of newly generated nodes.



23

Speaker notes

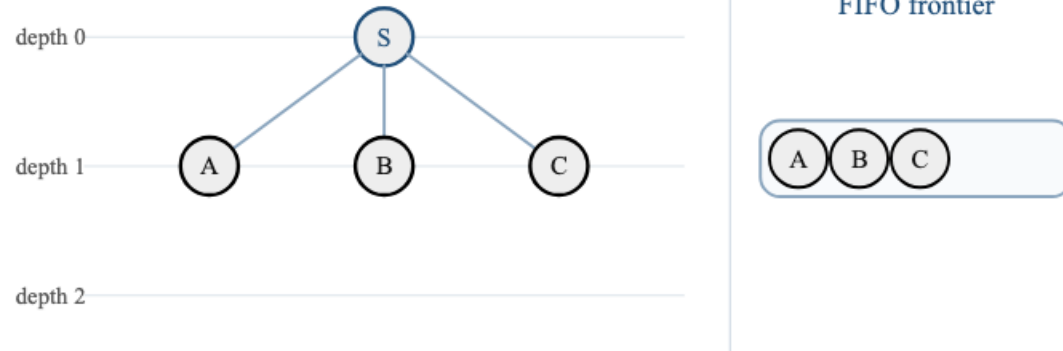
Begin with only S in the frontier. Breadth-first search removes the node that has waited longest: first in, first out.

BFS Explores One Depth at a Time

Slide 23 of 193 · Initial

BFS Explores One Depth at a Time

BFS uses a FIFO frontier: earlier nodes stay ahead of newly generated nodes.



23

Speaker notes

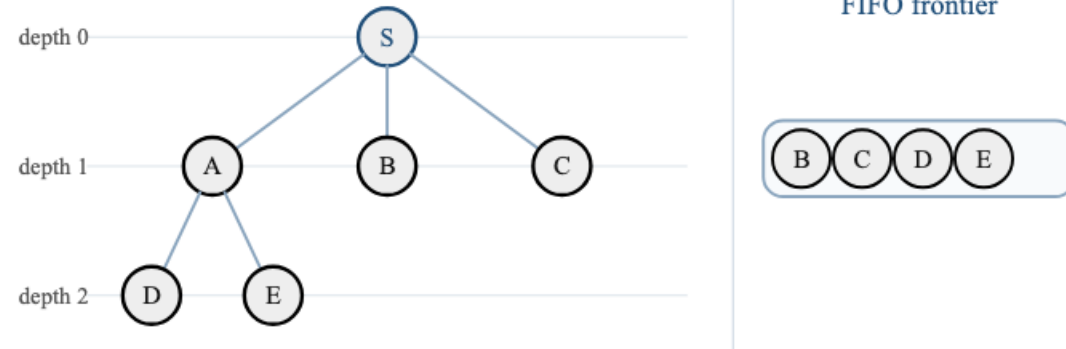
Expand S. Its children A, B, and C enter the frontier in that order.

BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 1

BFS Explores One Depth at a Time

BFS uses a FIFO frontier: earlier nodes stay ahead of newly generated nodes.



23

Speaker notes

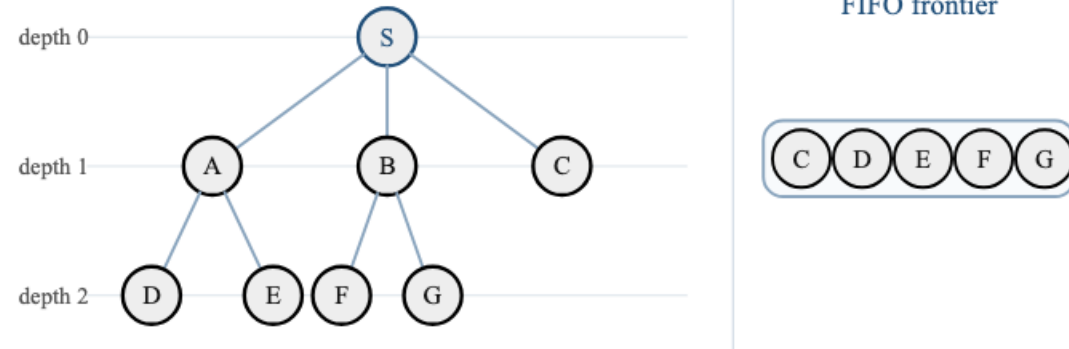
Expand A. Remove A from the front, then append its children D and E at the back. The frontier is now B, C, D, E.

BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 2

BFS Explores One Depth at a Time

BFS uses a FIFO frontier: earlier nodes stay ahead of newly generated nodes.



23

BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 3

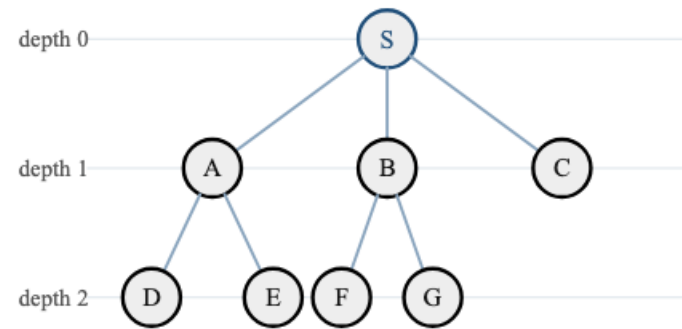
Speaker notes

Expand B. Remove B from the front, then append F and G. The frontier is now C, D, E, F, G.

BFS Explores One Depth at a Time

TASK

Which node should be expanded next?



FIFO frontier



23

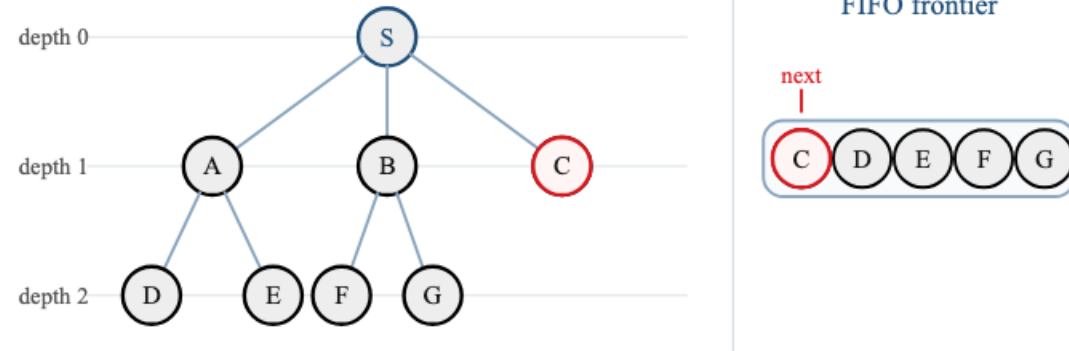
Speaker notes

Replace the FIFO statement with the task question. Ask students which node should be expanded next from the frontier C, D, E, F, G.

BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 4

BFS Explores One Depth at a Time

SOLUTION

23

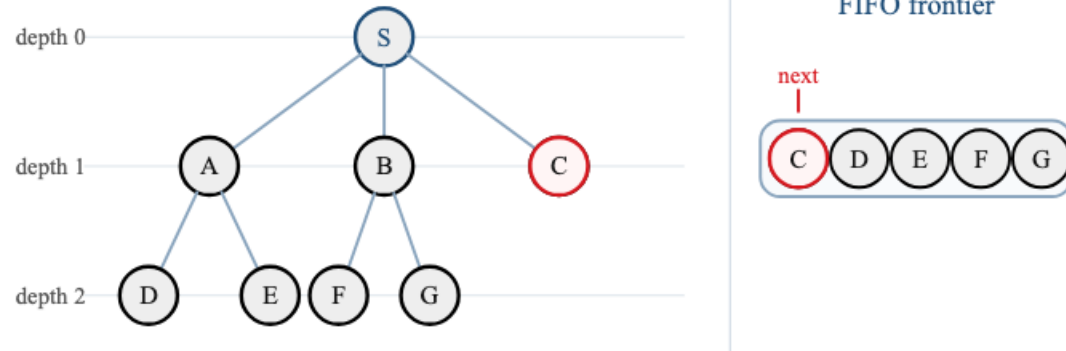
BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 5

Speaker notes

Reveal the answer: C is next. Highlight C both in the search tree and at the front of the frontier.

BFS Explores One Depth at a Time

SOLUTION

C is next: it entered the frontier at depth 1 before D, E, F, and G.

23

Speaker notes

C entered the frontier when S was expanded, before the depth-2 nodes D, E, F, and G were generated. A FIFO frontier therefore keeps every depth-1 node ahead of nodes added at depth 2.

BFS Explores One Depth at a Time

Slide 23 of 193 · Reveal 6

Frontier



```
main.py
1 from collections import deque
2 frontier = deque()
3 frontier.append("A") # 1st
4 frontier.append("B") # 2nd
5 frontier.append("C") # 3rd
6 print("frontier:", list(frontier))
7 print("next:", frontier.popleft())
8 print("remaining:", list(frontier))
9
```

[Download: comp3270.dirk.hk/queue_demo.py](#)

24

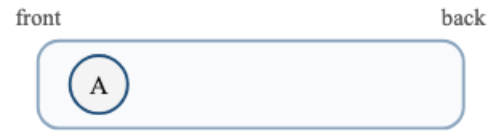
Python's deque Implements a FIFO Frontier

Slide 24 of 193 · Initial

Speaker notes

A deque—pronounced “deck”—is a double-ended queue. It supports adding and removing at either end. For a breadth-first frontier, we use only two operations: append at the back and remove from the front.

Frontier



append("A")

```
main.py
1 from collections import deque
2 frontier = deque()
3 frontier.append("A") # 1st
4 frontier.append("B") # 2nd
5 frontier.append("C") # 3rd
6 print("frontier:", list(frontier))
7 print("next:", frontier.popleft())
8 print("remaining:", list(frontier))
9
```

[Download: comp3270.dirk.hk/queue_demo.py](#)

24

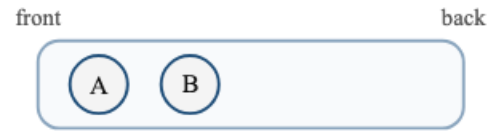
Python's deque Implements a FIFO Frontier

Slide 24 of 193 · Reveal 1

Speaker notes

Begin with an empty frontier. A is appended first, so A is at the front.

Frontier



```
append("A")  
append("B")
```

```
main.py  
1 from collections import deque  
2 frontier = deque()  
3 frontier.append("A") # 1st  
4 frontier.append("B") # 2nd  
5 frontier.append("C") # 3rd  
6 print("frontier:", list(frontier))  
7 print("next:", frontier.popleft())  
8 print("remaining:", list(frontier))  
9
```

[Download: comp3270.dirk.hk/queue_demo.py](#)

24

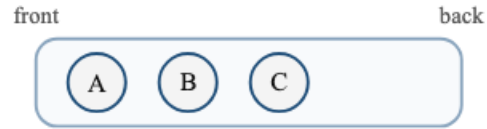
Python's deque Implements a FIFO Frontier

Slide 24 of 193 · Reveal 2

Speaker notes

B is appended second, behind A.

Frontier



```
append("A")  
append("B")  
append("C")
```

```
main.py  
1 from collections import deque  
2 frontier = deque()  
3 frontier.append("A") # 1st  
4 frontier.append("B") # 2nd  
5 frontier.append("C") # 3rd  
6 print("frontier:", list(frontier))  
7 print("next:", frontier.popleft())  
8 print("remaining:", list(frontier))  
9
```

[Download: comp3270.dirk.hk/queue_demo.py](#)

24

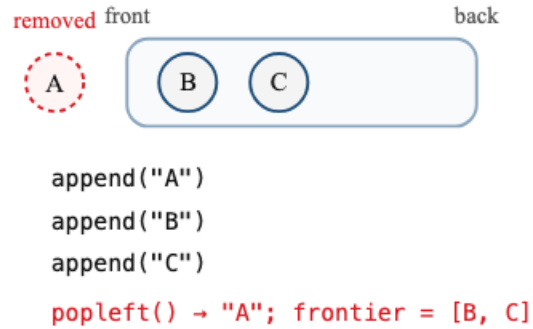
Python's deque Implements a FIFO Frontier

Slide 24 of 193 · Reveal 3

Speaker notes

C is appended third, behind B. The insertion order is therefore explicit: A, then B, then C.

Frontier



```
main.py  
1 from collections import deque  
2 frontier = deque()  
3 frontier.append("A") # 1st  
4 frontier.append("B") # 2nd  
5 frontier.append("C") # 3rd  
6 print("frontier:", list(frontier))  
7 print("next:", frontier.popleft())  
8 print("remaining:", list(frontier))  
9
```

[Download: comp3270.dirk.hk/queue_demo.py](#)

24

Python's deque Implements a FIFO Frontier

Slide 24 of 193 · Reveal 4

Speaker notes

`popleft()` removes and returns A, the item that entered first, leaving B and C. Run the code to confirm the same frontier order in Python.

A Search Node Stores the Path So Far

For our lecture code, a path string records one route through the search tree.

S

'S'

search tree	many possible routes
path	one route
current state	last character

25

Speaker notes

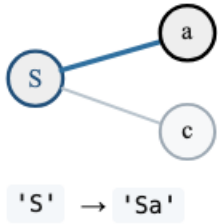
The search tree contains many possible routes. For our lecture code examples, we use a compact string to record one route from the start to a particular node. The initial one-state path is 'S'.

A Search Node Stores the Path So Far

Slide 25 of 193 · Initial

A Search Node Stores the Path So Far

For our lecture code, a path string records one route through the search tree.



search tree	many possible routes
path	one route
current state	last character

25

Speaker notes

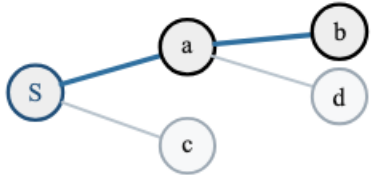
The tree branches to a and c. Following the highlighted branch to a gives the path 'Sa'; the other branch remains part of the tree, but not this path.

A Search Node Stores the Path So Far

Slide 25 of 193 · Reveal 1

A Search Node Stores the Path So Far

For our lecture code, a path string records one route through the search tree.



'S' → 'Sa' → 'Sab'

path[-1] gives 'b'

search tree	many possible routes
path	one route
current state	last character

25

Speaker notes

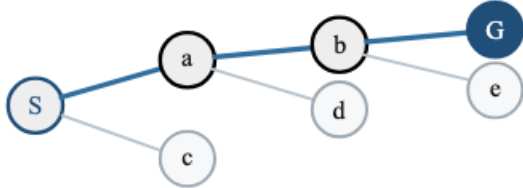
Following the next highlighted branch gives 'Sab'. The final character, obtained with path[-1], is 'b', so b is the current state.

A Search Node Stores the Path So Far

Slide 25 of 193 · Reveal 2

A Search Node Stores the Path So Far

For our lecture code, a path string records one route through the search tree.



'S' → 'Sa' → 'Sab' → 'SabG'

path[-1] gives 'G'

```
main.py
1 path = "Sab"
2 print("current:", path[-1])
3 path += "G"
4 print("path:", path)
5 print("current:", path[-1])
6
```

[Download: comp3270.dirk.hk/search_path_demo.py](#)

25

Speaker notes

Appending G updates the path to 'SabG'. Now path[-1] is 'G'. The runnable code performs exactly those two ideas: read the last state, then extend the path.

A Search Node Stores the Path So Far

Slide 25 of 193 · Reveal 3

BFS-TSA Repeats Three Operations

```
main.py  romania.py
1 from collections import deque
2 from romania import romania
3 def bfs_tsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     while frontier:
6         path = frontier.popleft()
7         state = path[-1]
8         if state == goal: return path
9         paths = [path + child for child in graph[state]]
10        frontier.extend(paths)
11        if trace: print(f"{path} -> {list(frontier)}")
12 print("solution:", bfs_tsa(romania, "A", "B", True))
13
```

Download: comp3270.dir.hk/bfs_tsa_romania.py

26

Speaker notes

The code on the right is the runnable BFS tree-search implementation used on the next Romania slide. Keep one invariant in mind: every frontier item is a path string.

BFS-TSA Repeats Three Operations

Slide 26 of 193 · Initial

BFS-TSA Repeats Three Operations

Remove the next path

Frontier ['A'] -> []

Path 'A'

```

1 from collections import deque
2 from romania import romania
3 def bfs_tsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     while frontier:
6         path = frontier.popleft()
7         state = path[-1]
8         if state == goal: return path
9         paths = [path + child for child in graph[state]]
10        frontier.extend(paths)
11        if trace: print(f"{path} -> {list(frontier)}")
12    print("solution:", bfs_tsa(romania, "A", "B", True))
13

```

Download: comp3270.dirk.hk/bfs_tsa_romania.py

26

Speaker notes

First, popleft() removes the path that has waited longest. The frontier changes from ['A'] to empty, and the removed item becomes the current path 'A'. That FIFO operation is what makes the search breadth-first.

BFS-TSA Repeats Three Operations

Slide 26 of 193 · Reveal 1

BFS-TSA Repeats Three Operations

Remove the next path

Frontier ['A'] → []

Path 'A'

Test its final state

Frontier []

Goal test
A ≠ B

```

1 from collections import deque
2 from romania import romania
3 def bfs_tsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     while frontier:
6         path = frontier.popleft()
7         state = path[-1]
8         if state == goal: return path
9         paths = [path + child for child in graph[state]]
10        frontier.extend(paths)
11        if trace: print(f"{path} → {list(frontier)}")
12    print("solution:", bfs_tsa(romania, "A", "B", True))
13

```

Download: comp3270.dirh.hk/bfs_tsa_romania.py

26

Speaker notes

Second, read the path's last character and test whether that state is the goal. The state is A, not B, and this test does not change the frontier.

BFS-TSA Repeats Three Operations

Slide 26 of 193 · Reveal 2

BFS-TSA Repeats Three Operations

Remove the next path

Frontier ['A'] → []

Path 'A'

Test its final state

Frontier []

Goal test
A ≠ B

Append every extension

Frontier [] → ['AS', 'AT', 'AZ']

```

1 from collections import deque
2 from romania import romania
3 def bfs_tsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     while frontier:
6         path = frontier.popleft()
7         state = path[-1]
8         if state == goal: return path
9         paths = [path + child for child in graph[state]]
10        frontier.extend(paths)
11        if trace: print(f"{path} → {list(frontier)}")
12    print("solution:", bfs_tsa(romania, "A", "B", True))
13

```

Download: comp3270.dir.hk/bfs_tsa_romania.py

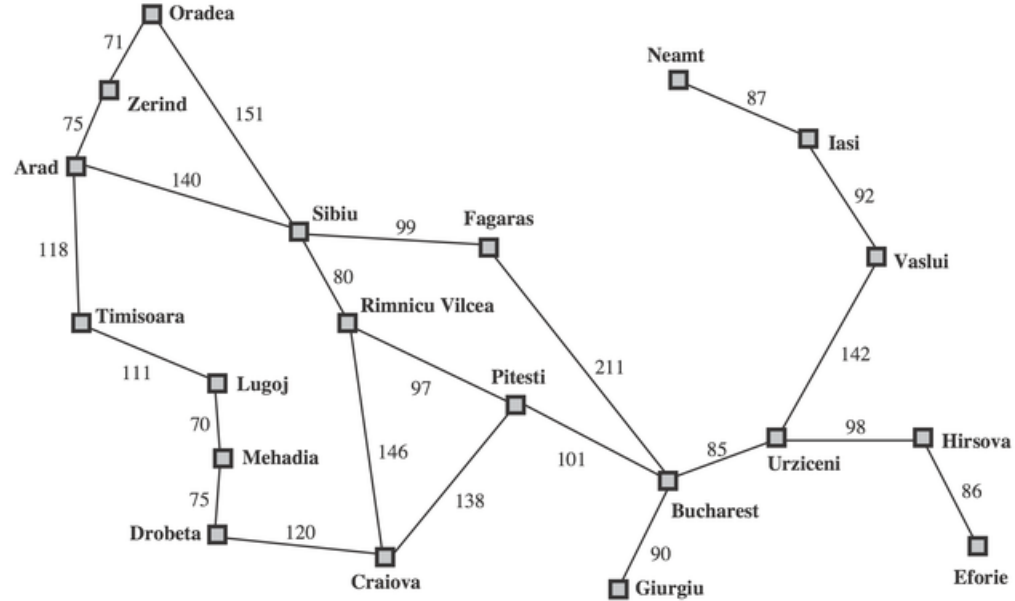
26

Speaker notes

Otherwise, extend 'A' with successors S, T, and Z. Appending 'AS', 'AT', and 'AZ' makes those three paths the current frontier.

BFS-TSA Repeats Three Operations

Slide 26 of 193 · Reveal 3



BFS-TSA Repeats Three Operations

Slide 26 of 193 · Reveal 4

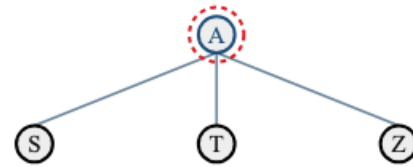
Speaker notes

No speaker notes for this stage.

Program Output

```
A -> ['AS', 'AT', 'AZ']
```

Search Tree



27

Speaker notes

Read the first trace line as an expansion. BFS-TSA removes 'A', expands state A, and appends the three paths 'AS', 'AT', and 'AZ'. The tree keeps only the state labels so the expansion order remains visually clear.

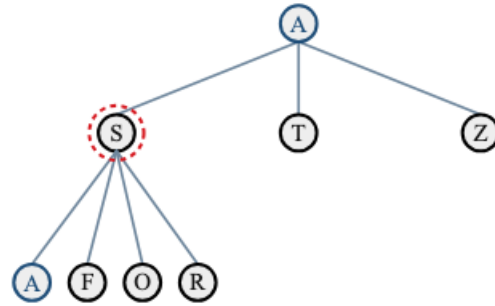
What Happens When a Path Returns to A?

Slide 27 of 193 · Initial

Program Output

```
A -> ['AS', 'AT', 'AZ']  
AS -> ['AT', 'AZ', 'ASA', 'ASF',  
      'ASO', 'ASR']
```

Search Tree



27

Speaker notes

FIFO means 'AS' is removed next. Expanding S creates four children: 'ASA', 'ASF', 'ASO', and 'ASR'. They go behind 'AT' and 'AZ', which were already waiting.

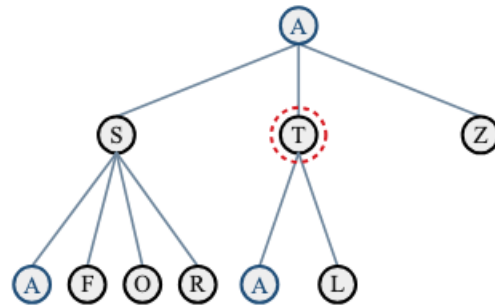
What Happens When a Path Returns to A?

Slide 27 of 193 · Reveal 1

Program Output

```
A -> ['AS', 'AT', 'AZ']  
AS -> ['AT', 'AZ', 'ASA', 'ASF',  
       'ASO', 'ASR']  
AT -> ['AZ', 'ASA', 'ASF', 'ASO',  
       'ASR', 'ATA', 'ATL']
```

Search Tree



27

Speaker notes

Next, BFS removes 'AT'. Expanding T adds 'ATA' and 'ATL' at the back. The path 'ASA' is still waiting behind 'AZ'.

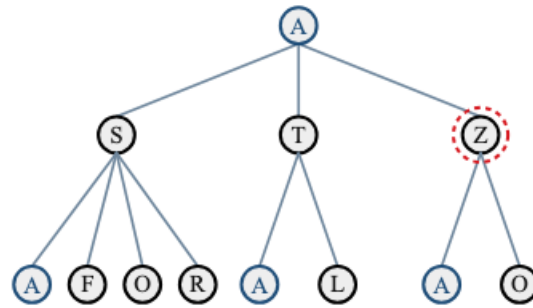
What Happens When a Path Returns to A?

Slide 27 of 193 · Reveal 2

Program Output

```
A -> ['AS', 'AT', 'AZ']  
AS -> ['AT', 'AZ', 'ASA', 'ASF',  
      'ASO', 'ASR']  
AT -> ['AZ', 'ASA', 'ASF', 'ASO',  
      'ASR', 'ATA', 'ATL']  
AZ -> ['ASA', 'ASF', 'ASO', 'ASR',  
      'ATA', 'ATL', 'AZA', 'AZO']
```

Search Tree



27

Speaker notes

Then BFS removes 'AZ'. Expanding Z adds 'AZA' and 'AZO'. Now 'ASA' has reached the front of the frontier.

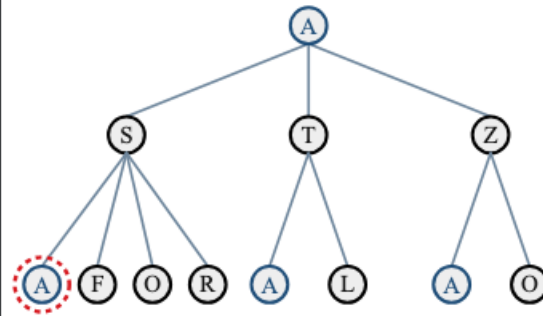
What Happens When a Path Returns to A?

Slide 27 of 193 · Reveal 3

Program Output

```
A -> ['AS', 'AT', 'AZ']  
AS -> ['AT', 'AZ', 'ASA', 'ASF',  
      'ASO', 'ASR']  
AT -> ['AZ', 'ASA', 'ASF', 'ASO',  
      'ASR', 'ATA', 'ATL']  
AZ -> ['ASA', 'ASF', 'ASO', 'ASR',  
      'ATA', 'ATL', 'AZA', 'AZO']
```

Search Tree



Explore A again?

27

Speaker notes

The next path is 'ASA', whose final state is A. Ask the question on screen and let students recognize the problem: tree search is about to expand A again.

What Happens When a Path Returns to A?

Slide 27 of 193 · Reveal 4

PROBLEM TSA revisits states

It does not remember where it has been



28

Speaker notes

TSA is doomed to revisit states because it has no memory of earlier expansions. Returning to A looks like a fresh opportunity, even though A has already been expanded.

TSA Repeats States Because It Has No Memory

Slide 28 of 193 · Initial

PROBLEM TSA revisits states

It does not remember where it has been

**SOLUTION** Remember explored states

explored = {'A', 'S'} $A \in \text{explored} \rightarrow \text{skip}$

28

Speaker notes

The fix is to remember expanded states. If A is already in the explored set, the repeated path is skipped instead of generating another copy of A's successors.

TSA Repeats States Because It Has No Memory

Slide 28 of 193 · Reveal 1

PROBLEM TSA revisits states

It does not remember where it has been

**SOLUTION** Remember explored states

explored = {'A', 'S'} $A \in \text{explored} \rightarrow \text{skip}$

TSA + explored states = **GSA**

Repeated states are skipped instead of expanded again

28

Speaker notes

This is the conceptual step from a Tree Search Algorithm to a Graph Search Algorithm: keep the same frontier, add memory for explored states, and suppress repeated expansions.

TSA Repeats States Because It Has No Memory

Slide 28 of 193 · Reveal 2

Frontier ['AS', 'AT', 'AZ']
stores path strings

Explored {'A'}
stores states only

```

1 from collections import deque
2 from romania import romania
3 def bfs_gsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     explored = set()
6     while frontier:
7         path = frontier.popleft()
8         state = path[-1]
9         if state == goal: return path
10        if state in explored: continue
11        explored.add(state)
12        paths = [path + child for child in graph[s
13        frontier.extend(paths)
14        if trace:
15            print(f"{path} -> {list(frontier)}")
16            print(f"explored: {sorted(explored)}\n")
17    print("solution:". bfs_gsa(romania, "A", "B", True)

```

Download: comp3270.dirk.hk/bfs_gsa_romania.py

29

Speaker notes

Graph search still needs complete path strings in its frontier so it can return a route. Its new explored set is different: it stores only state labels.

BFS-GSA Remembers Explored States

Slide 29 of 193 · Initial

Frontier ['AS', 'AT', 'AZ']

stores path strings

Explored {'A'}

stores states only

Create the memory

`explored = set()`

```

main.py  romania.py
1  from collections import deque
2  from romania import romania
3  def bfs_gsa(graph, start, goal, trace=False):
4      frontier = deque([start])
5      explored = set()
6      while frontier:
7          path = frontier.popleft()
8          state = path[-1]
9          if state == goal: return path
10         if state in explored: continue
11         explored.add(state)
12         paths = [path + child for child in graph[s
13         frontier.extend(paths)
14         if trace:
15             print(f"{path} -> {list(frontier)}")
16             print(f"explored: {sorted(explored)}\n")
17     print("solution:". bfs_gsa(romania, "A", "B", True)

```

Download: comp3270.dirk.hk/bfs_gsa_romania.py

29

Speaker notes

Start with an empty set.

BFS-GSA Remembers Explored States

Slide 29 of 193 · Reveal 1

Frontier ['AS', 'AT', 'AZ']
stores path strings

Explored {'A'}
stores states only

Create the memory
`explored = set()`

Skip a repeated state
`state in explored`

```

1 from collections import deque
2 from romania import romania
3 def bfs_gsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     explored = set()
6     while frontier:
7         path = frontier.popleft()
8         state = path[-1]
9         if state == goal: return path
10        if state in explored: continue
11        explored.add(state)
12        paths = [path + child for child in graph[s]]
13        frontier.extend(paths)
14        if trace:
15            print(f"{path} -> {list(frontier)}")
16            print(f"explored: {sorted(explored)}\n")
17    print("solution:". bfs_gsa(romania, "A", "B", True)

```

Download: comp3270.dirk.hk/bfs_gsa_romania.py

29

Speaker notes

When a path is removed, inspect its final state. If that state is already explored, skip this path instead of expanding it again.

BFS-GSA Remembers Explored States

Slide 29 of 193 · Reveal 2

Frontier ['AS', 'AT', 'AZ']
stores path strings

Explored {'A'}
stores states only

Create the memory
`explored = set()`

Skip a repeated state
`state in explored`

Remember before expanding
`explored.add(state)`

```

1 from collections import deque
2 from romania import romania
3 def bfs_gsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     explored = set()
6     while frontier:
7         path = frontier.popleft()
8         state = path[-1]
9         if state == goal: return path
10        if state in explored: continue
11        explored.add(state)
12        paths = [path + child for child in graph[s]]
13        frontier.extend(paths)
14        if trace:
15            print(f"{path} -> {list(frontier)}")
16            print(f"explored: {sorted(explored)}\n")
17    print("solution:". bfs_gsa(romania, "A", "B", True)

```

Download: comp3270.dirk.hk/bfs_gsa_romania.py

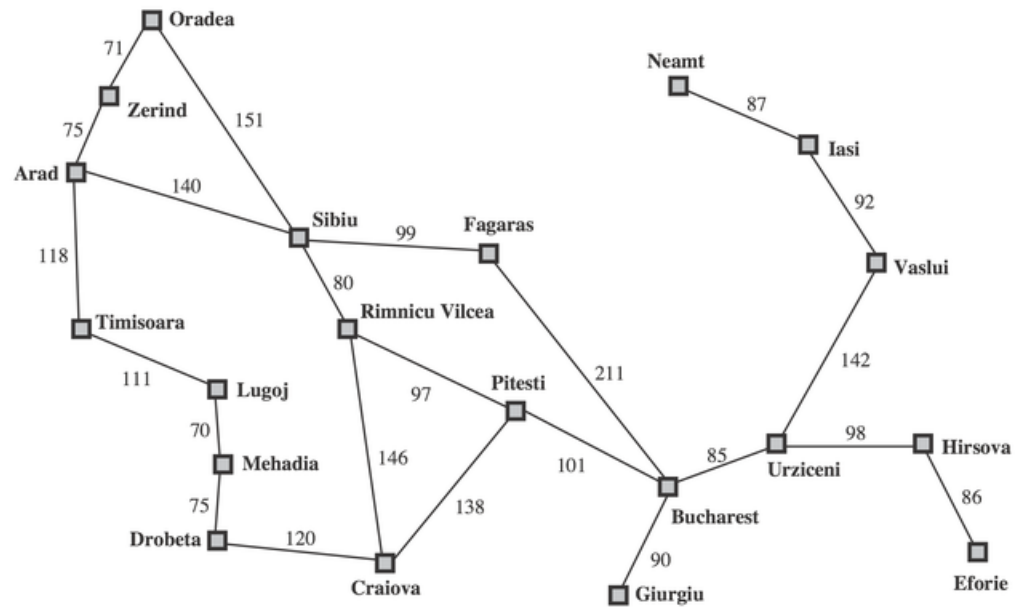
29

Speaker notes

Otherwise, add the state to explored before appending its successor paths.
These are the three meaningful additions to the TSA code.

BFS-GSA Remembers Explored States

Slide 29 of 193 · Reveal 3



Romania State-Space Graph

Slide 30 of 193 · Initial

Speaker notes

[Sources] - AIMA Romania route problem: <https://aima.cs.berkeley.edu/python/search.html>



frontier:
paths

explored:
states

```

1 from collections import deque
2 from romania import romania
3 def bfs_gsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     explored = set()
6     while frontier:
7         path = frontier.popleft()
8         state = path[-1]
9         if state == goal: return path
10        if state in explored: continue
11        explored.add(state)
12        paths = [path + child for child in graph[state]]
13        frontier.extend(paths)
14        if trace:
15            print(f"{path} -> {list(frontier)}")
16            print(f"explored: {sorted(explored)}\n")
17    print("solution:", bfs_gsa(romania, "A", "B", True))

```

Download: comp3270.dirk.hv/bfs_gsa_romania.py

31

Speaker notes

Run the GSA implementation. Each trace step keeps the same path -> frontier notation as TSA, then shows the explored states underneath. The trace ends when B is removed and the goal test succeeds, returning ASFB. No state is expanded a second time.

[Sources] - AIMA Romania route problem: <https://aima.cs.berkeley.edu/python/search.html> - Berkeley CS188, Uninformed Search: <https://inst.eecs.berkeley.edu/~cs188/textbook/search/uninformed.html> - Pedagogical redesign of recovered COMP3270 2024-25 source slide 28: 01_search.pptx.

Run BFS-GSA on Romania

Slide 31 of 193 · Initial

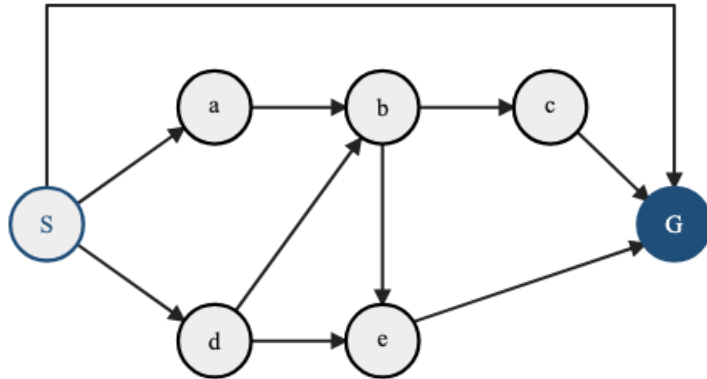
BFS-GSA Practice

TASK

Run BFS-GSA from S to G

Record Explored order
Returned path

Add successors alphabetically



32

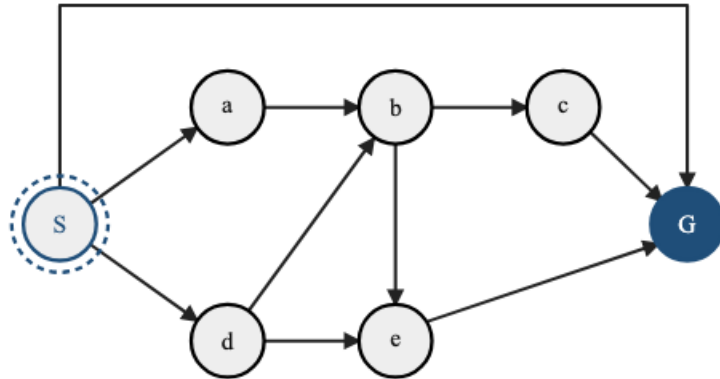
BFS-GSA Practice

Slide 32 of 193 · Initial

Speaker notes

Ask students to run BFS-GSA mentally from S to G. They must report two different answers: the order of explored states and the returned path. Successors enter the FIFO frontier alphabetically.

BFS-GSA Practice



TASK

Run BFS-GSA from S to G

Record Explored order

Returned path

Add successors alphabetically

SOLUTION

```

S -> ['Sa', 'Sd', 'SG']
explored: ['S']
  
```

32

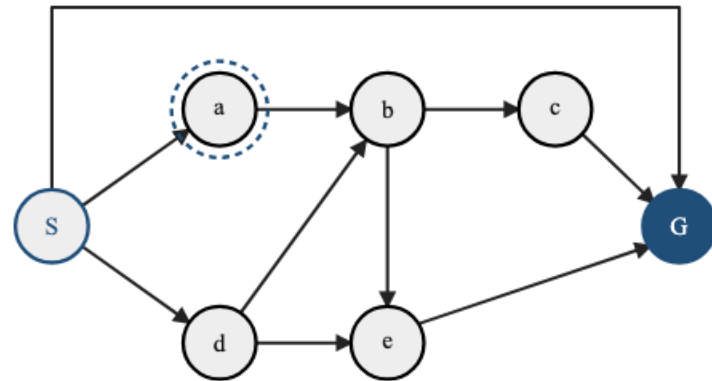
BFS-GSA Practice

Slide 32 of 193 · Reveal 1

Speaker notes

Start with path S. S is not the goal, so add S to explored and append Sa, Sd, and SG in alphabetical order.

BFS-GSA Practice



TASK

Run BFS-GSA from S to G

Record Explored order

Returned path

Add successors alphabetically

SOLUTION

```

S -> ['Sa', 'Sd', 'SG']
explored: ['S']
Sa -> ['Sd', 'SG', 'Sab']
explored: ['S', 'a']
  
```

32

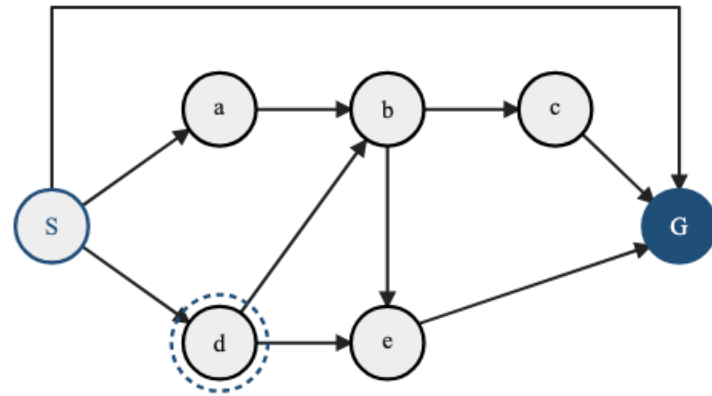
BFS-GSA Practice

Slide 32 of 193 · Reveal 2

Speaker notes

FIFO removes Sa next. Its final state is a. Add a to explored and append Sab behind the paths already waiting.

BFS-GSA Practice



TASK

Run BFS-GSA from S to G

Record Explored order

Returned path

Add successors alphabetically

SOLUTION

```

S -> ['Sa', 'Sd', 'SG']
explored: ['S']
Sa -> ['Sd', 'SG', 'Sab']
explored: ['S', 'a']
Sd -> ['SG', 'Sab', 'Sdb', 'Sde']
explored: ['S', 'a', 'd']
  
```

32

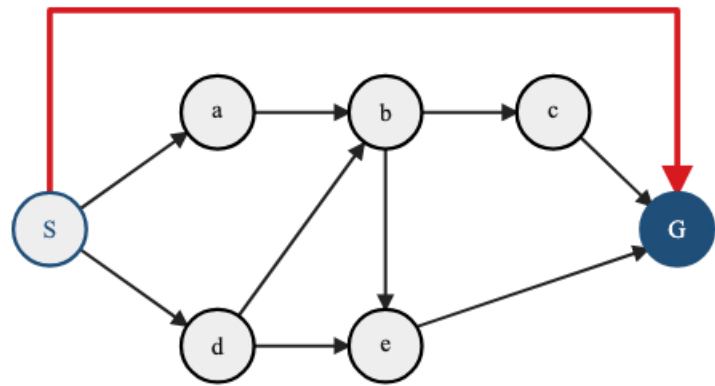
BFS-GSA Practice

Slide 32 of 193 · Reveal 3

Speaker notes

Next BFS removes Sd. Add d to explored, then append Sdb and Sde.

BFS-GSA Practice



TASK

Run BFS-GSA from S to G

Record Explored order
 Returned path

Add successors alphabetically

SOLUTION

```
S -> ['Sa', 'Sd', 'SG']
explored: ['S']
Sa -> ['Sd', 'SG', 'Sab']
explored: ['S', 'a']
Sd -> ['SG', 'Sab', 'Sdb', 'Sde']
explored: ['S', 'a', 'd']
solution: SG
```

Explored S → a → d Path S → G

Speaker notes

The next path is SG. The goal test succeeds before G is added to explored. Therefore the explored order is S, a, d, while the returned solution path is S to G.

BFS-GSA Practice

```

main.py
1 from collections import deque
2 def bfs_gsa(graph, start, goal, trace=False):
3     frontier = deque([start])
4     explored = set()
5     while frontier:
6         path = frontier.popleft()
7         state = path[-1]
8         if state == goal: return path
9         if state in explored: continue
10        explored.add(state)
11        paths = [path + child for child in graph[state]]
12        frontier.extend(paths)
13        if trace:
14            print(f"{path} -> {list(frontier)}")
15            print(f"explored: {sorted(explored)}\n")
16 practice = {"S": ["a", "d", "G"], "a": ["b"],
17            "d": ["b", "e"], "b": ["c", "e"],
18            "G": ["c"], "e": ["c"], "c": []}

```

Download: comp3270.dir.hk/bfs_gsa_practice.py

33

Run the BFS-GSA Practice Code

Slide 33 of 193 · Initial

Speaker notes

This is the same BFS-GSA implementation and practice graph used on the previous slide. Run it to reproduce the trace and confirm that the explored order and returned path are different outputs.

[Sources] - Adapted from the recovered COMP3270 2024–25 runnable bfsGsaPractice.py. - Python documentation for collections.deque: <https://docs.python.org/3/library/collections.html#collections.deque>

BFS-TSA Properties

TASK

Complete?

Optimal?

Time complexity?

Space complexity?

34

BFS-TSA Properties

Slide 34 of 193 · Initial

Speaker notes

Ask students to evaluate BFS-TSA using the four questions introduced earlier: completeness, optimality, time, and space. This analysis does not use an explored set.

BFS-TSA Properties

SOLUTION

Complete?

Yes

Optimal?

Time complexity?

Space complexity?

34

BFS-TSA Properties

Slide 34 of 193 · Reveal 1

Speaker notes

BFS-TSA is complete when the branching factor is finite and a solution exists at a finite depth.

BFS-TSA Properties

SOLUTION

Complete?

Yes

Optimal?

No

Time complexity?

Space complexity?

34

BFS-TSA Properties

Slide 34 of 193 · Reveal 2

Speaker notes

BFS-TSA is not optimal in general: it returns a shallowest solution, which need not be the least-cost solution.

BFS-TSA Properties

SOLUTION

Complete?

Yes

Optimal?

No

unless all step costs are equal

Time complexity?

Space complexity?

34

Speaker notes

The exception is when all step costs are equal. Then minimizing depth also minimizes path cost.

BFS-TSA Properties

Slide 34 of 193 · Reveal 3

BFS-TSA Properties

SOLUTION

Complete?

Yes

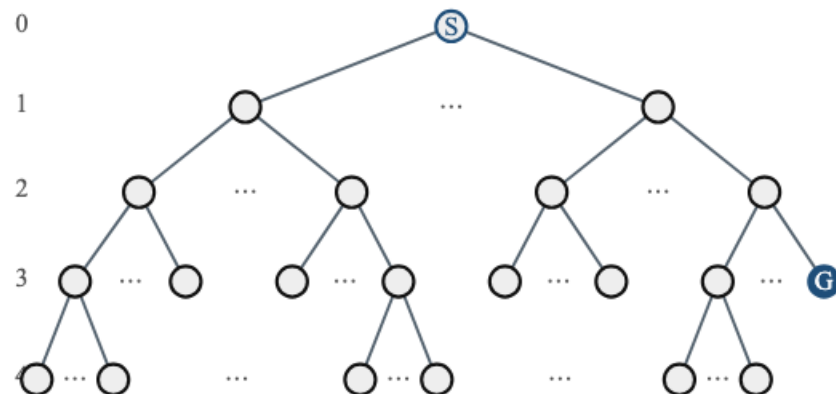
Optimal?

No

unless all step costs are equal

Time complexity?

Space complexity?



b maximum branching factor

d depth of the shallowest solution

34

Speaker notes

The goal G is the final depth-three path removed. Before reaching it, BFS expands earlier depth-three paths and adds their children at depth four. The dots indicate omitted successors and subtrees when the branching factor is greater than two.

BFS-TSA Properties

Slide 34 of 193 · Reveal 4

BFS-TSA Properties

SOLUTION

Complete?

Yes

Optimal?

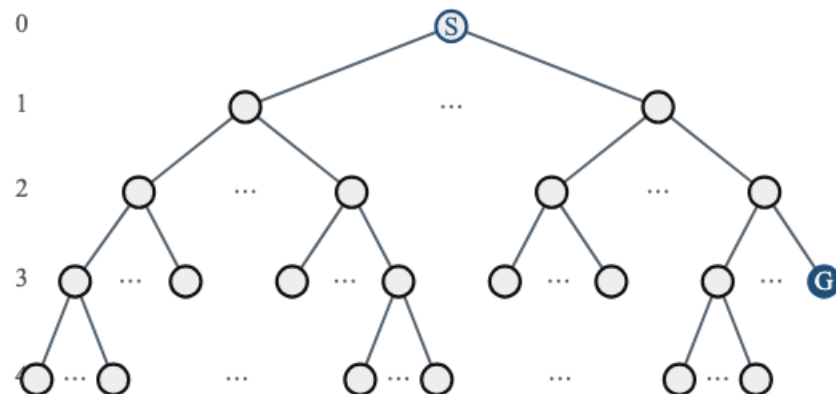
No

unless all step costs are equal

Time complexity?

$O(b^{d+1}) = O(b^d)$

Space complexity?



b maximum branching factor

d depth of the shallowest solution

*If we treat **b** as fixed

34

Speaker notes

Our code tests for the goal when a path is removed from the frontier. In the worst case, it generates paths at depth d plus one before removing the goal, giving $O(bd+1)$. If we treat b as fixed, this is also written $O(bd)$.

BFS-TSA Properties

Slide 34 of 193 · Reveal 5

BFS-TSA Properties

SOLUTION

Complete?

Yes

Optimal?

No

unless all step costs are equal

Time complexity?

$$O(b^{d+1}) = * O(b^d)$$

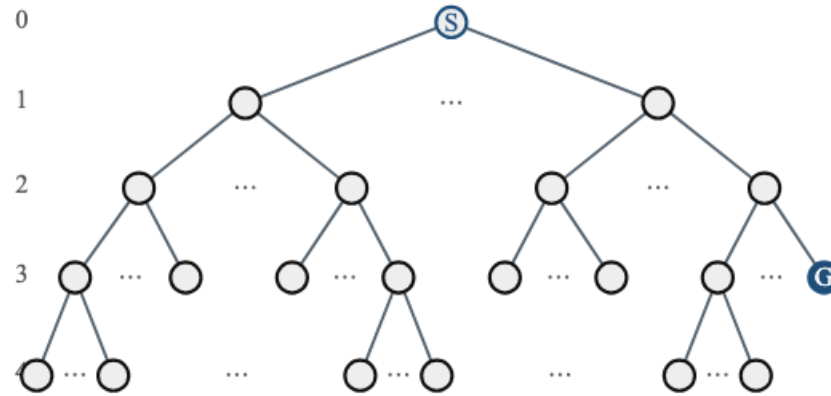
Space complexity?

$$O(b^{d+1}) = * O(b^d)$$

b maximum branching factor

d depth of the shallowest solution

*If we treat **b** as fixed



34

Speaker notes

Space has the same bound because the frontier may contain those depth d plus one paths while the goal is still waiting to be removed.

BFS-TSA Properties

Slide 34 of 193 · Reveal 6

BFS-TSA Properties

SOLUTION

Complete?
Yes

Optimal?
No

unless all step costs are equal

Time complexity?
 $O(b^{d+1}) = * O(b^d)$

Space complexity?
 $O(b^{d+1}) = * O(b^d)$

b maximum branching factor

d depth of the shallowest solution

*If we treat **b** as fixed

Assume $b = 10$, 1 million nodes/second, 1000 bytes/node

Depth	Nodes	Time	Memory
2	110	0.11 milliseconds	110 kilobytes
4	11,110	11 milliseconds	11 megabytes
6	1.11×10^6	1.1 seconds	1.1 gigabytes
8	1.11×10^8	1.9 minutes	111 gigabytes
10	1.11×10^{10}	3.1 hours	11 terabytes
12	1.11×10^{12}	12.9 days	1.1 petabytes
14	1.11×10^{14}	3.5 years	111 petabytes
16	1.11×10^{16}	352 years	11 exabytes

34

Speaker notes

Now replace the small tree with branching factor ten. The node count is the sum from depth one through depth d, excluding the root. Times and memory use decimal units and are rounded. Red values mark the point at which even this simplified search becomes impractical for an interactive system.

BFS-TSA Properties

BFS Explores the Shallowest Node First



frontier is a **queue** (first in first out data structure)

35

Speaker notes

This is the concise breadth-first counterpart to the next slide. BFS always removes a shallowest frontier node; implementing the frontier as a FIFO queue preserves that ordering.

[Sources] - Recovered COMP3270 2024–25 source slide 20: 01_search.pptx. - UC Berkeley CS188 search illustration: <http://ai.berkeley.edu>.

BFS Explores the Shallowest Node First

Slide 35 of 193 · Initial

DFS Explores the Deepest Node First



frontier is a **stack** (last in first out data structure)

36

DFS Explores the Deepest Node First

Slide 36 of 193 · Initial

Speaker notes

[Sources] - Recovered COMP3270 2024–25 source slide 33:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Frontier

bottom

top



```
main.py
1 from collections import deque
2 frontier = deque()
3 frontier.append("A") # 1st
4 frontier.append("B") # 2nd
5 frontier.append("C") # 3rd
6 print("frontier:", list(frontier))
7 print("next:", frontier.pop())
8 print("remaining:", list(frontier))
9
```

[Download: comp3270.diri.hk/stack_demo.py](#)

37

Speaker notes

The same deque can implement a stack. DFS adds and removes paths at the same end, so the newest path is selected first.

Python's deque Implements a LIFO Frontier

Slide 37 of 193 · Initial

Frontier

bottom

top



append("A")

```
main.py
1 from collections import deque
2 frontier = deque()
3 frontier.append("A") # 1st
4 frontier.append("B") # 2nd
5 frontier.append("C") # 3rd
6 print("frontier:", list(frontier))
7 print("next:", frontier.pop())
8 print("remaining:", list(frontier))
9
```

[Download: comp3270.diri.hk/stack_demo.py](#)

37

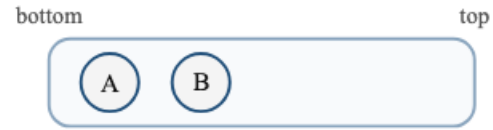
Speaker notes

Append A to the empty frontier.

Python's deque Implements a LIFO Frontier

Slide 37 of 193 · Reveal 1

Frontier



```
append("A")  
append("B")
```

```
main.py  
1 from collections import deque  
2 frontier = deque()  
3 frontier.append("A") # 1st  
4 frontier.append("B") # 2nd  
5 frontier.append("C") # 3rd  
6 print("frontier:", list(frontier))  
7 print("next:", frontier.pop())  
8 print("remaining:", list(frontier))  
9
```

Download: comp3270.drii.hk/stack_demo.py

37

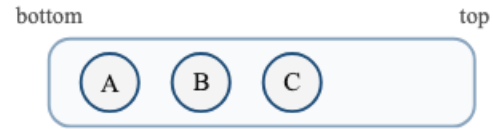
Speaker notes

Append B after A.

Python's deque Implements a LIFO Frontier

Slide 37 of 193 · Reveal 2

Frontier



```
append("A")  
append("B")  
append("C")
```

```
main.py  
1 from collections import deque  
2 frontier = deque()  
3 frontier.append("A") # 1st  
4 frontier.append("B") # 2nd  
5 frontier.append("C") # 3rd  
6 print("frontier:", list(frontier))  
7 print("next:", frontier.pop())  
8 print("remaining:", list(frontier))  
9
```

Download: comp3270.dir.hk/stack_demo.py

37

Speaker notes

Append C after B. C is now at the top of the stack.

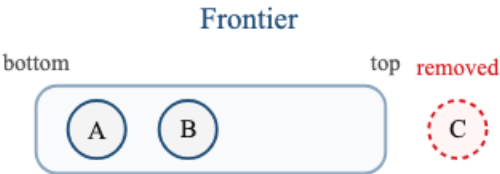
Python's deque Implements a LIFO Frontier

Slide 37 of 193 · Reveal 3

bottom

Frontier

top removed



```
append("A")
append("B")
append("C")
pop() -> "C"; frontier = [A, B]
```

```
1 from collections import deque
2 frontier = deque()
3 frontier.append("A") # 1st
4 frontier.append("B") # 2nd
5 frontier.append("C") # 3rd
6 print("frontier:", list(frontier))
7 print("next:", frontier.pop())
8 print("remaining:", list(frontier))
9
```

Download: comp3270.diri.hk/stack_demo.py

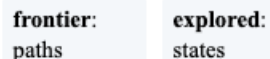
37

Python's deque Implements a LIFO Frontier

Slide 37 of 193 · Reveal 4

Speaker notes

pop() removes C, the path added most recently, leaving A and B. Compare this directly with popleft() on the FIFO slide.



```
1 from collections import deque
2 from romania import romania
3 def dfs_gsa(graph, start, goal, trace=False):
4     frontier = deque([start])
5     explored = set()
6     while frontier:
7         path = frontier.pop()
8         state = path[-1]
9         if state == goal: return path
10        if state in explored: continue
11        explored.add(state)
12        paths = [path + child for child in graph[state]]
13        frontier.extend(paths)
14        if trace:
15            print(f"{path} -> {list(frontier)}")
16            print(f"explored: {sorted(explored)}\n")
17    print("solution:", dfs_gsa(romania, "A", "B", True))
```

Download: comp3270.dirk.hk/dfs_gsa_romania.py

38

This is the BFS-GSA program with one strategic change: `popleft()` becomes `pop()`. The frontier still stores paths, the explored set still stores states, and the trace notation is unchanged.

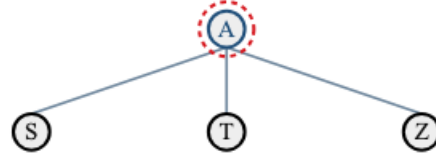
[Sources] - AIMA Romania route problem: <https://aima.cs.berkeley.edu/python/search.html> - Python documentation for collections.deque: <https://docs.python.org/3/library/collections.html#collections.deque>

Slide 38 of 193 · Initial

Program Output

```
A -> ['AS', 'AT', 'AZ']
explored: ['A']
```

Search Tree



39

Speaker notes

Read the trace exactly as before. Expanding A appends AS, AT, and AZ. Because DFS removes from the right, AZ is next.

DFS-GSA Follows the Newest Path

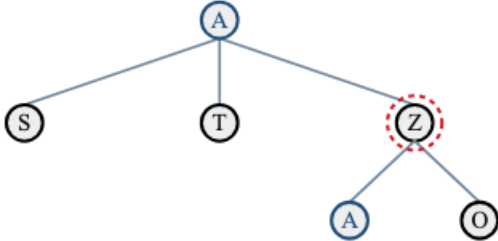
Slide 39 of 193 · Initial

Program Output

```
A -> ['AS', 'AT', 'AZ']
explored: ['A']

AZ -> [..., 'AZA', 'AZO']
explored: ['A', 'Z']
```

Search Tree



39

Speaker notes

Expand Z. AZO is newer than AZA, so AZO is selected next.

DFS-GSA Follows the Newest Path

Slide 39 of 193 · Reveal 1

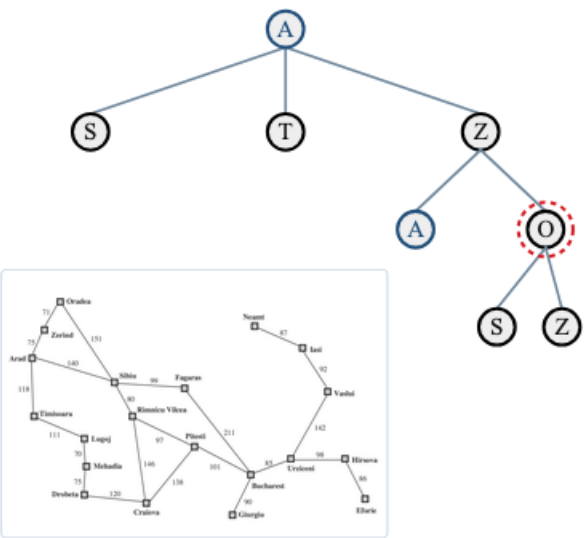
Program Output

```
A -> ['AS', 'AT', 'AZ']
explored: ['A']

AZ -> [..., 'AZA', 'AZO']
explored: ['A', 'Z']

AZO -> [..., 'AZOS', 'AZOZ']
explored: ['A', 'O', 'Z']
```

Search Tree



39

Speaker notes

Expand O. AZOZ is removed first but ends in an explored state, so it is skipped. AZOS is then expanded.

DFS-GSA Follows the Newest Path

Slide 39 of 193 · Reveal 2

Program Output

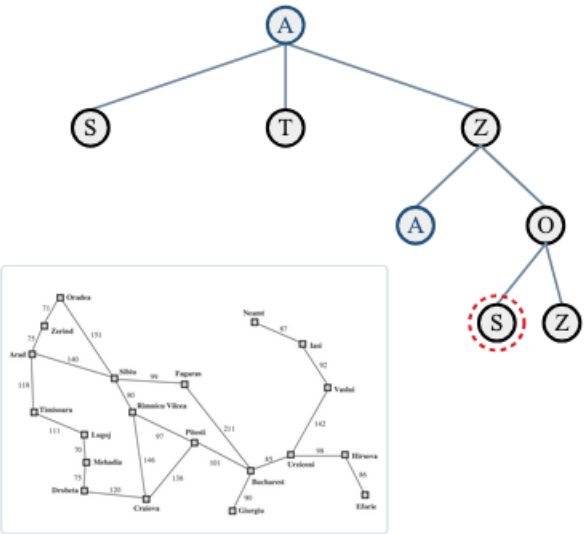
```
A -> ['AS', 'AT', 'AZ']
explored: ['A']

AZ -> [..., 'AZA', 'AZO']
explored: ['A', 'Z']

AZO -> [..., 'AZOS', 'AZOZ']
explored: ['A', 'O', 'Z']

AZOS -> [..., 'AZOSR']
explored: ['A', 'O', 'S', 'Z']
```

Search Tree



39

Speaker notes

At S, the newest successor is R. The trace is following one branch rather than completing an entire depth.

DFS-GSA Follows the Newest Path

Slide 39 of 193 · Reveal 3

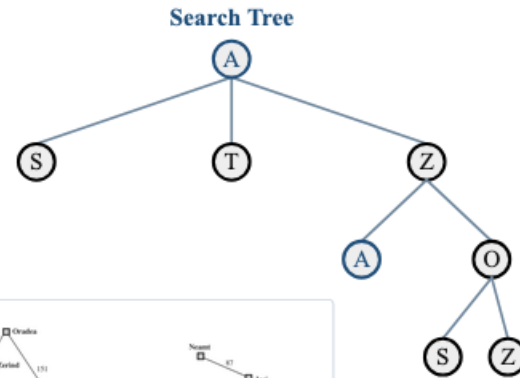
Program Output

```
A -> ['AS', 'AT', 'AZ']
explored: ['A']
```

```
AZ -> [..., 'AZA', 'AZ0']
explored: ['A', 'Z']
```

```
AZ0 -> [..., 'AZ0S', 'AZ0Z']
explored: ['A', '0', 'Z']
```

```
AZ0S -> [..., 'AZ0SR']
explored: ['A', '0', 'S', 'Z']
```



LIFO follows the newest path.

39

Speaker notes

Emphasize the invariant: the rightmost frontier path was added most recently and is removed next.

DFS-GSA Follows the Newest Path

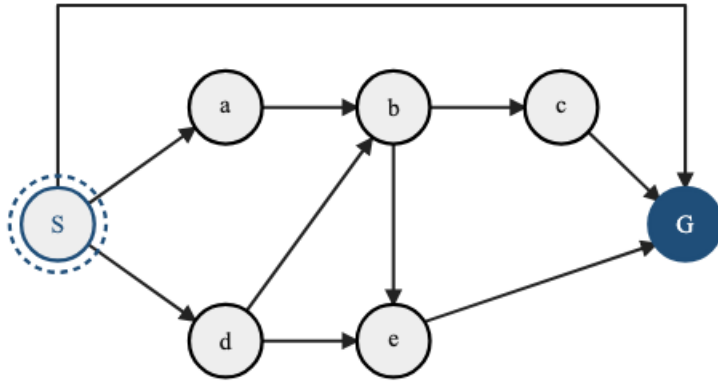
Slide 39 of 193 · Reveal 4

DFS-GSA Practice

TASKRun DFS-GSA from **S** to **G****Record** Explored order

Returned path

Add successors alphabetically



40

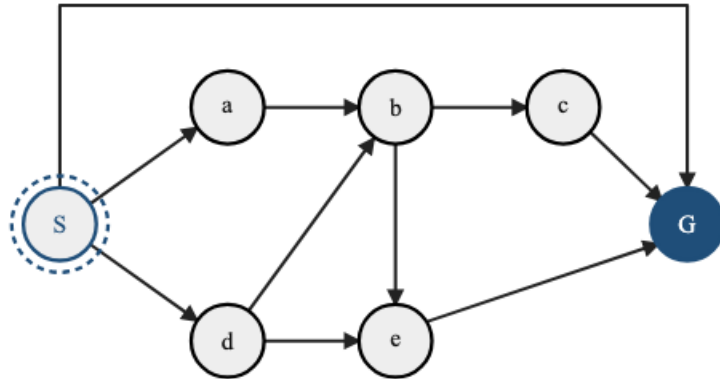
DFS-GSA Practice

Slide 40 of 193 · Initial

Speaker notes

Use the same graph and successor order as the BFS practice task. The only change is that DFS removes from the right of the frontier.

DFS-GSA Practice



TASK

Run DFS-GSA from S to G

Record Explored order

Returned path

Add successors alphabetically

SOLUTION

```

S -> ['Sa', 'Sd', 'SG']
explored: ['S']
  
```

40

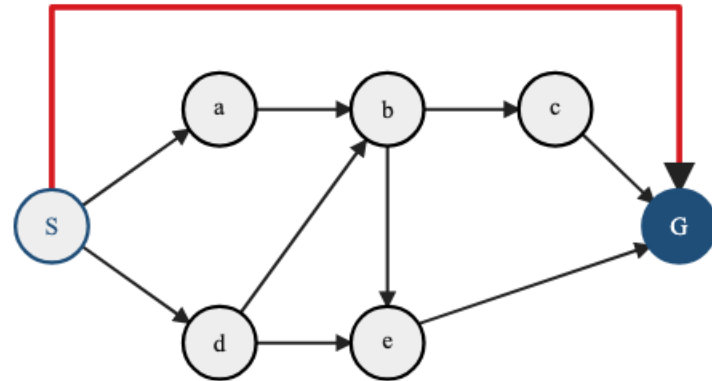
DFS-GSA Practice

Slide 40 of 193 · Reveal 1

Speaker notes

Expand S and append Sa, Sd, and SG alphabetically. The explored set is now S.

DFS-GSA Practice



TASK

Run DFS-GSA from S to G

Record Explored order
Returned path

Add successors alphabetically

SOLUTION

```

S -> ['Sa', 'Sd', 'SG']
explored: ['S']
solution: SG
  
```

Explored S Path S → G

40

Speaker notes

LIFO removes SG next because it was added last. The goal test succeeds, so DFS returns SG and no other state is explored.

DFS-GSA Practice

Slide 40 of 193 · Reveal 2

main.py

```

1 from collections import deque
2 def dfs_gsa(graph, start, goal, trace=False):
3     frontier = deque([start])
4     explored = set()
5     while frontier:
6         path = frontier.pop()
7         state = path[-1]
8         if state == goal: return path
9         if state in explored: continue
10        explored.add(state)
11        paths = [path + child for child in graph[state]]
12        frontier.extend(paths)
13        if trace:
14            print(f"{path} -> {list(frontier)}")
15            print(f"explored: {sorted(explored)}\n")
16 practice = {"S": ["a", "d", "G"], "a": ["b"],
17            "d": ["b", "e"], "b": ["c", "e"],
18            "c": ["G"], "e": ["G"], "G": []}
19 print("solution:", dfs_gsa(practice, "S", "G", True))

```

Download: comp3270.dir.hk/dfs_gsa_practice.py

41

Speaker notes

Run the DFS-GSA implementation on the same practice graph. Compare the single explored state with the three explored states produced by BFS-GSA.

Run the DFS-GSA Practice Code

Slide 41 of 193 · Initial

DFS-TSA Properties

TASK

Complete?

Optimal?

Time complexity?

Space complexity?

42

DFS-TSA Properties

Slide 42 of 193 · Initial

Speaker notes

Evaluate DFS-TSA using the same four questions used for BFS-TSA. There is no explored set.

DFS-TSA Properties

SOLUTION

Complete?

No

Optimal?

Time complexity?

Space complexity?

42

DFS-TSA Properties

Slide 42 of 193 · Reveal 1

Speaker notes

DFS-TSA is not complete in general because it can follow an infinite branch or cycle forever.

DFS-TSA Properties

SOLUTION

Complete?

No

unless the search tree is finite

Optimal?

Time complexity?

Space complexity?

42

Speaker notes

If the search tree is finite, every branch eventually ends and DFS-TSA becomes complete.

DFS-TSA Properties

Slide 42 of 193 · Reveal 2

DFS-TSA Properties

SOLUTION

Complete?

No

unless the search tree is finite

Optimal?

No

Time complexity?

Space complexity?

42

Speaker notes

DFS is not optimal. It can return a deep goal before considering a shallower or cheaper goal.

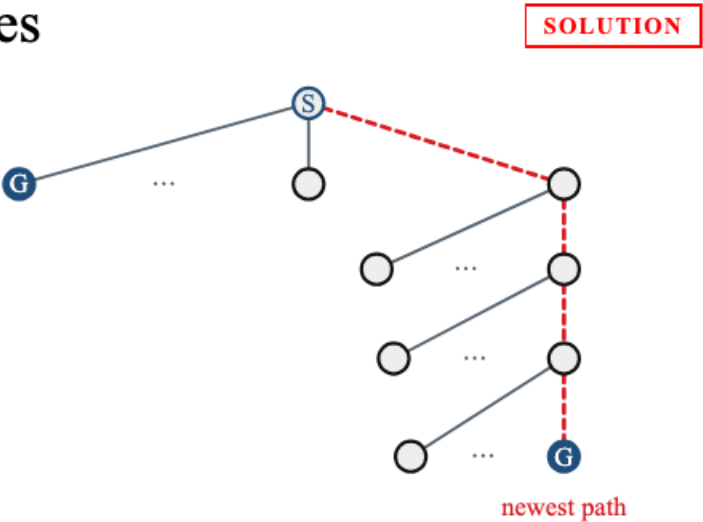
DFS-TSA Properties

Slide 42 of 193 · Reveal 3

DFS-TSA Properties

Complete?	0
No	
unless the search tree is finite	1
Optimal?	
No	2
Time complexity?	⋮
Space complexity?	m

b maximum branching factor
m maximum search-tree depth



42

Speaker notes

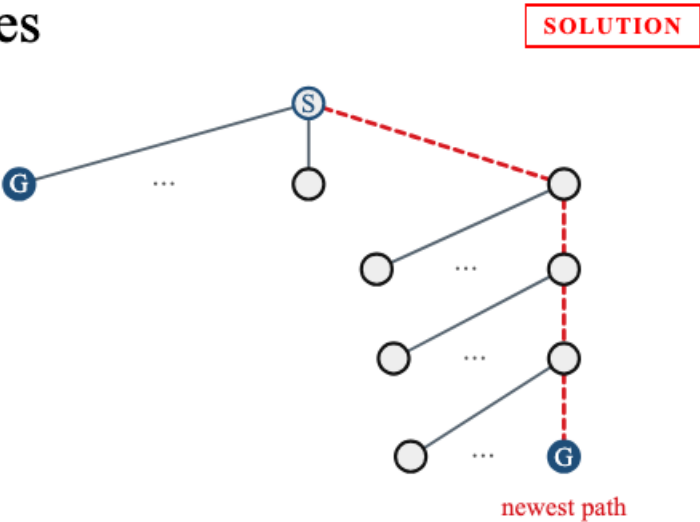
The diagram shows the key memory advantage: DFS follows one path while retaining the unexpanded sibling paths along that branch.

DFS-TSA Properties

DFS-TSA Properties

Complete?	0
No	
unless the search tree is finite	1
Optimal?	
No	2
Time complexity?	$\vdots$
$O(b^m)$	
Space complexity?	m

b maximum branching factor
m maximum search-tree depth



42

Speaker notes

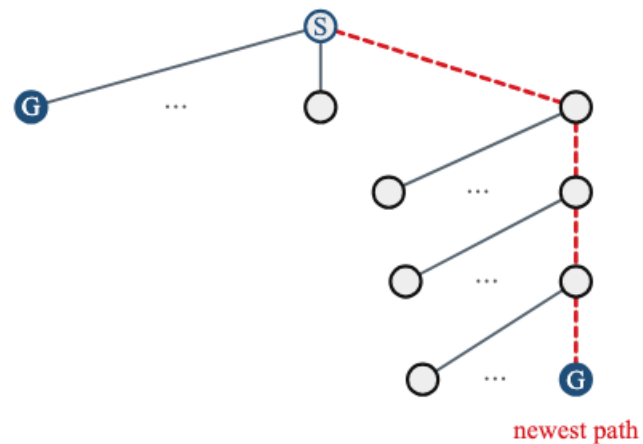
In the worst case, DFS explores a b-ary tree to maximum depth m, taking $O(b^m)$ time.

DFS-TSA Properties

DFS-TSA Properties

Complete?	0
No	
unless the search tree is finite	1
Optimal?	
No	2
Time complexity?	$\vdots$
$O(b^m)$	
Space complexity?	m
$O(bm)$	

b maximum branching factor
m maximum search-tree depth



42

Speaker notes

At each of m levels, the frontier retains at most $O(b)$ siblings, so tree-search space is $O(bm)$.

DFS-TSA Properties

Slide 42 of 193 · Reveal 6

BFS vs. DFS

TASK

BFS

When can it outperform DFS?

DFS

When can it outperform BFS?

43

Speaker notes

Ask students to compare the two strategies using the shape of the search tree. Neither answer is unconditional because successor order also matters.

BFS vs. DFS

Slide 43 of 193 · Initial

BFS vs. DFS

SOLUTION

BFS

When can it outperform DFS?

When a solution is shallow

DFS

When can it outperform BFS?

43

Speaker notes

BFS has an advantage when a solution is shallow because it examines the tree one depth at a time.

BFS vs. DFS

Slide 43 of 193 · Reveal 1

BFS vs. DFS

SOLUTION

BFS

When can it outperform DFS?

When a solution is shallow

DFS

When can it outperform BFS?

When a solution lies deep along an early branch

43

Speaker notes

DFS can reach a deep solution quickly when that solution lies along a branch explored early. It may still perform poorly if it first enters a long unproductive branch.

BFS vs. DFS

Slide 43 of 193 · Reveal 2

TSA vs. GSA

	TSA	GSA
Memory		
Repeated state		
Trade-off		

44

Speaker notes

Compare tree search and graph search one item at a time. The row labels establish the three questions before either answer appears.

TSA vs. GSA

TSA vs. GSA

	TSA	GSA
Memory	Frontier	
Repeated state		
Trade-off		

44

Speaker notes

Tree search stores only the frontier.

TSA vs. GSA

Slide 44 of 193 · Reveal 1

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state		
Trade-off		

44

Speaker notes

Graph search stores the frontier and an explored set.

TSA vs. GSA

Slide 44 of 193 · Reveal 2

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state	Expand it again	
Trade-off		

44

Speaker notes

Without an explored set, tree search can expand a state again.

TSA vs. GSA

Slide 44 of 193 · Reveal 3

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state	Expand it again	Skip it
Trade-off		

44

Speaker notes

Graph search skips a path when its final state has already been explored.

TSA vs. GSA

Slide 44 of 193 · Reveal 4

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state	Expand it again	Skip it
Trade-off	Less memory	

44

Speaker notes

Tree search uses less memory because it does not store explored states.

TSA vs. GSA

Slide 44 of 193 · Reveal 5

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state	Expand it again	Skip it
Trade-off	Less memory	Fewer repeated expansions

44

Speaker notes

Graph search often avoids repeated expansions, at the cost of storing the explored set.

TSA vs. GSA

Slide 44 of 193 · Reveal 6

TSA vs. GSA

	TSA	GSA
Memory	Frontier	Frontier + explored set
Repeated state	Expand it again	Skip it
Trade-off	Less memory	Fewer repeated expansions

BFS, DFS, and UCS can each use either algorithm

44

Speaker notes

This bookkeeping choice is independent of the frontier rule. BFS, DFS, and UCS can each use tree search or graph search.

TSA vs. GSA

Slide 44 of 193 · Reveal 7

UCS Explores the Cheapest Node First



frontier is a **priority queue** (smallest total path cost first)

45

Speaker notes

The mining picture is the uniform-cost counterpart to the BFS and DFS search illustrations. UCS does not commit to the shallowest or deepest path. It repeatedly expands the frontier path whose accumulated cost is smallest.

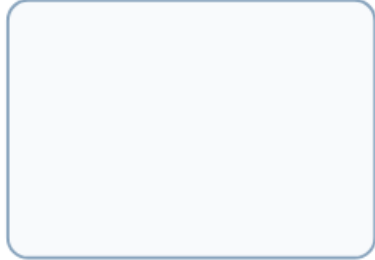
[Sources] - Recovered COMP3270 2024–25 source slide 43: 01_search.pptx. - UC Berkeley CS188 search illustration: <http://ai.berkeley.edu>.

UCS Explores the Cheapest Node First

Slide 45 of 193 · Initial

Frontier

smallest total path cost first



```
main.py
1 from heapq import heappop, heappush
2 frontier = []
3 heappush(frontier, (140, "AS"))
4 heappush(frontier, (118, "AT"))
5 heappush(frontier, (75, "AZ"))
6 print("frontier:", frontier)
7 while frontier:
8     print("remove:", heappop(frontier))
```

[Download: comp3270.dirk.hki/priority_queue_demo.py](#)

46

Speaker notes

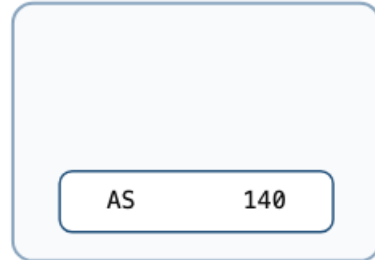
A priority queue orders frontier paths by priority. For UCS, the priority is the total path cost.

A Priority Queue Removes the Cheapest Path

Slide 46 of 193 · Initial

Frontier

smallest total path cost first

`heappush(frontier, (140, "AS"))`

```
main.py
1 from heapq import heappop, heappush
2 frontier = []
3 heappush(frontier, (140, "AS"))
4 heappush(frontier, (118, "AT"))
5 heappush(frontier, (75, "AZ"))
6 print("frontier:", frontier)
7 while frontier:
8     print("remove:", heappop(frontier))
```

[Download: comp3270.dirk.hki/priority_queue_demo.py](https://comp3270.dirk.hki/priority_queue_demo.py)

46

Speaker notes

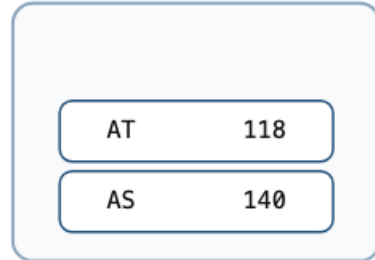
Insert AS with total cost 140.

A Priority Queue Removes the Cheapest Path

Slide 46 of 193 · Reveal 1

Frontier

smallest total path cost first



```
heappush(frontier, (140, "AS"))  
heappush(frontier, (118, "AT"))
```

```
main.py  
1 from heapq import heappop, heappush  
2 frontier = []  
3 heappush(frontier, (140, "AS"))  
4 heappush(frontier, (118, "AT"))  
5 heappush(frontier, (75, "AZ"))  
6 print("frontier:", frontier)  
7 while frontier:  
8     print("remove:", heappop(frontier))
```

[Download: comp3270.dirk.hki/priority_queue_demo.py](#)

46

Speaker notes

Insert AT with total cost 118. It has higher priority because its cost is smaller.

A Priority Queue Removes the Cheapest Path

Slide 46 of 193 · Reveal 2

Frontier

smallest total path cost first

AZ	75
AT	118
AS	140

```
heappush(frontier, (140, "AS"))  
heappush(frontier, (118, "AT"))  
heappush(frontier, (75, "AZ"))
```

```
main.py  
1 from heapq import heappop, heappush  
2 frontier = []  
3 heappush(frontier, (140, "AS"))  
4 heappush(frontier, (118, "AT"))  
5 heappush(frontier, (75, "AZ"))  
6 print("frontier:", frontier)  
7 while frontier:  
8     print("remove:", heappop(frontier))
```

[Download: comp3270.dirk.hki/priority_queue_demo.py](#)

46

Speaker notes

Insert AZ with total cost 75. It now has the highest priority.

A Priority Queue Removes the Cheapest Path

Slide 46 of 193 · Reveal 3

Frontier

smallest total path cost first

AT 118

AS 140

AZ 75

removed first

heappop(frontier)
returns (75, "AZ")

```
main.py
1 from heapq import heappop, heappush
2 frontier = []
3 heappush(frontier, (140, "AS"))
4 heappush(frontier, (118, "AT"))
5 heappush(frontier, (75, "AZ"))
6 print("frontier:", frontier)
7 while frontier:
8     print("remove:", heappop(frontier))
```

Download: comp3270.dirk.hki/priority_queue_demo.py

46

Speaker notes

heappop() removes AZ. The Python heap stores each entry as a tuple containing the cost and path. Its list representation need not be fully sorted; the guarantee is that heappop() removes the smallest tuple.

A Priority Queue Removes the Cheapest Path

Slide 46 of 193 · Reveal 4



frontier: (cost, path)

explored: states

main.py romania.py

```

1 from heapq import heappop, heappush
2 from romania import romania
3 def ucs_gsa(graph, start, goal, trace=False):
4     frontier = [(0, start)]
5     explored = set()
6     explored_order = []
7     while frontier:
8         cost, path = heappop(frontier)
9         state = path[-1]
10        if state == goal: return cost, path, explored_order
11        if state in explored: continue
12        explored.add(state)
13        explored_order.append(state)
14        for step_cost, child in graph[state]:
15            heappush(frontier, (cost + step_cost, path + child))
16        if trace:
17            print(f"{path} @ {cost} -> {sorted(frontier)}")
18            print(f"explored: {sorted(explored)}\n")
19    cost, path, order = ucs_gsa(romania, "A", "B", True)
20    print("solution:", (cost, path))
21    print("explored order:", " ".join(order))

```

Download: comp3270.dir.hk/ucs_gsa_romania.py

47

Speaker notes

The graph-search structure is unchanged. The frontier now stores cost-path pairs, and heappop() removes the pair with the smallest cost. The goal test still occurs when a path leaves the frontier.

[Sources] - AIMA, Chapter 3, uniform-cost search. - Romania route problem: <https://aima.cs.berkeley.edu/python/search.html>

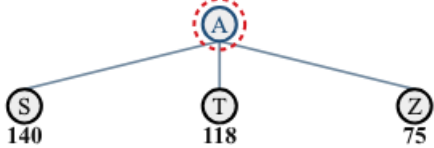
Run UCS-GSA on Romania

Slide 47 of 193 · Initial

Program Output

```
A @ 0 -> [(75, 'AZ'), (118, 'AT'),  
(140, 'AS')]  
explored: ['A']
```

Search Tree



48

Speaker notes

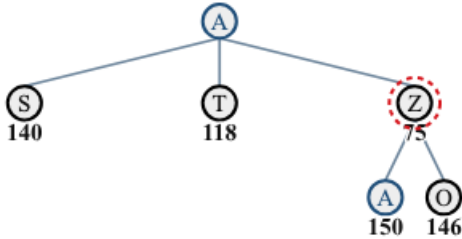
Expanding A inserts three paths. UCS selects AZ because 75 is the smallest total cost.

UCS-GSA Follows the Cheapest Path

Program Output

```
A @ 0 -> [(75, 'AZ'), (118, 'AT'),  
(140, 'AS')]  
explored: ['A']  
  
AZ @ 75 -> [(118, 'AT'), (140, 'AS'),  
...]  
explored: ['A', 'Z']
```

Search Tree



48

Speaker notes

Expanding Z adds AZA at cost 150 and AZO at cost 146. Neither jumps ahead of AT, which already costs 118.

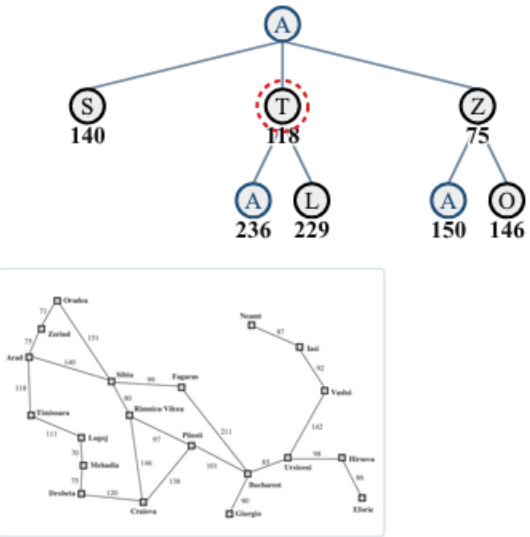
UCS-GSA Follows the Cheapest Path

Slide 48 of 193 · Reveal 1

Program Output

```
A @ 0 -> [(75, 'AZ'), (118, 'AT'), (140, 'AS')]  
explored: ['A']  
  
AZ @ 75 -> [(118, 'AT'), (140, 'AS'), ...]  
explored: ['A', 'Z']  
  
AT @ 118 -> [(140, 'AS'), (146, 'AZO'), ...]  
explored: ['A', 'T', 'Z']
```

Search Tree



48

Speaker notes

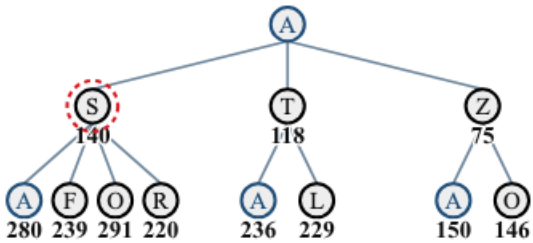
AT leaves next at cost 118 and generates ATA at cost 236 and ATL at cost 229.

UCS-GSA Follows the Cheapest Path

Program Output

```
A @ 0 -> [(75, 'AZ'), (118, 'AT'), (140, 'AS')]  
explored: ['A']  
  
AZ @ 75 -> [(118, 'AT'), (140, 'AS'), ...]  
explored: ['A', 'Z']  
  
AT @ 118 -> [(140, 'AS'), (146, 'AZO'), ...]  
explored: ['A', 'T', 'Z']  
  
AS @ 140 -> [(146, 'AZO'), (150, 'AZA'), ...]  
explored: ['A', 'S', 'T', 'Z']
```

Search Tree



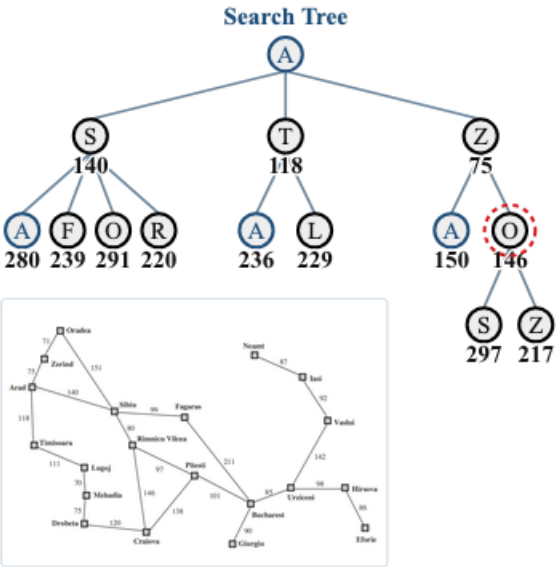
Speaker notes

AS follows at cost 140 and generates four children. ASR has the smallest of their costs at 220, but AZO is already waiting at cost 146.

UCS-GSA Follows the Cheapest Path

Program Output

```
A @ 0 -> [(75, 'AZ'), (118, 'AT'), (140, 'AS')]  
explored: ['A']  
  
AZ @ 75 -> [(118, 'AT'), (140, 'AS'), ...]  
explored: ['A', 'Z']  
  
AT @ 118 -> [(140, 'AS'), (146, 'AZO'), ...]  
explored: ['A', 'T', 'Z']  
  
AS @ 140 -> [(146, 'AZO'), (150, 'AZA'), ...]  
explored: ['A', 'S', 'T', 'Z']  
  
AZO @ 146 -> [(150, 'AZA'), (217, 'AZOZ'), ...]  
explored: ['A', 'O', 'S', 'T', 'Z']
```



48

Speaker notes

AZO therefore leaves next at cost 146 and generates AZOS at 297 and AZOZ at 217. The search may return to a shallower or deeper path whenever that path becomes cheapest.

UCS-GSA Follows the Cheapest Path

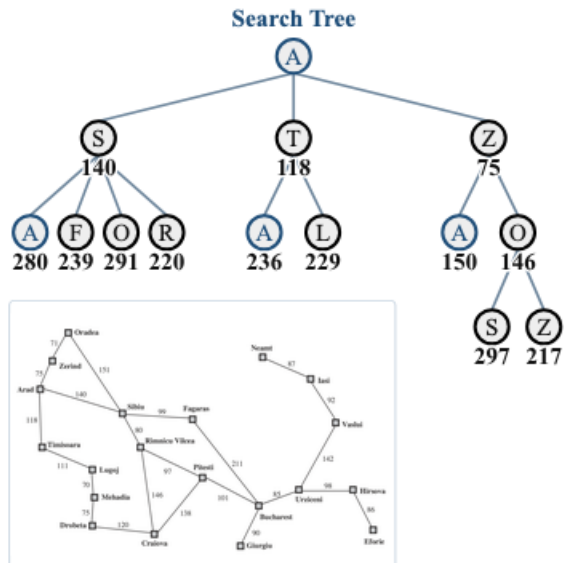
```
A @ 0 -> [(75, 'AZ'), (118, 'AT'),
(140, 'AS')]
explored: ['A']

AZ @ 75 -> [(118, 'AT'), (140, 'AS'),
...]
explored: ['A', 'Z']

AT @ 118 -> [(140, 'AS'), (146, 'AZO'),
...]
explored: ['A', 'T', 'Z']

AS @ 140 -> [(146, 'AZO'), (150,
'AZA'), ...]
explored: ['A', 'S', 'T', 'Z']

AZO @ 146 -> [(150, 'AZA'), (217,
'AZOZ'), ...]
explored: ['A', 'O', 'S', 'T', 'Z']
```



Next path = smallest total path cost

48

The invariant is total path cost, not depth and not insertion time.

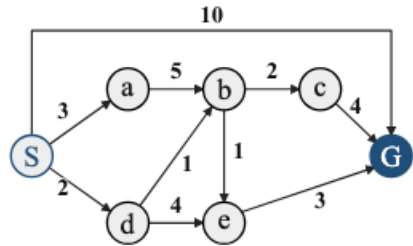
UCS-GSA Practice

TASK

 Run UCS-GSA from **S** to **G**

Record Explored order
 Returned path and cost

Add successors alphabetically



49

Speaker notes

Ask students to trace UCS-GSA. They must report the explored order, returned path, and total path cost. Successors are listed alphabetically, while the priority queue removes the smallest cost.

UCS-GSA Practice

Slide 49 of 193 · Initial

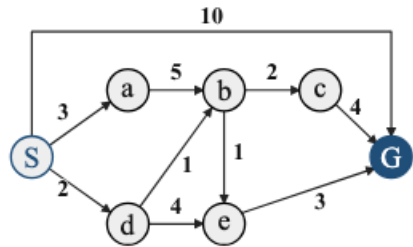
UCS-GSA Practice

SOLUTION

Run UCS-GSA from S to G

Record Explored order
 Returned path and cost

Add successors alphabetically



```
S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']
```

49

Speaker notes

S generates Sd at cost 2, Sa at cost 3, and SG at cost 10.

UCS-GSA Practice

Slide 49 of 193 · Reveal 1

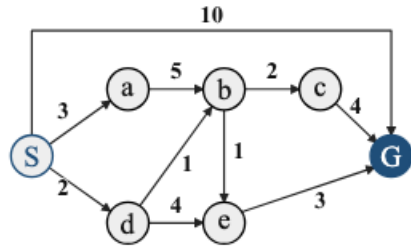
UCS-GSA Practice

SOLUTION

 Run UCS-GSA from **S** to **G**
Record Explored order

Returned path and cost

Add successors alphabetically



```
S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']
```

```
Sd @ 2 -> [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']
```

49

Speaker notes

Sd is cheapest. Expanding d adds Sdb at cost 3 and Sde at cost 6.

UCS-GSA Practice

Slide 49 of 193 · Reveal 2

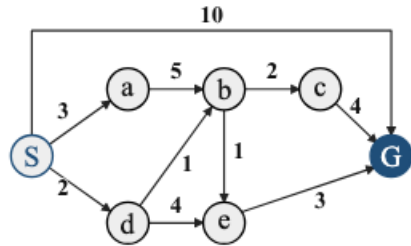
UCS-GSA Practice

SOLUTION

 Run UCS-GSA from **S** to **G**
Record Explored order

Returned path and cost

Add successors alphabetically



```
S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']
```

```
Sd @ 2 -> [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']
```

```
Sa @ 3 -> [(3, 'Sdb'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'd']
```

49

UCS-GSA Practice

Slide 49 of 193 · Reveal 3

Speaker notes

Sa and Sdb tie at cost 3. Python's tuple ordering removes Sa first, and expanding a adds Sab at cost 8.

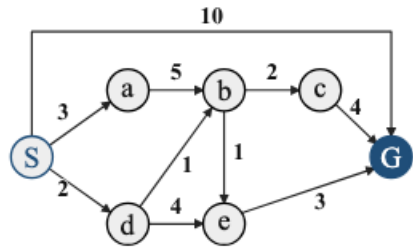
UCS-GSA Practice

SOLUTION

Run UCS-GSA from S to G

Record Explored order
 Returned path and cost

Add successors alphabetically



S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']

Sd @ 2 -> [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']

Sa @ 3 -> [(3, 'Sdb'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'd']

Sdb @ 3 -> [(4, 'Sdbe'), (5, 'Sdbc'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd']

Speaker notes

Sdb now leaves at cost 3. Expanding b adds Sdbe at cost 4 and Sdbc at cost 5.

UCS-GSA Practice

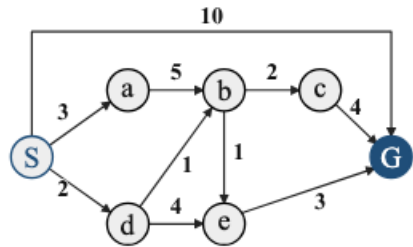
UCS-GSA Practice

SOLUTION

Run UCS-GSA from S to G

Record Explored order
 Returned path and cost

Add successors alphabetically



S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']

Sd @ 2 -> [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']

Sa @ 3 -> [(3, 'Sdb'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'd']

Sdb @ 3 -> [(4, 'Sdbe'), (5, 'Sdbc'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd']

Sdbe @ 4 -> [(5, 'Sdbc'), (6, 'Sde'), (7, 'SdbeG'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd', 'e']

49

Speaker notes

Sdbe is cheapest at cost 4. Expanding e adds the goal path SdbeG at cost 7.

UCS-GSA Practice

Slide 49 of 193 · Reveal 5

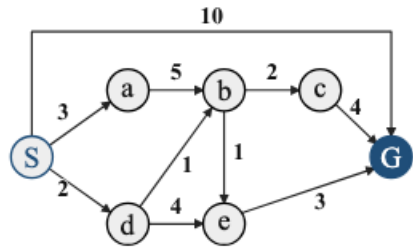
UCS-GSA Practice

SOLUTION

Run UCS-GSA from S to G

Record Explored order
Returned path and cost

Add successors alphabetically



S @ 0 -> [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']

Sd @ 2 -> [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']

Sa @ 3 -> [(3, 'Sdb'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'd']

Sdb @ 3 -> [(4, 'Sdbe'), (5, 'Sdbc'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd']

Sdbe @ 4 -> [(5, 'Sdbc'), (6, 'Sde'), (7, 'SdbeG'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd', 'e']

Sdbc @ 5 -> [(6, 'Sde'), (7, 'SdbeG'), (8, 'Sab'), (9, 'SdbcG'), (10, 'SG')]
explored: ['S', 'a', 'b', 'c', 'd', 'e']

49

Speaker notes

Sdbc still costs only 5, so c is explored before the goal path is removed.

UCS-GSA Practice

Slide 49 of 193 · Reveal 6

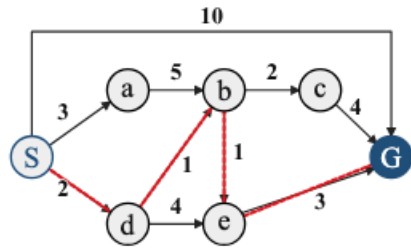
UCS-GSA Practice

SOLUTION

Run UCS-GSA from S to G

Record Explored order
 Returned path and cost

Add successors alphabetically



Solution (7, 'SdbeG')
Explored S → d → a → b → e → c
Path S → d → b → e → G

S @ 0 → [(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
explored: ['S']

Sd @ 2 → [(3, 'Sa'), (3, 'Sdb'), (6, 'Sde'), (10, 'SG')]
explored: ['S', 'd']

Sa @ 3 → [(3, 'Sdb'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'd']

Sdb @ 3 → [(4, 'Sdbe'), (5, 'Sdbc'), (6, 'Sde'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd']

Sdbe @ 4 → [(5, 'Sdbc'), (6, 'Sde'), (7, 'SdbeG'), (8, 'Sab'), (10, 'SG')]
explored: ['S', 'a', 'b', 'd', 'e']

Sdbc @ 5 → [(6, 'Sde'), (7, 'SdbeG'), (8, 'Sab'), (9, 'SdbcG'), (10, 'SG')]
explored: ['S', 'a', 'b', 'c', 'd', 'e']

49

Speaker notes

The goal path SdbeG leaves at cost 7 before the direct path SG at cost 10.
Separate the returned path from the explored order.

UCS-GSA Practice

main.py

```

1 from heapq import heappop, heappush
2 def ucs_gsa(graph, start, goal, trace=False):
3     frontier = [(0, start)]
4     explored = set()
5     explored_order = []
6     while frontier:
7         cost, path = heappop(frontier)
8         state = path[-1]
9         if state == goal: return cost, path, explored_order
10        if state in explored: continue
11        explored.add(state)
12        explored_order.append(state)
13        for step_cost, child in graph[state]:
14            heappush(frontier, (cost + step_cost, path + child))
15        if trace:
16            print(f"{path} @ {cost} -> {sorted(frontier)}")
17            print(f"explored: {sorted(explored)}\n")
18 practice = {"S": [(3, "a"), (2, "d"), (10, "G")], "a": [(5, "b")],
19            "d": [(1, "b"), (4, "e")], "b": [(2, "c"), (1, "e")],
20            "c": [(4, "G")], "e": [(3, "G")], "G": []}
21 cost, path, order = ucs_gsa(practice, "S", "G", True)
22 print("solution:", (cost, path))
23 print("explored order:", " -> ".join(order))

```

Download: comp3270.dirk.hk/ucs_gsa_practice.py

50

Speaker notes

Run the program to reproduce the trace. Point out that the implementation differs from BFS and DFS only in the frontier representation and the cost added when a successor path enters the frontier.

Run the UCS-GSA Practice Code

Slide 50 of 193 · Initial

UCS-TSA Properties

TASK

Complete?

Optimal?

Time complexity?

Space complexity?

51

UCS-TSA Properties

Slide 51 of 193 · Initial

Speaker notes

Evaluate uniform-cost tree search using the same four questions as BFS and DFS.

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

Optimal?

Time complexity?

Space complexity?

51

Speaker notes

Throughout our examples, step costs are nonnegative. Completeness needs the stronger condition that each step costs at least some positive ϵ ; otherwise infinitely many zero-cost steps could fit below a finite solution cost.

UCS-TSA Properties

Slide 51 of 193 · Reveal 1

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

Optimal?

Yes

goal test on removal

Time complexity?

Space complexity?

51

Speaker notes

UCS is optimal when the goal test occurs after the path leaves the priority queue. At that point no cheaper path remains.

UCS-TSA Properties

Slide 51 of 193 · Reveal 2

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

Optimal?

Yes

goal test on removal

Time complexity?

Space complexity?

C^* cost of an optimal solution

ϵ smallest positive step cost

51

Speaker notes

C^* denotes the optimal solution cost. Epsilon denotes the smallest positive step cost.

UCS-TSA Properties

Slide 51 of 193 · Reveal 3

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

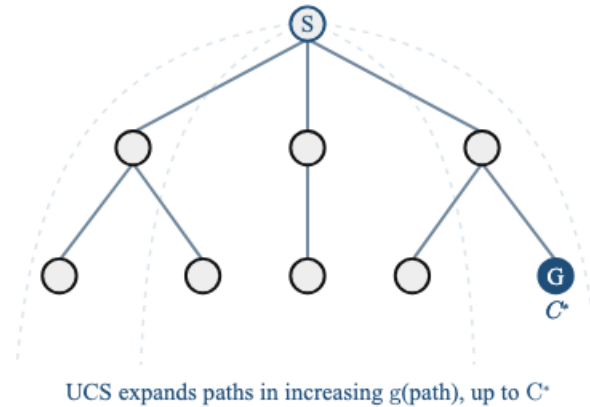
Optimal?

Yes

goal test on removal

Time complexity?

Space complexity?



C^* cost of an optimal solution

ϵ smallest positive step cost

51

Speaker notes

The diagram replaces depth layers with cost contours. The old ϵ , 2ϵ , and 3ϵ labels have been removed because siblings at the same depth need not have those exact costs.

UCS-TSA Properties

Slide 51 of 193 · Reveal 4

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

Optimal?

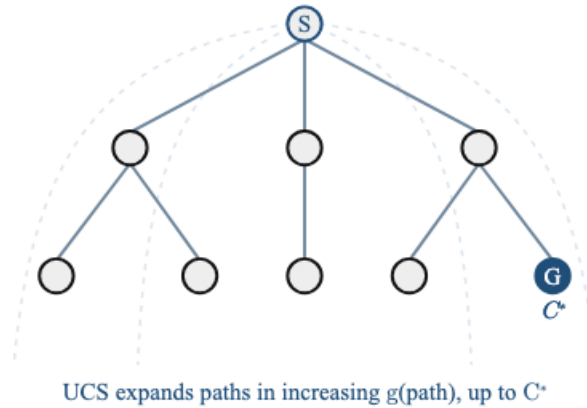
Yes

goal test on removal

Time complexity?

$$O(b^{\lceil C^*/\epsilon \rceil + 1})$$

Space complexity?



C^* cost of an optimal solution

ϵ smallest positive step cost

51

Speaker notes

A path costing at most C^* can contain at most floor of C^*/ϵ steps. Including the next generated layer gives the stated exponential time bound.

UCS-TSA Properties

Slide 51 of 193 · Reveal 5

UCS-TSA Properties

SOLUTION

Optimality allows $c \geq 0$; completeness requires $c \geq \epsilon > 0$

Complete?

Yes, if

finite b ; $c \geq \epsilon > 0$

Optimal?

Yes

goal test on removal

Time complexity?

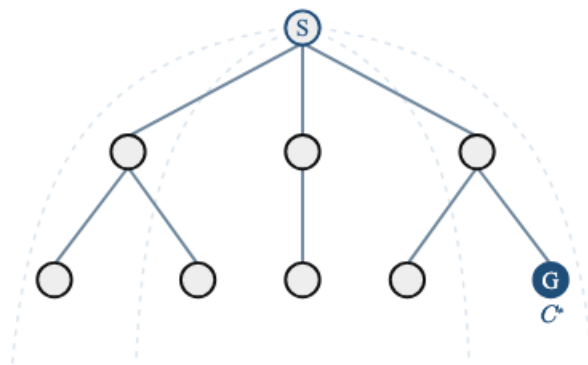
$$O(b^{\lceil C^*/\epsilon \rceil + 1})$$

Space complexity?

$$O(b^{\lceil C^*/\epsilon \rceil + 1})$$

C^* cost of an optimal solution

ϵ smallest positive step cost



UCS expands paths in increasing $g(\text{path})$, up to C^*

51

Speaker notes

The frontier can retain the generated paths, so the space bound matches the time bound.

UCS-TSA Properties

Slide 51 of 193 · Reveal 6

Uninformed Search Strategies

Strategy	Frontier	Next path	Optimal?
BFS	FIFO queue	Shallowest	Only with equal step costs

52

Speaker notes

BFS uses a FIFO frontier and selects a shallowest path. It is optimal only when every step has the same cost.

Uninformed Search Strategies

Uninformed Search Strategies

Strategy	Frontier	Next path	Optimal?
BFS	FIFO queue	Shallowest	Only with equal step costs
DFS	LIFO stack	Deepest	No

52

Speaker notes

DFS changes the same frontier to LIFO and follows a deepest path. It does not guarantee the least-cost solution.

Uninformed Search Strategies

Uninformed Search Strategies

Strategy	Frontier	Next path	Optimal?
BFS	FIFO queue	Shallowest	Only with equal step costs
DFS	LIFO stack	Deepest	No
UCS	Priority queue	Cheapest	Yes

UCS uses only costs already incurred: $g(\text{path})$

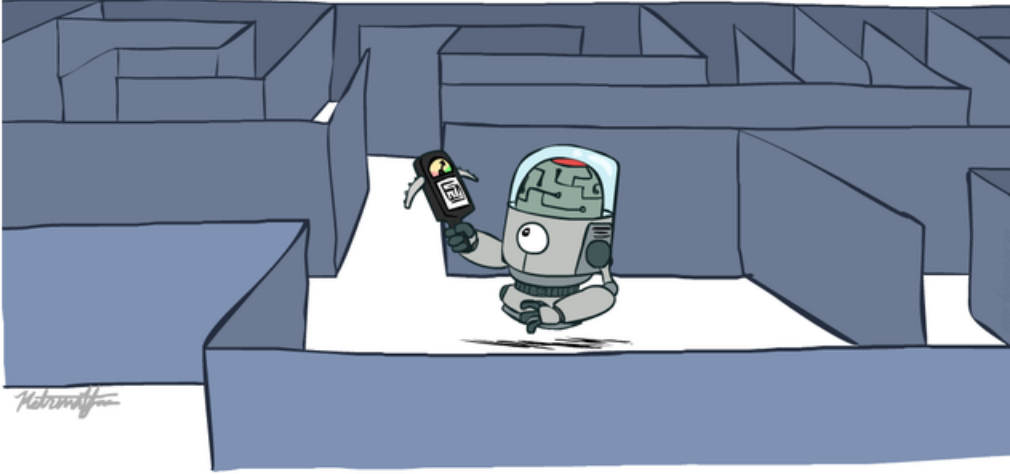
52

Speaker notes

UCS orders the frontier by the cost already incurred. It is still uninformed because it has no estimate of the cost remaining to a goal. That missing estimate motivates informed search.

Uninformed Search Strategies

Informed Search



53

Informed Search

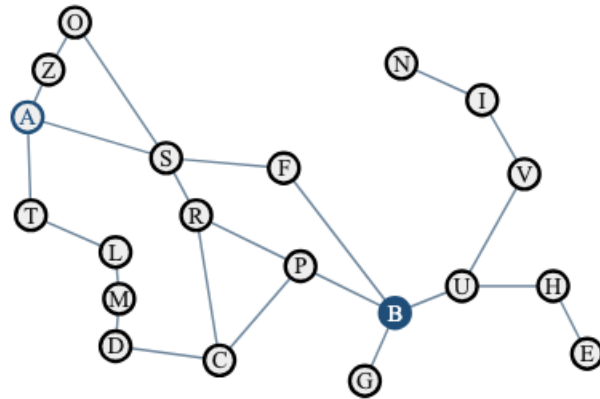
Slide 53 of 193 · Initial

Speaker notes

[Sources] - Recovered COMP3270 2024-25 source slide 53:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

A Heuristic Estimates the Remaining Cost

How far is a node from the goal?



$h(n)$: Estimated cost of the cheapest path from the state at node n to a goal

54

A Heuristic Estimates the Remaining Cost

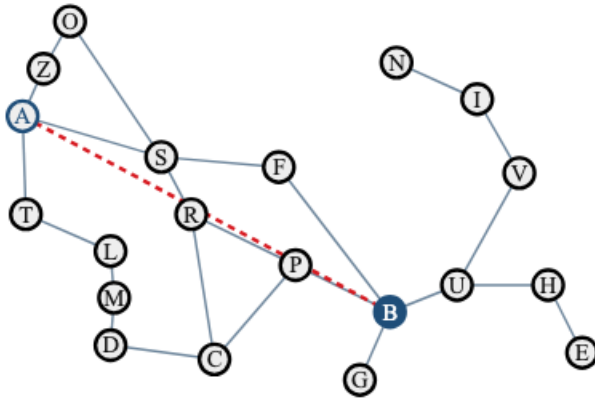
Slide 54 of 193 · Initial

Speaker notes

UCS knows the cost so far. A heuristic adds an estimate of the cost still to come.

A Heuristic Estimates the Remaining Cost

How far is a node from the goal?



$h(n)$: Estimated cost of the cheapest path from the state at node n to a goal

Straight-line distance to B

$$h(A) = 366 \text{ km}$$

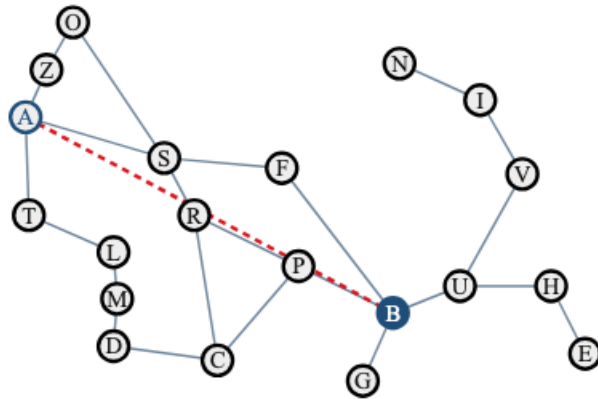
An estimate, not a road route

Speaker notes

For Romania, use straight-line distance to B. A's estimate is 366 km. The dashed line is an estimate, not an available road.

A Heuristic Estimates the Remaining Cost

How far is a node from the goal?



$h(n)$: Estimated cost of the cheapest path from the state at node n to a goal

Straight-line distance to B

$h(A) = 366$ km

An estimate, not a road route

$h(B) = 0$

Use the estimate to guide which frontier node to explore next

54

Speaker notes

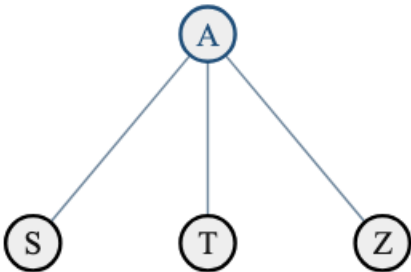
At the goal, the remaining cost is zero. Next we will use these estimates to decide which frontier node to explore.

A Heuristic Estimates the Remaining Cost

Slide 54 of 193 · Reveal 2

Greedy Search Selects the Lowest $h(n)$

Expand the frontier node estimated closest to the goal



Frontier after expanding A

Priority queue · smallest h first

Path	Priority
AS	$h(S) = 253$
AT	$h(T) = 329$
AZ	$h(Z) = 374$

55

Speaker notes

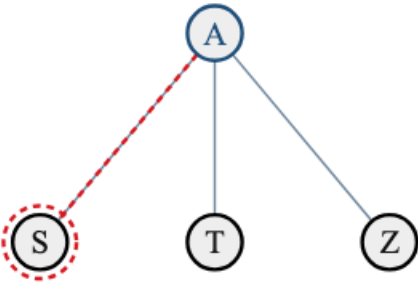
This is greedy best-first search. The frontier still stores paths in a priority queue. Each priority is h at the path's final state. Frontier after expanding A, compare the three estimates to B.

Greedy Search Selects the Lowest $h(n)$

Slide 55 of 193 · Initial

Greedy Search Selects the Lowest $h(n)$

Expand the frontier node estimated closest to the goal



Frontier after expanding A

Priority queue · smallest h first

Path	Priority
AS	$h(S) = 253$
AT	$h(T) = 329$
AZ	$h(Z) = 374$

Remove AS; explore S next

55

Speaker notes

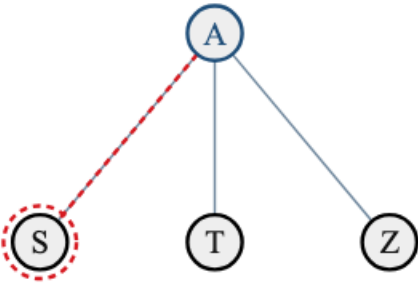
Path AS ends at S, whose estimate is 253. Remove AS and explore S next. This number is the estimated remaining distance, not the cost of reaching S.

Greedy Search Selects the Lowest $h(n)$

Slide 55 of 193 · Reveal 1

Greedy Search Selects the Lowest $h(n)$

Expand the frontier node estimated closest to the goal



Frontier after expanding A

Priority queue · smallest h first

Path	Priority
AS	$h(S) = 253$
AT	$h(T) = 329$
AZ	$h(Z) = 374$

Remove AS; explore S next

Greedy uses the estimated cost remaining; UCS uses the cost so far

55

Speaker notes

UCS ranks by cost so far; greedy ranks by estimated cost remaining. Being closest by the estimate does not guarantee the cheapest complete route.

Greedy Search Selects the Lowest $h(n)$

Slide 55 of 193 · Reveal 2

Greedy Follows the Heuristic

Select the frontier node with the smallest $h(n)$



56

Greedy Follows the Heuristic

Slide 56 of 193 · Initial

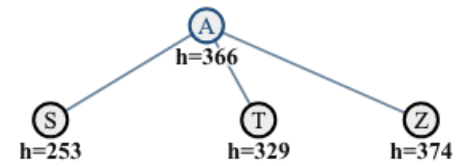
Speaker notes

The original mining illustration suggests following a promising direction. Greedy uses only the estimated remaining cost, not the cost already paid. Ask what this focus might overlook.

Search Trace · @ = h(final state)

```
A @ 366 →  
[(253, 'AS'), (329, 'AT'), (374, 'AZ')]  
explored: ['A']
```

Search Tree



57

Speaker notes

Expand A. The frontier paths are AS, AT and AZ. Their priorities are the estimates at S, T and Z, not the costs travelled.

Greedy-GSA Follows the Smallest Estimate

Slide 57 of 193 · Initial

Search Trace · @ = h(final state)

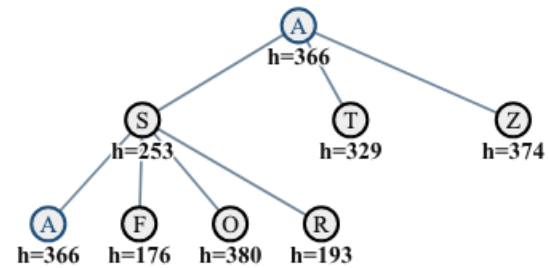
```

A @ 366 →
[(253, 'AS'), (329, 'AT'), (374, 'AZ')]
explored: ['A']

AS @ 253 →
[(176, 'ASF'), (193, 'ASR'), ...]
explored: ['A', 'S']

```

Search Tree



57

Speaker notes

Remove AS and expand S. Its children enter the frontier; ASF has priority 176, lower than ASR at 193.

Greedy-GSA Follows the Smallest Estimate

Slide 57 of 193 · Reveal 1

Search Trace · @ = h(final state)

```

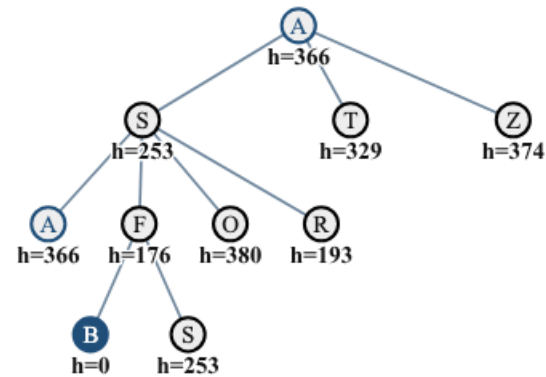
A @ 366 →
[(253, 'AS'), (329, 'AT'), (374, 'AZ')]
explored: ['A']

AS @ 253 →
[(176, 'ASF'), (193, 'ASR'), ...]
explored: ['A', 'S']

ASF @ 176 →
[(0, 'ASFB'), (193, 'ASR'), ...]
explored: ['A', 'F', 'S']

```

Search Tree



57

Speaker notes

Remove ASF and expand F. ASFB enters with priority zero. The path ASR is still waiting.

Greedy-GSA Follows the Smallest Estimate

Slide 57 of 193 · Reveal 2

Search Trace · @ = h(final state)

```

A @ 366 →
[(253, 'AS'), (329, 'AT'), (374, 'AZ')]
explored: ['A']

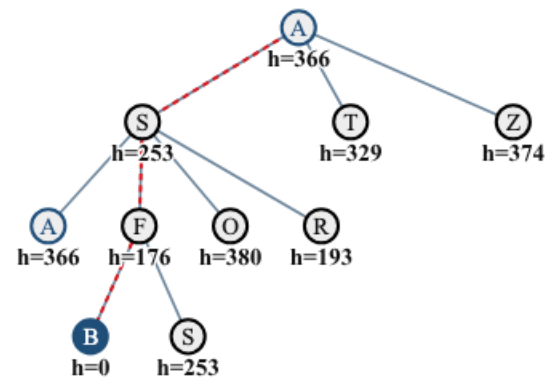
AS @ 253 →
[(176, 'ASF'), (193, 'ASR'), ...]
explored: ['A', 'S']

ASF @ 176 →
[(0, 'ASFB'), (193, 'ASR'), ...]
explored: ['A', 'F', 'S']

ASFB @ 0 → goal B
solution: ASFB

```

Search Tree



57

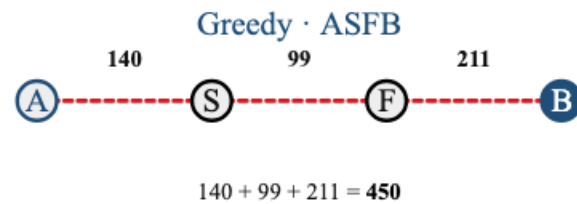
Speaker notes

Remove ASFB. Its final state B passes the goal test, so return the path before adding B to explored. The zero is h(B), not the route cost.

Greedy-GSA Follows the Smallest Estimate

Slide 57 of 193 · Reveal 3

Greedy Search Is Not Always Optimal



58

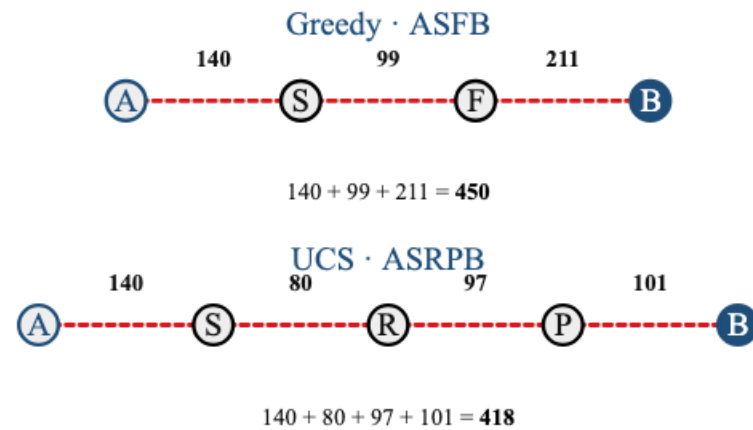
Speaker notes

Greedy returned ASFB. Add the actual step costs: 140 plus 99 plus 211 gives 450. The heuristic priority is separate from this path cost.

Greedy Search Is Not Always Optimal

Slide 58 of 193 · Initial

Greedy Search Is Not Always Optimal



58

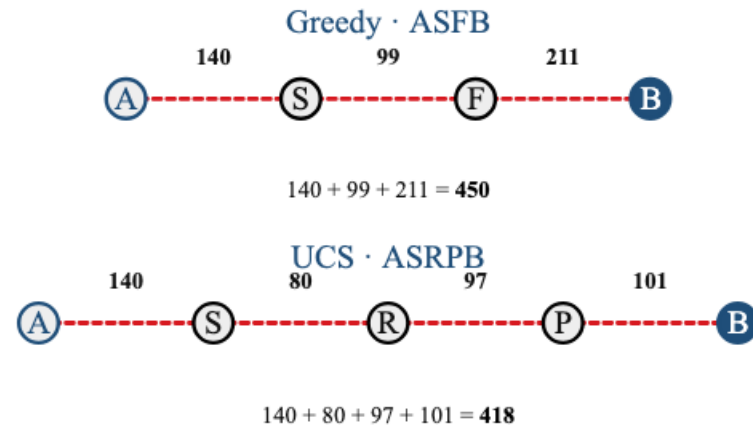
Speaker notes

UCS returns ASRPB at cost 418. This cheaper route is a counterexample to greedy optimality, even with straight-line distance as the heuristic.

Greedy Search Is Not Always Optimal

Slide 58 of 193 · Reveal 1

Greedy Search Is Not Always Optimal



A smaller h-value does not guarantee a cheaper complete path

58

Speaker notes

Greedy ignores the cost already paid. Next we can combine cost so far and estimated cost remaining.

Greedy Search Is Not Always Optimal

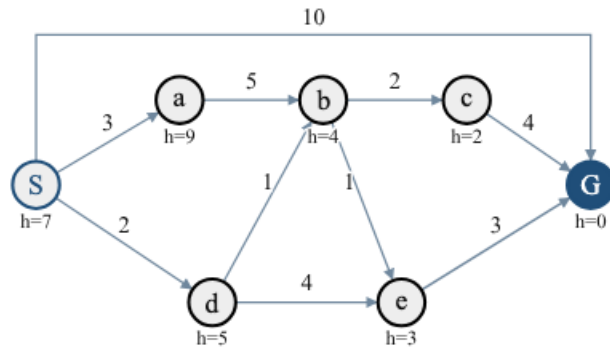
Slide 58 of 193 · Reveal 2

Greedy-TSA Practice

Start: S · Goal: G

Which states are expanded?

Which path is returned? Is it the cheapest?



59

Greedy-TSA Practice

Slide 59 of 193 · Initial

Speaker notes

Use the same graph as the earlier BFS, DFS and UCS exercises. Ask students to predict the expansion order and returned path. Remind them that greedy uses only h . Invite predictions before revealing the frontier.

Greedy-TSA Practice

Start: S · Goal: G

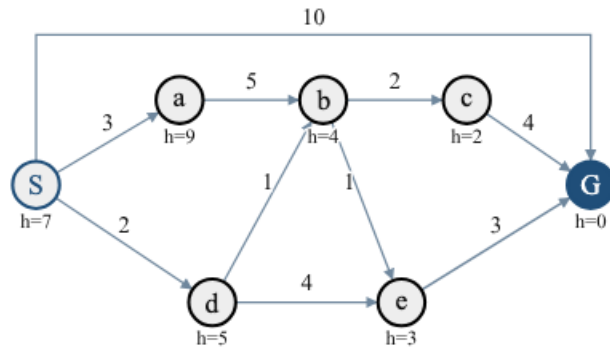
Which states are expanded?

Which path is returned? Is it the cheapest?

Initial frontier: [(7, 'S')]

Exploring: S at $h = 7$

Frontier:
[(0, 'SG'), (5, 'Sd'), (9, 'Sa')]



59

Greedy-TSA Practice

Slide 59 of 193 · Reveal 1

Speaker notes

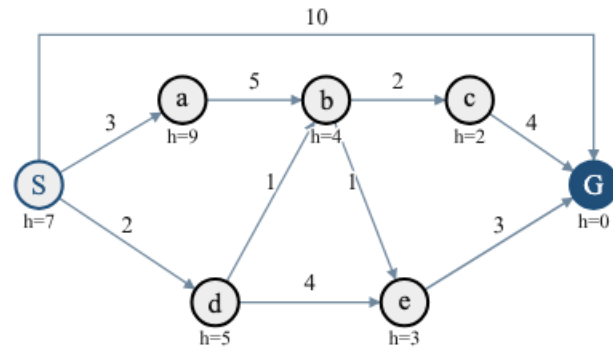
SG has the lowest heuristic priority, zero.

Greedy-TSA Practice

Start: S · Goal: G

Which states are expanded?

Which path is returned? Is it the cheapest?



Initial frontier: [(7, 'S')]

Exploring: S at h = 7

Frontier:
[(0, 'SG'), (5, 'Sd'), (9, 'Sa')]

Solution path: SG

59

Speaker notes

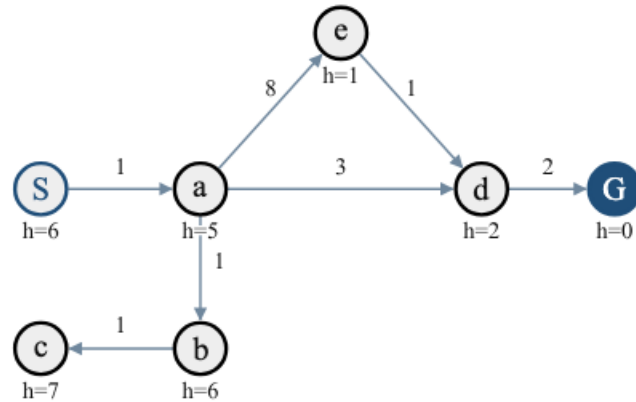
The goal test succeeds before expansion, so G is not expanded. SG costs 10; UCS returned SdbeG at cost 7. Greedy selected using h only.

Greedy-TSA Practice

Slide 59 of 193 · Reveal 2

A* Motivation: UCS

Cheap so far can still lead away from the goal



After expanding a

Sab: $g=2$

Sad: $g=4$

Sae: $g=9$

60

A* Motivation: UCS

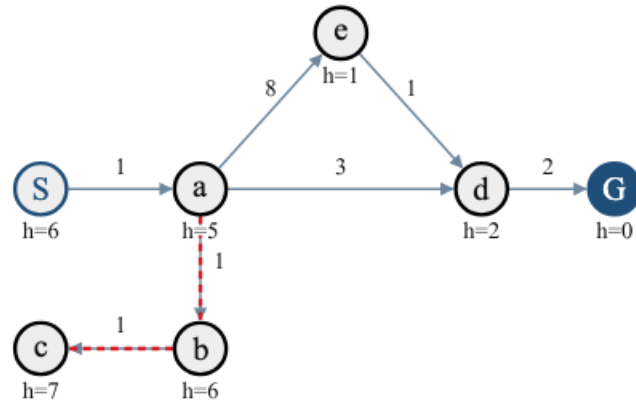
Slide 60 of 193 · Initial

Speaker notes

Keep the previous offering's shared example. Start at S and seek G. After S and a are expanded, UCS compares the three path costs shown.

A* Motivation: UCS

Cheap so far can still lead away from the goal



After expanding a

Sab: $g=2$

Sad: $g=4$

Sae: $g=9$

Follow the cheap branch

Sab $\rightarrow$ Sabc

c is a dead end

60

A* Motivation: UCS

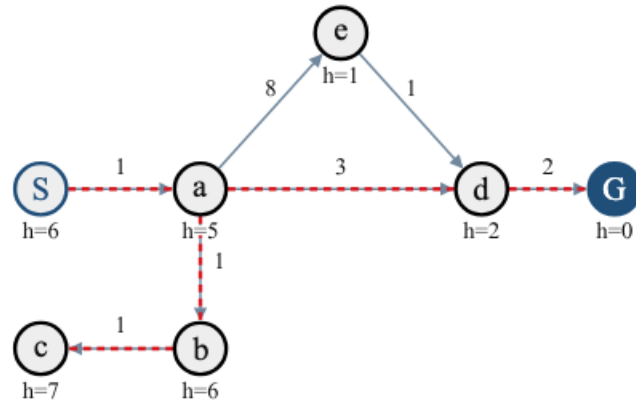
Slide 60 of 193 · Reveal 1

Speaker notes

UCS selects Sab at cost 2, then Sabc at cost 3, before Sad at cost 4. The cheap branch leads to a dead end. UCS has no guidance about the goal.

A* Motivation: UCS

Cheap so far can still lead away from the goal



After expanding a

Sab: $g=2$

Sad: $g=4$

Sae: $g=9$

Follow the cheap branch

Sab $\rightarrow$ Sabc

c is a dead end

Return to the waiting path

Sad $\rightarrow$ SadG

Solution cost: 6

60

A* Motivation: UCS

Slide 60 of 193 · Reveal 2

Speaker notes

UCS returns to Sad and reaches G at cost 6. It finds the best route here, but spends expansions on b and c first.

A* Motivation: Greedy Search

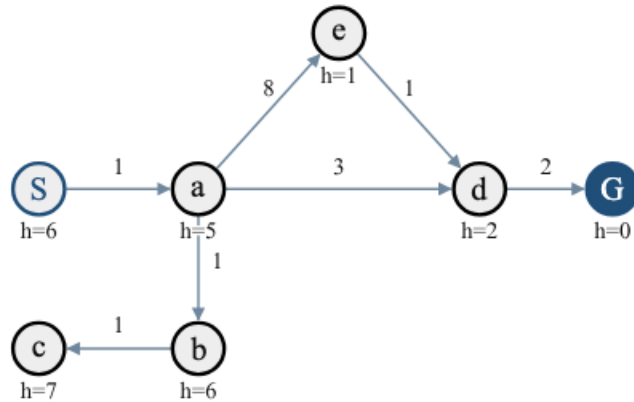
Greedy selects using $h(n)$ alone

Same frontier, new priority

Sab: $h(b)=6$

Sad: $h(d)=2$

Sae: $h(e)=1$



61

A* Motivation: Greedy Search

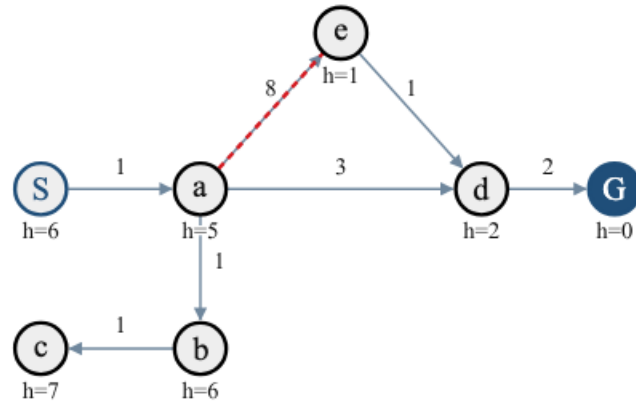
Slide 61 of 193 · Initial

Speaker notes

Use exactly the same graph. Greedy ignores the path cost and compares the estimates at b, d and e.

A* Motivation: Greedy Search

Greedy selects using $h(n)$ alone



Same frontier, new priority

Sab: $h(b)=6$

Sad: $h(d)=2$

Sae: $h(e)=1$

Chase the smallest estimate

Sae is selected first

$h(e)=1$ is the smallest estimate

61

A* Motivation: Greedy Search

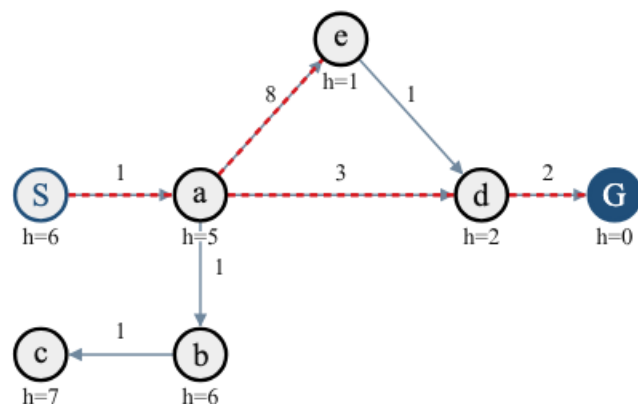
Slide 61 of 193 · Reveal 1

Speaker notes

Greedy expands e because $h(e)=1$. It does not use the path cost. Expanding e generates Saed, but Sad is still waiting in the frontier.

A* Motivation: Greedy Search

Greedy selects using $h(n)$ alone



Same frontier, new priority

Sab: $h(b)=6$

Sad: $h(d)=2$

Sae: $h(e)=1$

Chase the smallest estimate

Sae is selected first

$h(e)=1$ is the smallest estimate

Then select Sad

Sad and Saed tie at $h=2$

Path-string order selects Sad

61

Speaker notes

Sad and Saed both end at d and tie at $h=2$. Our tuple ordering chooses Sad alphabetically. This example still returns the cost-6 path; the problem shown here is the unnecessary expansion of e, not a claim that this trace returns a worse route.

A* Motivation: Greedy Search

Slide 61 of 193 · Reveal 2

The Tortoise and the Hare



UCS

Cost so far

62

Speaker notes

Recall the tortoise and the hare. The slow, steady tortoise is our analogy for UCS: careful about the cost already paid, but without a sense of direction toward the goal.

The Tortoise and the Hare

Slide 62 of 193 · Initial

The Tortoise and the Hare



UCS

Cost so far



Greedy

Estimated cost remaining

62

Speaker notes

The hare represents greedy search: it follows what looks promising, but can rush into an expensive detour. These are memory aids, not guarantees about relative speed.

The Tortoise and the Hare

Slide 62 of 193 · Reveal 1

The Tortoise and the Hare



UCS

Cost so far



Greedy

Estimated cost remaining

Can we combine their strengths?

62

Speaker notes

Can we keep the tortoise's attention to cost and the hare's guidance toward the goal? Let students suggest what information to combine.

The Tortoise and the Hare

Slide 62 of 193 · Reveal 2

The Tortoise and the Hare



UCS

Cost so far



Greedy

Estimated cost remaining



A*

Use both costs

62

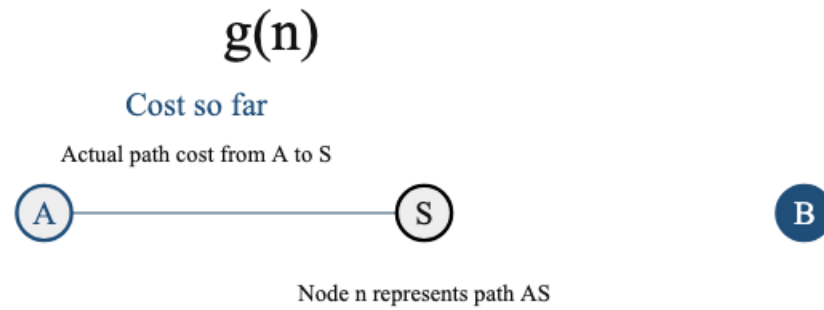
Speaker notes

Reveal the tortoise riding the hare. A* combines the cost so far with the estimated cost remaining. Now give that idea a precise rule.

The Tortoise and the Hare

Slide 62 of 193 · Reveal 3

A* Combines Cost So Far and Cost Remaining



63

Speaker notes

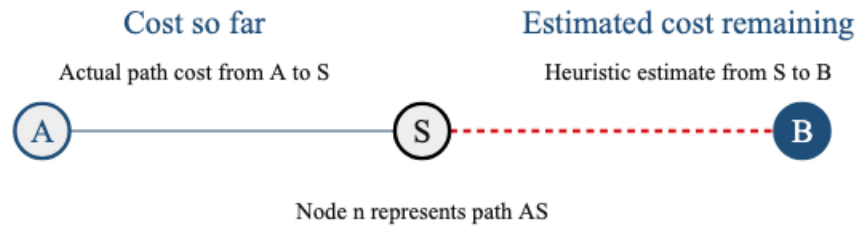
UCS uses $g(n)$, the actual cost of the path so far. Here node n represents path AS, whose final state is S.

A* Combines Cost So Far and Cost Remaining

Slide 63 of 193 · Initial

A* Combines Cost So Far and Cost Remaining

$$g(n) + h(n)$$



63

Speaker notes

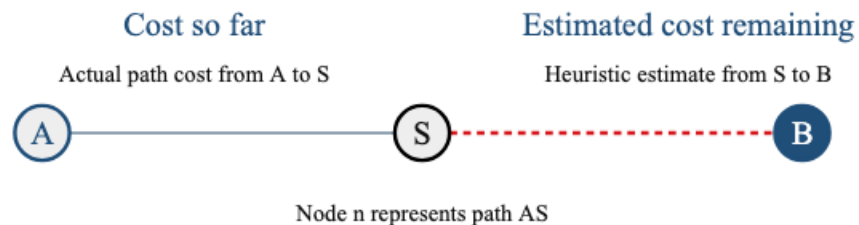
Greedy uses $h(n)$, the estimated remaining cost from S to B. The dashed segment is an estimate, not a road.

A* Combines Cost So Far and Cost Remaining

Slide 63 of 193 · Reveal 1

A* Combines Cost So Far and Cost Remaining

$$g(n) + h(n) = f(n)$$



Select the frontier node with the smallest $f(n)$

63

Speaker notes

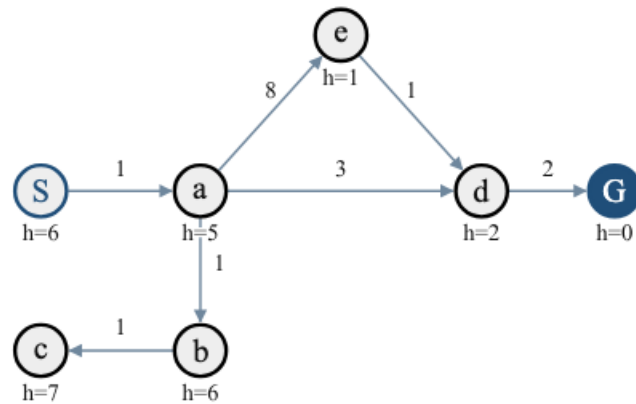
A* adds the two quantities. Its priority $f(n)$ estimates the total cost of a solution through this node. Remove the smallest f -value from the priority queue.

A* Combines Cost So Far and Cost Remaining

Slide 63 of 193 · Reveal 2

A* Motivation: Combine Both Costs

Now revisit the same choices using $g + h$



After expanding a

$$S_{ab}: 2 + 6 = 8$$

$$S_{ad}: 4 + 2 = 6$$

$$S_{ae}: 9 + 1 = 10$$

64

Speaker notes

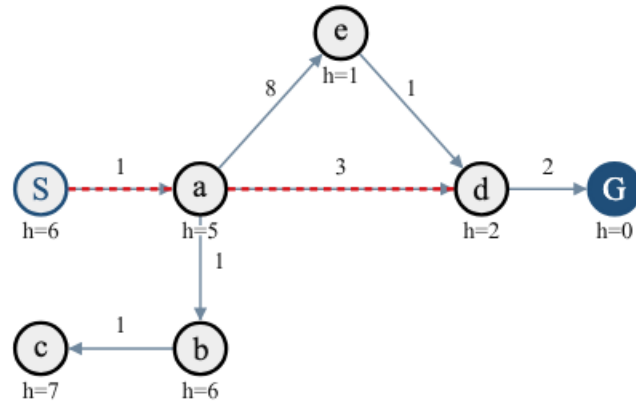
Now apply the combined priority to the same frontier. Sad wins because it balances cost already paid and estimated cost remaining.

A* Motivation: Combine Both Costs

Slide 64 of 193 · Initial

A* Motivation: Combine Both Costs

Now revisit the same choices using $g + h$



After expanding a

$$\text{Sab: } 2 + 6 = 8$$

$$\text{Sad: } 4 + 2 = 6$$

$$\text{Sae: } 9 + 1 = 10$$

Select Sad

Its estimated total cost is smallest

$$\text{SadG: } 6 + 0 = 6$$

64

A* Motivation: Combine Both Costs

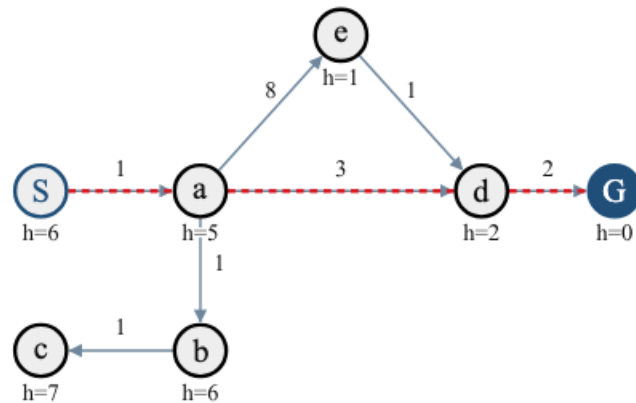
Slide 64 of 193 · Reveal 1

Speaker notes

Expand d and generate SadG with $f=6$. Sab still has $f=8$ and Sae has $f=10$.

A* Motivation: Combine Both Costs

Now revisit the same choices using $g + h$



After expanding a

Sab: $2 + 6 = 8$

Sad: $4 + 2 = 6$

Sae: $9 + 1 = 10$

Select Sad

Its estimated total cost is smallest

SadG: $6 + 0 = 6$

Reach G without either detour

Expanded: $S \rightarrow a \rightarrow d$

Returned path: SadG, cost 6

64

A* Motivation: Combine Both Costs

Slide 64 of 193 · Reveal 2

Speaker notes

Remove SadG and stop. In this example A* avoids the b-c dead end and the detour to e. We will examine later which heuristic conditions guarantee optimality.

A* Uses Both Costs

Select the frontier node with the smallest $g(n) + h(n)$



65

A* Uses Both Costs

Slide 65 of 193 · Initial

Speaker notes

Revisit the mining analogy after the formula and shared worked example. A* accounts for both the cost so far and the estimated remaining cost. The illustration is a memory aid; optimality still depends on the heuristic and algorithm.

main.py

```

1 from heapq import heappush, heappop
2 def aStarTsa(graph, h, start, goal):
3     frontier = [(h[start], start)]
4     print("Initial frontier:", frontier)
5     while frontier:
6         f, path = heappop(frontier)
7         state = path[-1]
8         if state == goal:
9             return f, path
10        print("Exploring:", state, "at priority", f)
11        for cost, child in graph[state]:
12            new_f = f + cost - h[state] + h[child]
13            heappush(frontier, (new_f, path + child))
14        print(frontier)
15 graph = {'S': [(1, 'a')], 'a': [(1, 'b'), (3, 'd'), (8, 'e')],
16         'b': [(1, 'c')], 'c': [], 'd': [(2, 'G')],
17         'e': [(1, 'd')], 'G': []}
18 h = {'S': 6, 'a': 5, 'b': 6, 'c': 7, 'd': 2, 'e': 1, 'G': 0}
19 print("Solution path:", aStarTsa(graph, h, 'S', 'G'))

```

Download: a_star_tsa.py

66

Speaker notes

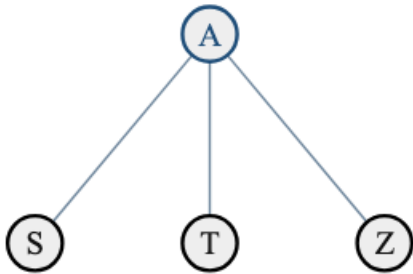
Run the original motivation example in Python. The heap stores (f, path), as last year. Subtract the current heuristic to recover g, add the edge cost, then add the child heuristic. The output says priority rather than cost because f is not generally the actual path cost.

A*-TSA Code

Slide 66 of 193 · Initial

A* Selects the Smallest $g + h$

Same frontier paths · a new priority



Frontier after expanding A

g : path cost · h : estimate at final state

Path	g	h	$f = g + h$
AS	140	253	
AT	118	329	
AZ	75	374	

67

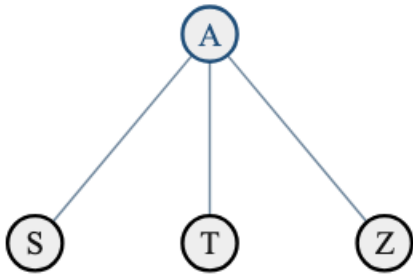
Speaker notes

Keep the same paths AS, AT and AZ. The g column records each path's cost; h is the estimate at its final state.

A* Selects the Smallest $g + h$

A* Selects the Smallest $g + h$

Same frontier paths · a new priority



Frontier after expanding A

g : path cost · h : estimate at final state

Path	g	h	$f = g + h$
AS	140	253	393
AT	118	329	447
AZ	75	374	449

67

Speaker notes

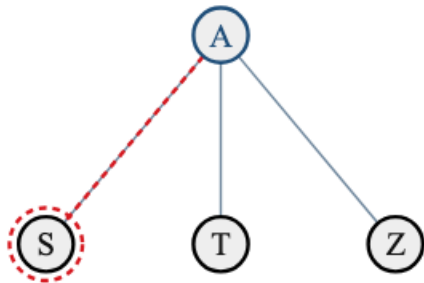
Add across each row. AS has 140 plus 253, giving 393. AT gives 447, and AZ gives 449.

A* Selects the Smallest $g + h$

Slide 67 of 193 · Reveal 1

A* Selects the Smallest $g + h$

Same frontier paths · a new priority



Frontier after expanding A

g : path cost · h : estimate at final state

Path	g	h	$f = g + h$
AS	140	253	393
AT	118	329	447
AZ	75	374	449

Remove AS: 393 is smallest

67

Speaker notes

A* removes AS. Greedy also chose AS at this first step, but UCS chose AZ. The priority rules differ even when two methods happen to make the same choice.

A* Selects the Smallest $g + h$

Slide 67 of 193 · Reveal 2

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447

Tree shows the competing routes
Other generated branches are omitted

68

Speaker notes

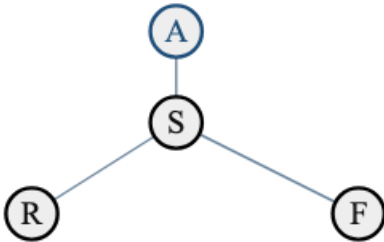
Continue from the first choice: AS has f=393. The table shows the two smallest frontier priorities after each expansion, not the entire heap.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Initial

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447
AS	393	ASR: 413 · ASF: 415

Tree shows the competing routes
Other generated branches are omitted

68

Speaker notes

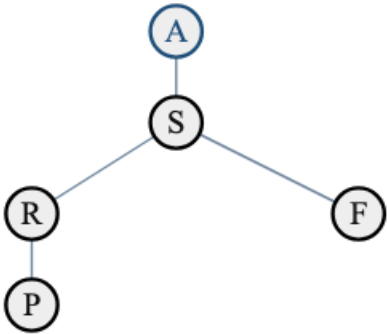
Expand S: ASR at 413 beats ASF at 415, so explore R next.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Reveal 1

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447
AS	393	ASR: 413 · ASF: 415
ASR	413	ASF: 415 · ASRP: 417

Tree shows the competing routes
Other generated branches are omitted

68

Speaker notes

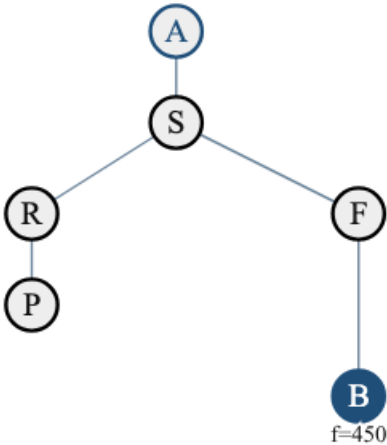
Expanding R generates ASRP at 417. ASF is still waiting at 415, so A* switches to that branch next.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Reveal 2

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Tree shows the competing routes
Other generated branches are omitted

Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447
AS	393	ASR: 413 · ASF: 415
ASR	413	ASF: 415 · ASRP: 417
ASF	415	ASRP: 417 · AT: 447

68

Speaker notes

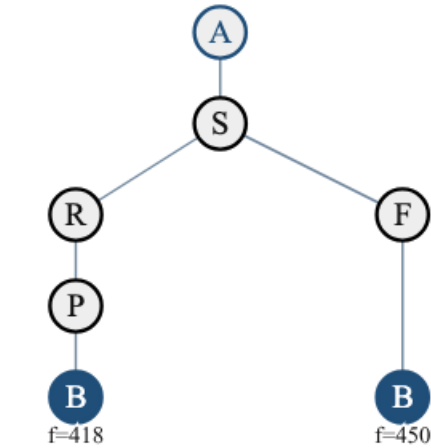
Expanding F generates ASFB at 450. Finding a goal does not stop the search: ASRP at 417 is still better.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Reveal 3

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Tree shows the competing routes
Other generated branches are omitted

Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447
AS	393	ASR: 413 · ASF: 415
ASR	413	ASF: 415 · ASRP: 417
ASF	415	ASRP: 417 · AT: 447
ASRP	417	ASRPB: 418 · AT: 447
<hr/>		
ASRPB: g=418, h(B)=0, f=418		

68

Speaker notes

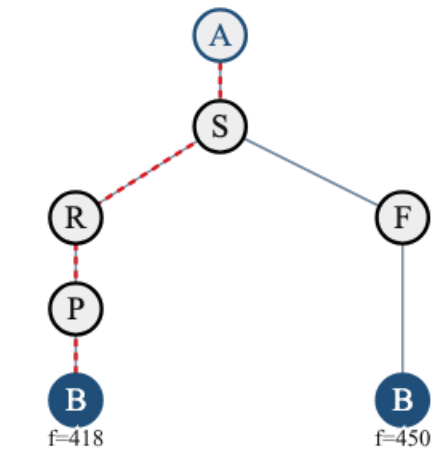
Expand P and generate ASRPB at 418. It now beats AT at 447 and the earlier goal ASFB at 450.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Reveal 4

A* Keeps Comparing Frontier Paths

Which path leaves the frontier next?



Tree shows the competing routes
Other generated branches are omitted

Expanded path	f	Two lowest frontier priorities
A	366	AS: 393 · AT: 447
AS	393	ASR: 413 · ASF: 415
ASR	413	ASF: 415 · ASRP: 417
ASF	415	ASRP: 417 · AT: 447
ASRP	417	ASRPB: 418 · AT: 447
<hr/>		
ASRPB: g=418, h(B)=0, f=418		
Remove ASRPB and stop		
Solution cost: 418		

68

Speaker notes

Remove ASRPB and pass the goal test. Expanded states were A, S, R, F, P; B is tested before expansion.

A* Keeps Comparing Frontier Paths

Slide 68 of 193 · Reveal 5

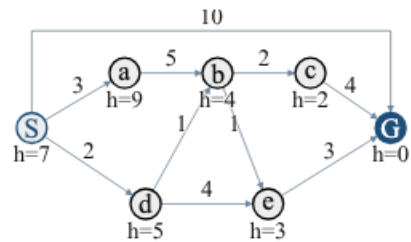
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



69

Speaker notes

Use the same graph as the earlier practice tasks. Ask students to track path cost separately from the priority. Reveal the output after students predict each step. The frontier is displayed in priority order.

A*-TSA Practice {`.comp3270-source-slide` `.source-slide-heading` `.comp3270-native-slide` `data-source-slide="astar-practice"`}

Slide 69 of 193 · Initial

A*-TSA Practice

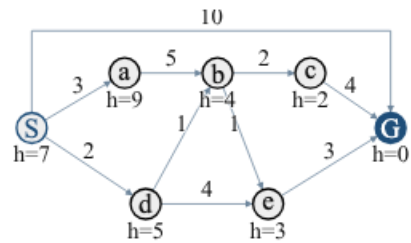
Start: S · Goal: G

Initial frontier: [(7, 'S')]

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



69

Speaker notes

No speaker notes for this stage.

A*-TSA Practice {`.comp3270-source-slide` `.source-slide-heading` `.comp3270-native-slide` `data-source-slide="astar-practice"`}

Slide 69 of 193 · Reveal 1

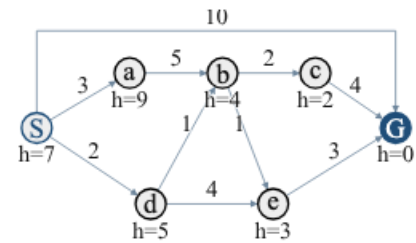
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



69

A*-TSA Practice {`.comp3270-source-slide` `.source-slide-heading` `.comp3270-native-slide` `data-source-slide="astar-practice"`}

Slide 69 of 193 · Reveal 2

Speaker notes

No speaker notes for this stage.

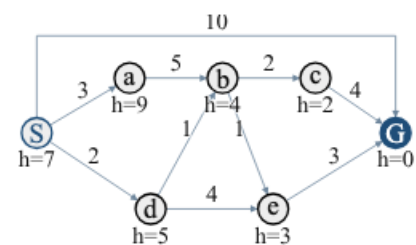
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



69

A*-TSA Practice {`.comp3270-source-slide` `.source-slide-heading` `.comp3270-native-slide` `data-source-slide="astar-practice"`}

Slide 69 of 193 · Reveal 3

Speaker notes

No speaker notes for this stage.

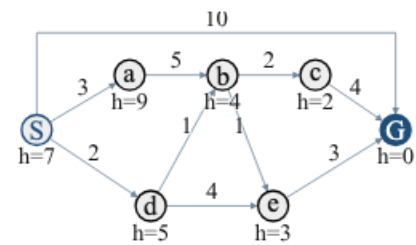
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



Initial frontier: [(7, 'S')]

S @ 7 → [(7, 'Sd'),
(10, 'SG'), (12, 'Sa')]

Sd @ 7 → [(7, 'Sdb'), (9, 'Sde'),
(10, 'SG'), (12, 'Sa')]

Sdb @ 7 → [(7, 'Sdbc'), (7, 'Sdbe'), (9, 'Sde'),
(10, 'SG'), (12, 'Sa')]

69

A*-TSA Practice {`.comp3270-source-slide` `.source-slide-heading` `.comp3270-native-slide` `data-source-slide="astar-practice"`}

Slide 69 of 193 · Reveal 4

Speaker notes

No speaker notes for this stage.

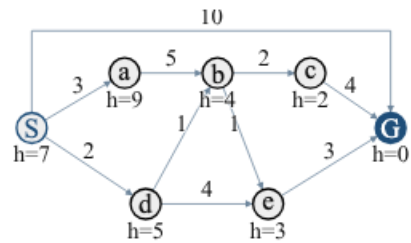
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



Initial frontier: [(7, 'S')]

S @ 7 → [(7, 'Sd'),
(10, 'SG'), (12, 'Sa')]

Sd @ 7 → [(7, 'Sdb'), (9, 'Sde'),
(10, 'SG'), (12, 'Sa')]

Sdb @ 7 → [(7, 'Sdbc'), (7, 'Sdbe'), (9, 'Sde'),
(10, 'SG'), (12, 'Sa')]

Sdbc @ 7 → [(7, 'Sdbe'), (9, 'SdbcG'), (9, 'Sde'),
(10, 'SG'), (12, 'Sa')]

69

Speaker notes

Sdbc wins the $f=7$ tie over Sdbe by path-string order. Priorities are f , not g .

A*-TSA Practice {[.comp3270-source-slide](#) [.source-slide-heading](#) [.comp3270-native-slide](#) [data-source-slide="astar-practice"](#)}

Slide 69 of 193 · Reveal 5

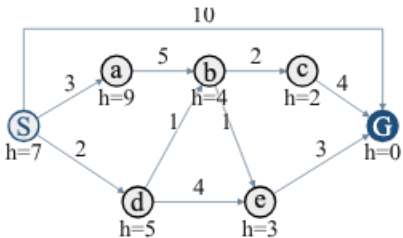
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



Initial frontier: [(7, 'S')]

S @ 7 → [(7, 'Sd'), (10, 'SG'), (12, 'Sa')]

Sd @ 7 → [(7, 'Sdb'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdb @ 7 → [(7, 'Sdbc'), (7, 'Sdbe'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdbc @ 7 → [(7, 'Sdbe'), (9, 'SdbcG'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdbe @ 7 → [(7, 'SdbeG'), (9, 'SdbcG'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

69

A*-TSA Practice {[.comp3270-source-slide](#) [.source-slide-heading](#) [.comp3270-native-slide](#) [data-source-slide="astar-practice"](#)}

Slide 69 of 193 · Reveal 6

Speaker notes

No speaker notes for this stage.

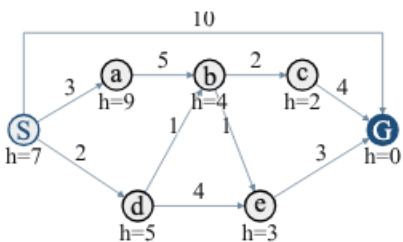
A*-TSA Practice

Start: S · Goal: G

Use $f(n)=g(n)+h(n)$

Predict the expansion order
and returned path

Break ties by path-string order



Initial frontier: [(7, 'S')]

S @ 7 → [(7, 'Sd'), (10, 'SG'), (12, 'Sa')]

Sd @ 7 → [(7, 'Sdb'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdb @ 7 → [(7, 'Sdbc'), (7, 'Sdbe'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdbc @ 7 → [(7, 'Sdbe'), (9, 'SdbcG'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

Sdbe @ 7 → [(7, 'SdbeG'), (9, 'SdbcG'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]

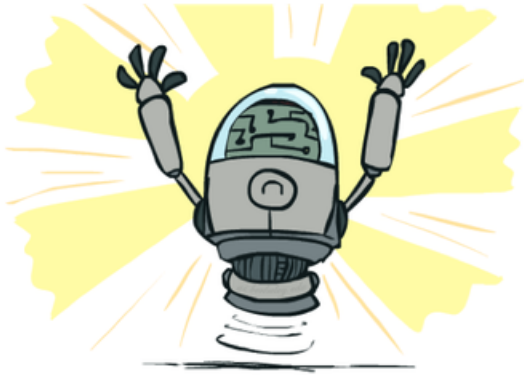
Solution path: (7, 'SdbeG')

69

Speaker notes

Expanded: S, d, b, c, e. The goal test returns SdbeG at cost 7 before expanding G.

Is A* Always Optimal?



It worked on Romania. Is that a guarantee?

70

Is A* Always Optimal?

Slide 70 of 193 · Initial

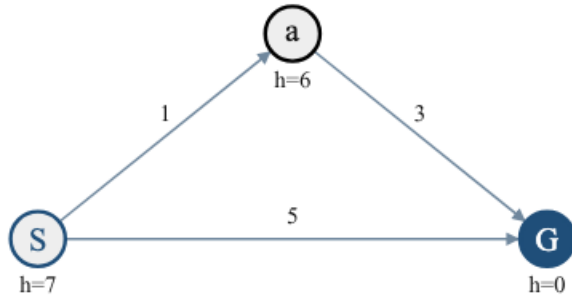
Speaker notes

Use the original celebratory robot as a transition. Ask whether our successful examples prove that A* always returns the cheapest path.

Is A*-TSA Optimal?

Start: S · Goal: G

Which path will A*-TSA return?



71

Speaker notes

Ask students to predict the returned path before revealing the output.

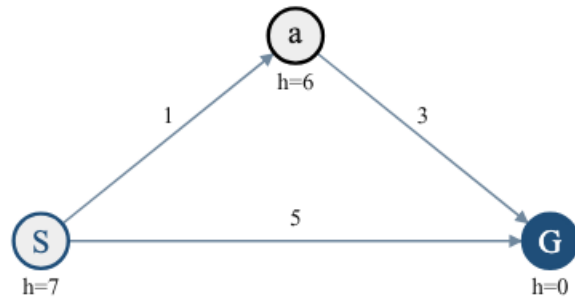
Is A*-TSA Optimal? {.comp3270-source-slide .source-slide-heading .comp3270-native-slide data-source-slide="astar-overestimate-question"}

Slide 71 of 193 · Initial

Is A*-TSA Optimal?

Start: S · Goal: G

Which path will A*-TSA return?



Initial frontier: [(7, 'S')]
 S @ 7 → [(5, 'SG'), (7, 'Sa')]

71

Speaker notes

Tuples contain f and path. SG has priority 5; Sa has priority 7.

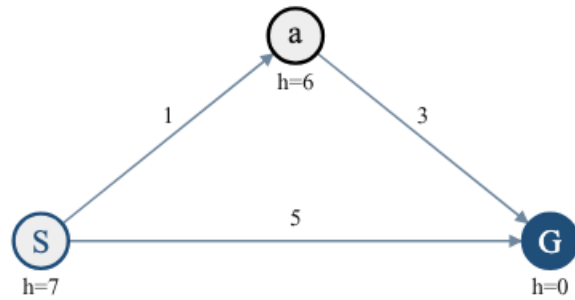
Is A*-TSA Optimal? {`.comp3270-source-slide .source-slide-heading .comp3270-native-slide data-source-slide="astar-overestimate-question"`}

Slide 71 of 193 · Reveal 1

Is A*-TSA Optimal?

Start: S · Goal: G

Which path will A*-TSA return?



Initial frontier: [(7, 'S')]
S @ 7 → [(5, 'SG'), (7, 'Sa')]

Solution path: (5, 'SG')

71

Speaker notes

SG passes the goal test on removal. A*-TSA returns cost 5 without exploring a.

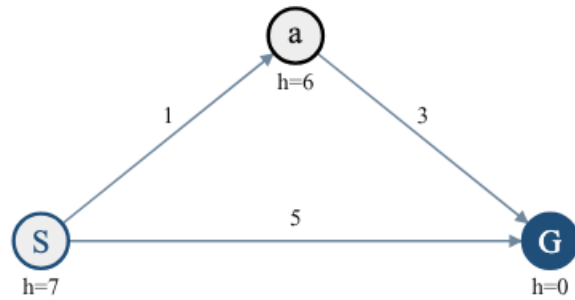
Is A*-TSA Optimal? {.comp3270-source-slide .source-slide-heading .comp3270-native-slide data-source-slide="astar-overestimate-question"}

Slide 71 of 193 · Reveal 2

Is A*-TSA Optimal?

Start: S · Goal: G

Which path will A*-TSA return?



Initial frontier: [(7, 'S')]
 S @ 7 → [(5, 'SG'), (7, 'Sa')]

Solution path: (5, 'SG')

Returned: SG, cost 5

Cheaper path: SaG, cost 1 + 3 = 4

71

Speaker notes

SaG costs only 4. The algorithm returns a solution, but it is not optimal.

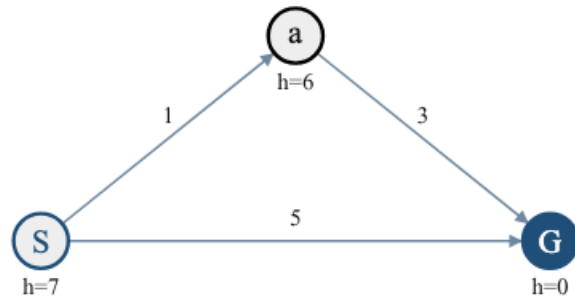
Is A*-TSA Optimal? {`.comp3270-source-slide .source-slide-heading .comp3270-native-slide data-source-slide="astar-overestimate-question"`}

Slide 71 of 193 · Reveal 3

Is A*-TSA Optimal?

Start: S · Goal: G

Which path will A*-TSA return?



Initial frontier: [(7, 'S')]
 S @ 7 → [(5, 'SG'), (7, 'Sa')]

Solution path: (5, 'SG')

Returned: SG, cost 5

Cheaper path: SaG, cost 1 + 3 = 4

$h(a) = 6$ overestimates the
 true remaining cost of 3

71

Speaker notes

The inflated estimate hides the cheaper route. This motivates admissibility; $h(S)=7$ also exceeds its true remaining cost of 4.

Is A*-TSA Optimal? {.comp3270-source-slide .source-slide-heading .comp3270-native-slide data-source-slide="astar-overestimate-question"}

Slide 71 of 193 · Reveal 4

Admissibility of Heuristic

An optimistic estimate

$$0 \leq h(n) \leq h^*(n)$$

72

Admissibility of Heuristic

Slide 72 of 193 · Initial

Speaker notes

Define admissibility only after the failure example. h^* is the true remaining cost, not the heuristic we can conveniently compute.

Admissibility of Heuristic

An optimistic estimate

$$0 \leq h(n) \leq h^*(n)$$

$h^*(n)$ is the true cheapest remaining cost

$h(n)$ may underestimate, but must not overestimate

72

Admissibility of Heuristic

Slide 72 of 193 · Reveal 1

Speaker notes

An underestimate can lead to extra search, but it does not inflate a promising path above its true completion cost.

Admissibility of Heuristic

An optimistic estimate

$$0 \leq h(n) \leq h^*(n)$$

$h^*(n)$ is the true cheapest remaining cost

$h(n)$ may underestimate, but must not overestimate

At a goal

$$h(\text{goal})=0$$

72

Speaker notes

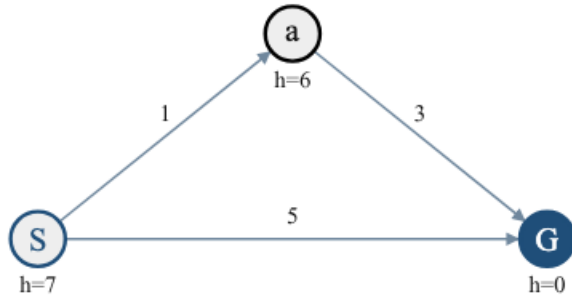
At a goal the remaining cost is zero, so the admissible heuristic must also be zero.

Admissibility of Heuristic

Slide 72 of 193 · Reveal 2

Admissibility of Heuristic

Is h admissible for this graph?



73

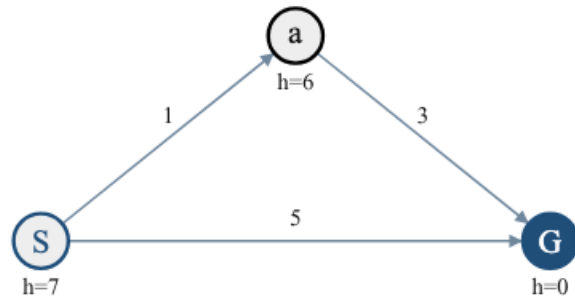
Admissibility of Heuristic

Slide 73 of 193 · Initial

Speaker notes

Ask whether h is admissible. Let students check the remaining costs before revealing either comparison.

Admissibility of Heuristic



Is h admissible for this graph?

$$h(a) = 6 > h^*(a) = 3$$

73

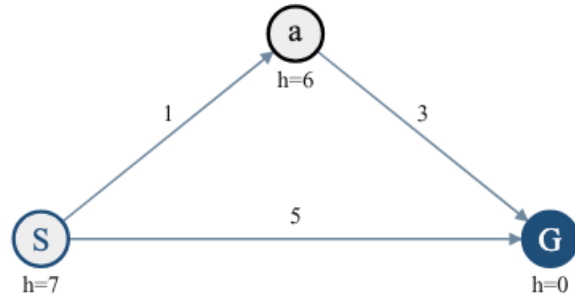
Speaker notes

At a , the remaining cost is 3, so 6 overestimates. Ask students to check S as well.

Admissibility of Heuristic

Slide 73 of 193 · Reveal 1

Admissibility of Heuristic



Is h admissible for this graph?

$$h(a) = 6 > h^*(a) = 3$$

$$h(S) = 7 > h^*(S) = 4$$

73

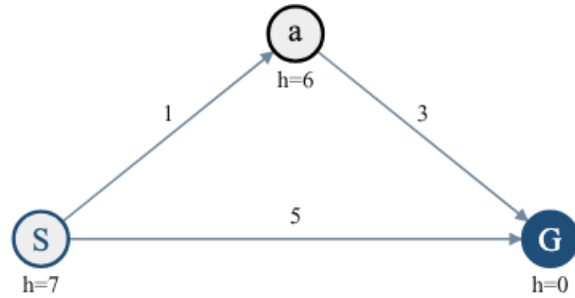
Speaker notes

From S, the cheapest route costs 4, so 7 also overestimates. The heuristic is not admissible.

Admissibility of Heuristic

Slide 73 of 193 · Reveal 2

Admissibility of Heuristic



Is h admissible for this graph?

$$h(a) = 6 > h^*(a) = 3$$

$$h(S) = 7 > h^*(S) = 4$$

Keep the same edge costs

What values are allowed for $h(a)$?

What values are allowed for $h(S)$?

73

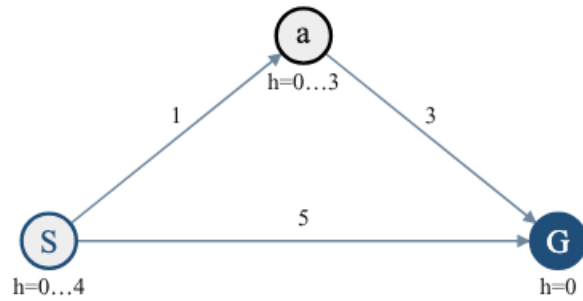
Speaker notes

Now ask for the full allowed ranges, keeping the edge costs fixed. Leave their answers for the next slide.

Admissibility of Heuristic

Slide 73 of 193 · Reveal 3

Admissibility: Check Every State



From a

$$h^*(a)=3$$

$$\text{Therefore } 0 \leq h(a) \leq 3$$

74

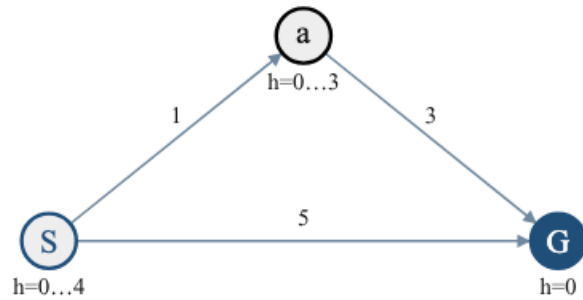
Speaker notes

The only route from a costs 3.

Admissibility: Check Every State

Slide 74 of 193 · Initial

Admissibility: Check Every State



From a

$$h^*(a) = 3$$

$$\text{Therefore } 0 \leq h(a) \leq 3$$

From S

$$h^*(S) = \min(5, 1+3) = 4$$

$$\text{Therefore } 0 \leq h(S) \leq 4$$

74

Speaker notes

The cheapest route from S goes through a and costs 4. Changing only $h(a)$ would leave the original $h(S)=7$ inadmissible.

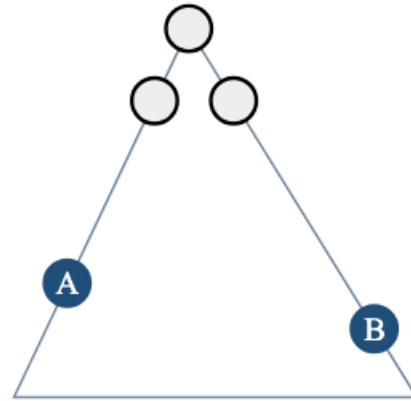
Admissibility: Check Every State

Slide 74 of 193 · Reveal 1

Optimality of A*

Let

- A be an optimal goal node
- B be a suboptimal goal node
- h be admissible



75

Optimality of A*

Slide 75 of 193 · Initial

Speaker notes

Restore the original search-space triangle. A and B are goal nodes in this proof, not Romania states. We are considering A*-TSA.

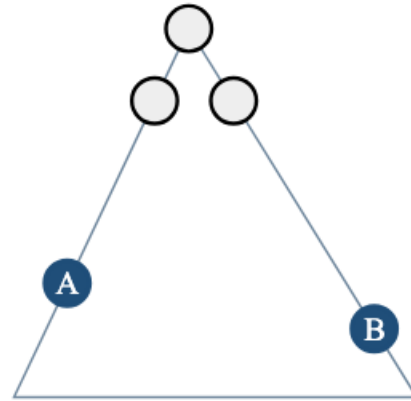
Optimality of A*

Let

- A be an optimal goal node
- B be a suboptimal goal node
- h be admissible

Claim

- A will exit the frontier before B



75

Speaker notes

A cheaper goal must leave the frontier before a more expensive goal. The next slide compares a frontier ancestor with B.

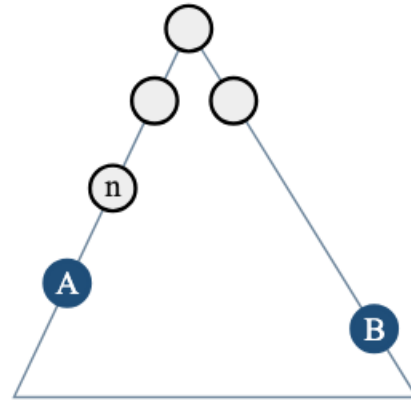
Optimality of A*

Slide 75 of 193 · Reveal 1

Optimality of A*

Let n be an ancestor of A

Let B and n both be on the frontier



76

Speaker notes

Keep the original ancestor argument and diagram.

Optimality of A*

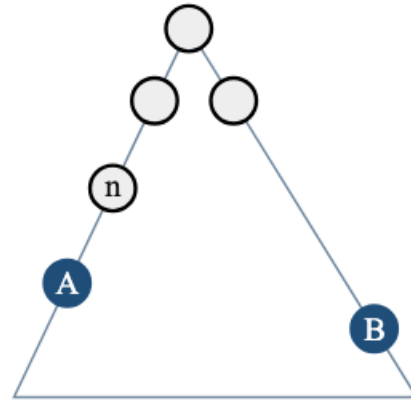
Slide 76 of 193 · Initial

Optimality of A*

Let n be an ancestor of A

Let B and n both be on the frontier

Claim: n exits the frontier before B



76

Speaker notes

Ask how the frontier priorities compare.

Optimality of A*

Slide 76 of 193 · Reveal 1

Optimality of A*

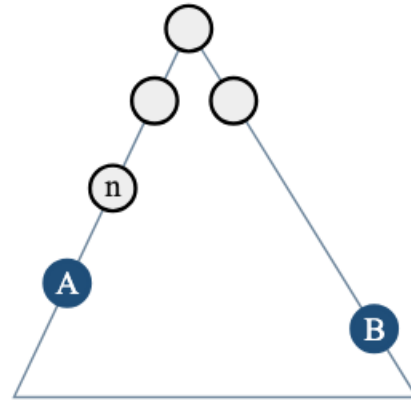
Let n be an ancestor of A

Let B and n both be on the frontier

Claim: n exits the frontier before B

$$f(n) \leq f(A)$$

Admissibility, with n on the optimal route to A



76

Speaker notes

$f(n)=g(n)+h(n)$ is at most the true cost of the optimal completion through A , which is $f(A)$.

Optimality of A*

Slide 76 of 193 · Reveal 2

Optimality of A*

Let n be an ancestor of A

Let B and n both be on the frontier

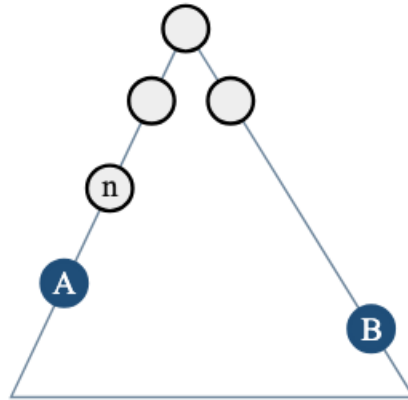
Claim: n exits the frontier before B

$$f(n) \leq f(A)$$

Admissibility, with n on the optimal route to A

$$f(A) < f(B)$$

B is suboptimal and both goal heuristics are zero



76

Speaker notes

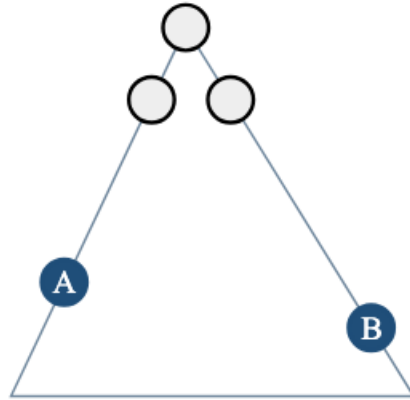
Together these inequalities give $f(n)$ less than $f(B)$, so n must be removed first.

Optimality of A*

Slide 76 of 193 · Reveal 3

Optimality of A*

All ancestors of A exit the frontier before B



77

Optimality of A*

Slide 77 of 193 · Initial

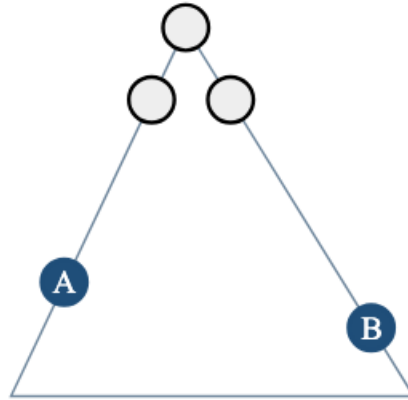
Speaker notes

Repeat the ancestor argument along the optimal route.

Optimality of A*

All ancestors of A exit the frontier before B

A exits the frontier before B



77

Speaker notes

The same reasoning eventually reaches the optimal goal. Assume the usual termination conditions.

Optimality of A*

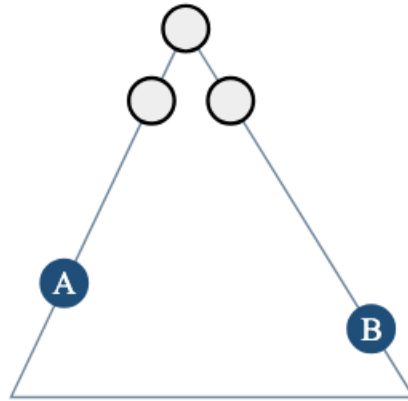
Slide 77 of 193 · Reveal 1

Optimality of A*

All ancestors of A exit the frontier before B

A exits the frontier before B

A*-TSA is optimal with admissible h



77

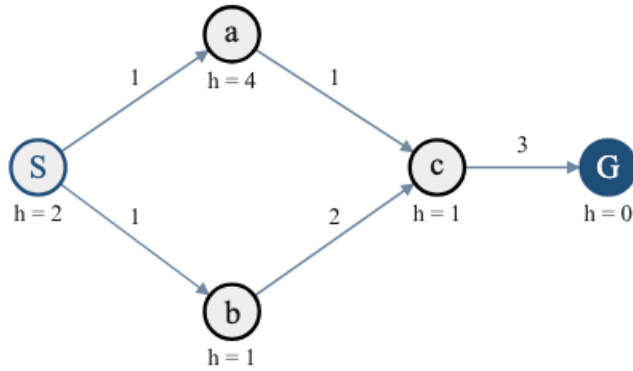
Speaker notes

Preserve the original conclusion explicitly scoped to tree search. The next example tests whether the explored set changes the result.

Optimality of A*

Slide 77 of 193 · Reveal 2

Is This Heuristic Admissible?



78

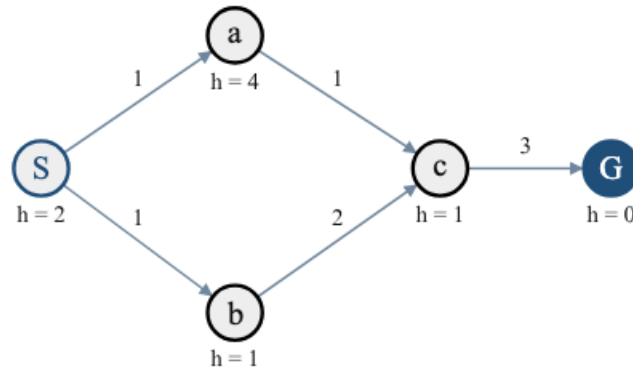
Is This Heuristic Admissible?

Slide 78 of 193 · Initial

Speaker notes

Ask students whether this heuristic is admissible before revealing the checks.

Is This Heuristic Admissible?



True remaining costs

$h^*(G) = 0, h^*(c) = 3$

$h^*(a) = 4, h^*(b) = 5$

$h^*(S) = 5$

78

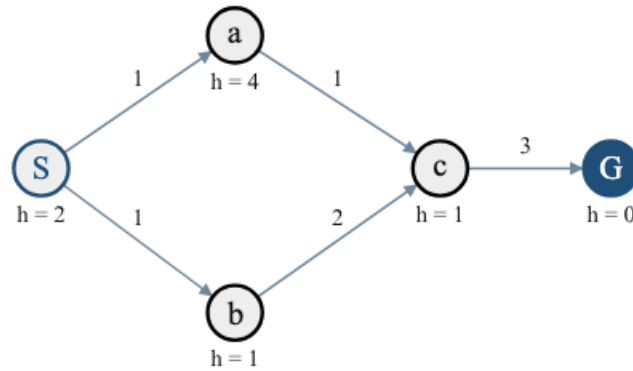
Is This Heuristic Admissible?

Slide 78 of 193 · Reveal 1

Speaker notes

Work backward from G to obtain the true remaining costs.

Is This Heuristic Admissible?



True remaining costs

$$h^*(G) = 0, h^*(c) = 3$$

$$h^*(a) = 4, h^*(b) = 5$$

$$h^*(S) = 5$$

Compare with h

$$0 \leq 0 \quad \cdot \quad 1 \leq 3 \quad \cdot \quad 4 \leq 4$$

$$1 \leq 5 \quad \cdot \quad 2 \leq 5$$

78

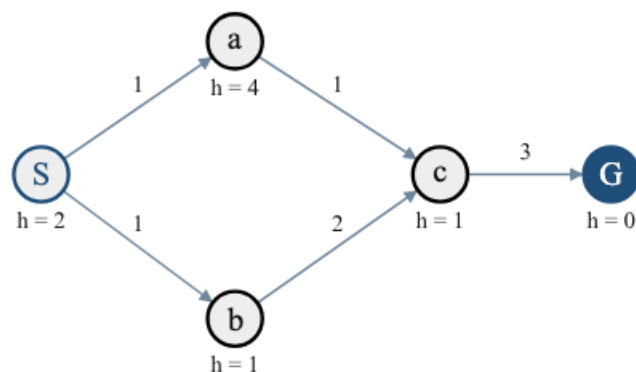
Is This Heuristic Admissible?

Slide 78 of 193 · Reveal 2

Speaker notes

Each displayed heuristic is a lower bound.

Is This Heuristic Admissible?



True remaining costs

$$h^*(G) = 0, h^*(c) = 3$$

$$h^*(a) = 4, h^*(b) = 5$$

$$h^*(S) = 5$$

Compare with h

$$0 \leq 0 \quad \cdot \quad 1 \leq 3 \quad \cdot \quad 4 \leq 4$$

$$1 \leq 5 \quad \cdot \quad 2 \leq 5$$

All estimates pass

Yes, this heuristic is admissible

78

Speaker notes

Confirm admissibility before tracing TSA. Each heuristic is at most the true remaining cost.

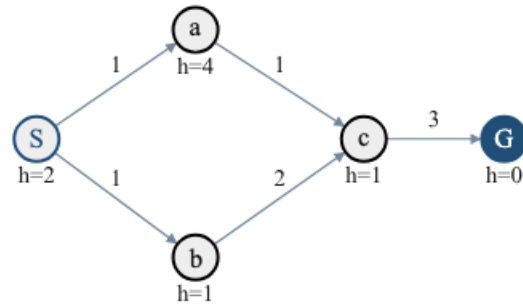
Is This Heuristic Admissible?

Slide 78 of 193 · Reveal 3

Optimality of A*: TSA

Start: S · Goal: G

Initial frontier: [(2, 'S')]



79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading comp3270-native-slide data-source-slide="astar-tsa-success"`}

Slide 79 of 193 · Initial

Speaker notes

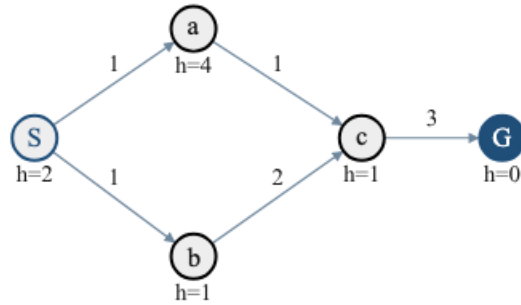
Ask students to predict the TSA run. Track f separately from path cost.

Optimality of A*: TSA

Start: S · Goal: G

Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]



79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading`
`comp3270-native-slide data-source-slide="astar-tsa-success"`}

Slide 79 of 193 · Reveal 1

Speaker notes

No speaker notes for this stage.

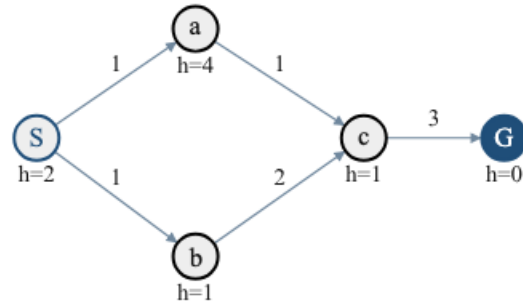
Optimality of A*: TSA

Start: S · Goal: G

Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]

Sb @ 2 → [(4, 'Sbc'), (5, 'Sa')]



79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading comp3270-native-slide data-source-slide="astar-tsa-success"`}

Slide 79 of 193 · Reveal 2

Speaker notes

No speaker notes for this stage.

Optimality of A*: TSA

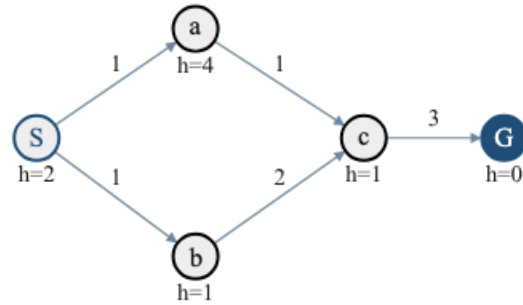
Start: S · Goal: G

Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]

Sb @ 2 → [(4, 'Sbc'), (5, 'Sa')]

Sbc @ 4 → [(5, 'Sa'), (6, 'SbcG')]



79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading`
`comp3270-native-slide data-source-slide="astar-tsa-success"`}

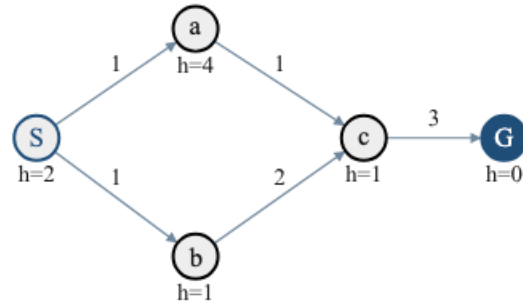
Slide 79 of 193 · Reveal 3

Speaker notes

No speaker notes for this stage.

Optimality of A*: TSA

Start: S · Goal: G



Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]

Sb @ 2 → [(4, 'Sbc'), (5, 'Sa')]

Sbc @ 4 → [(5, 'Sa'), (6, 'SbcG')]

Sa @ 5 → [(3, 'Sac'), (6, 'SbcG')]

79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading`
`comp3270-native-slide data-source-slide="astar-tsa-success"`}

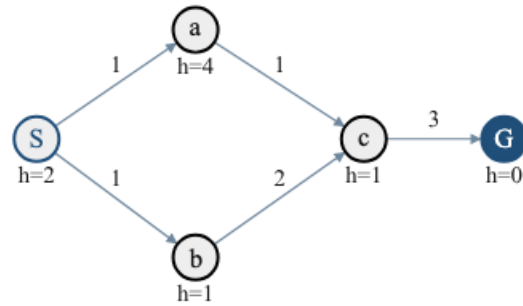
Slide 79 of 193 · Reveal 4

Speaker notes

No speaker notes for this stage.

Optimality of A*: TSA

Start: S · Goal: G



Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]

Sb @ 2 → [(4, 'Sbc'), (5, 'Sa')]

Sbc @ 4 → [(5, 'Sa'), (6, 'SbcG')]

Sa @ 5 → [(3, 'Sac'), (6, 'SbcG')]

Sac @ 3 → [(5, 'SacG'), (6, 'SbcG')]

79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading comp3270-native-slide data-source-slide="astar-tsa-success"`}

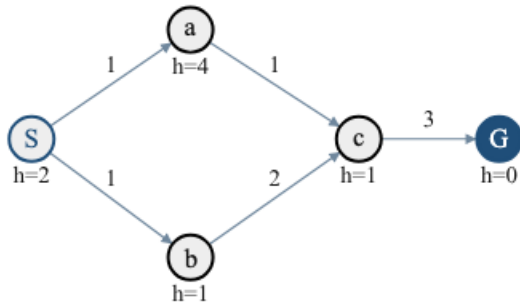
Slide 79 of 193 · Reveal 5

Speaker notes

No speaker notes for this stage.

Optimality of A*: TSA

Start: S · Goal: G



```
Initial frontier: [(2, 'S')]
S @ 2 -> [(2, 'Sb'), (5, 'Sa')]
Sb @ 2 -> [(4, 'Sbc'), (5, 'Sa')]
Sbc @ 4 -> [(5, 'Sa'), (6, 'SbcG')]
Sa @ 5 -> [(3, 'Sac'), (6, 'SbcG')]
Sac @ 3 -> [(5, 'SacG'), (6, 'SbcG')]
Solution path: (5, 'SacG')
```

79

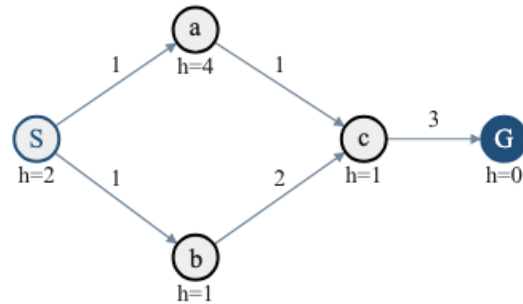
Speaker notes

No speaker notes for this stage.

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading comp3270-native-slide data-source-slide="astar-tsa-success"`}

Optimality of A*: TSA

Start: S · Goal: G



Initial frontier: [(2, 'S')]

S @ 2 → [(2, 'Sb'), (5, 'Sa')]

Sb @ 2 → [(4, 'Sbc'), (5, 'Sa')]

Sbc @ 4 → [(5, 'Sa'), (6, 'SbcG')]

Sa @ 5 → [(3, 'Sac'), (6, 'SbcG')]

Sac @ 3 → [(5, 'SacG'), (6, 'SbcG')]

Solution path: (5, 'SacG')

Optimal path: SacG, cost 5

79

Optimality of A*: TSA {`.comp3270-source-slide .source-slide-heading`
`comp3270-native-slide data-source-slide="astar-tsa-success"`}

Slide 79 of 193 · Reveal 7

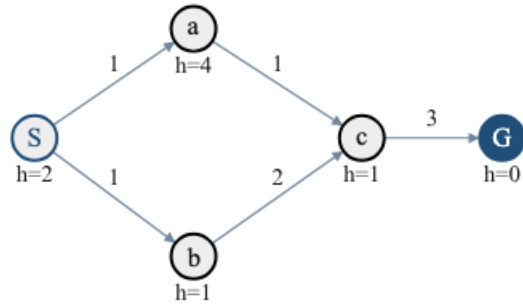
Speaker notes

TSA expands c again via the cheaper Sac path and returns SacG at cost 5.
 Frontier tuples are (f, path), displayed in priority order.

Optimality of A*: GSA

Our GSA never expands a state twice

Initial frontier: [(2, 'S')]
Explored: $\emptyset$



80

Optimality of A*: GSA

Slide 80 of 193 · Initial

Speaker notes

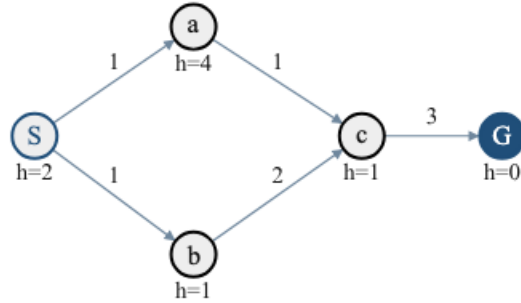
Trace the same graph using our explored-set rule.

Optimality of A*: GSA

Our GSA never expands a state twice

Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}



80

Optimality of A*: GSA

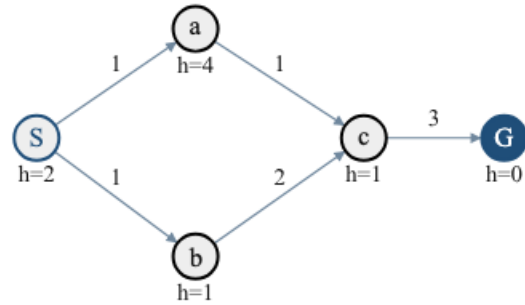
Slide 80 of 193 · Reveal 1

Speaker notes

No speaker notes for this stage.

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

80

Speaker notes

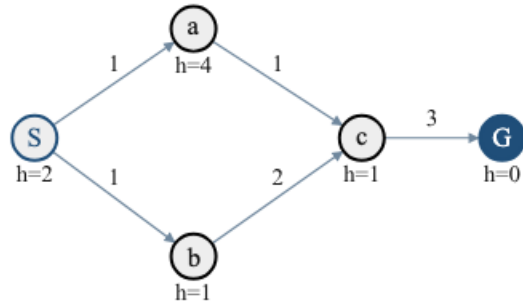
No speaker notes for this stage.

Optimality of A*: GSA

Slide 80 of 193 · Reveal 2

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

Sbc @ 4 $\rightarrow$ [(5, 'Sa'), (6, 'SbcG')]
Explored: {S, b, c}

80

Speaker notes

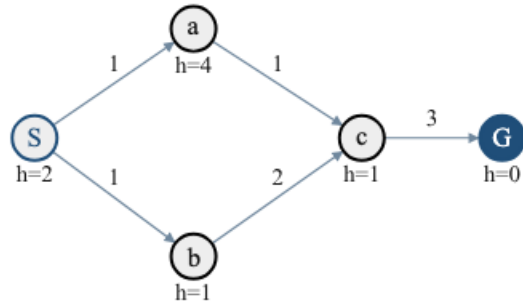
No speaker notes for this stage.

Optimality of A*: GSA

Slide 80 of 193 · Reveal 3

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

Sbc @ 4 $\rightarrow$ [(5, 'Sa'), (6, 'SbcG')]
Explored: {S, b, c}

Sa @ 5 $\rightarrow$ [(3, 'Sac'), (6, 'SbcG')]
Explored: {S, b, c, a}

80

Speaker notes

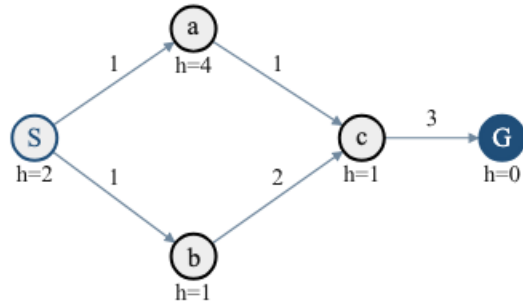
No speaker notes for this stage.

Optimality of A*: GSA

Slide 80 of 193 · Reveal 4

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

Sbc @ 4 $\rightarrow$ [(5, 'Sa'), (6, 'SbcG')]
Explored: {S, b, c}

Sa @ 5 $\rightarrow$ [(3, 'Sac'), (6, 'SbcG')]
Explored: {S, b, c, a}

Skip Sac: c already explored

80

Speaker notes

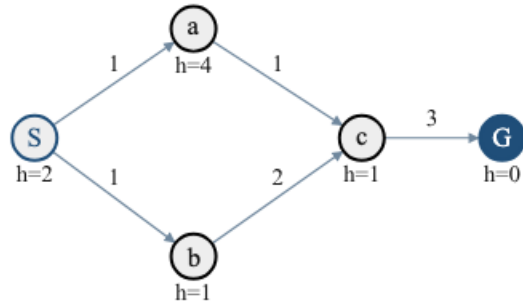
No speaker notes for this stage.

Optimality of A*: GSA

Slide 80 of 193 · Reveal 5

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

Sbc @ 4 $\rightarrow$ [(5, 'Sa'), (6, 'SbcG')]
Explored: {S, b, c}

Sa @ 5 $\rightarrow$ [(3, 'Sac'), (6, 'SbcG')]
Explored: {S, b, c, a}

Skip Sac: c already explored

Solution path: (6, 'SbcG')

80

Optimality of A*: GSA

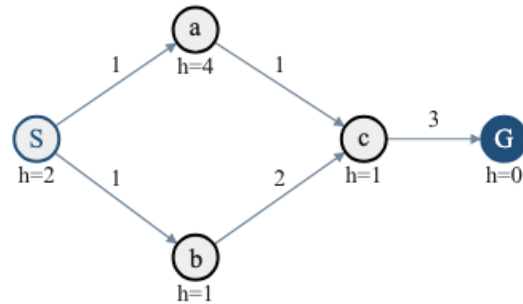
Slide 80 of 193 · Reveal 6

Speaker notes

No speaker notes for this stage.

Optimality of A*: GSA

Our GSA never expands a state twice



Initial frontier: [(2, 'S')]
Explored: $\emptyset$

S @ 2 $\rightarrow$ [(2, 'Sb'), (5, 'Sa')]
Explored: {S}

Sb @ 2 $\rightarrow$ [(4, 'Sbc'), (5, 'Sa')]
Explored: {S, b}

Sbc @ 4 $\rightarrow$ [(5, 'Sa'), (6, 'SbcG')]
Explored: {S, b, c}

Sa @ 5 $\rightarrow$ [(3, 'Sac'), (6, 'SbcG')]
Explored: {S, b, c, a}

Skip Sac: c already explored

Solution path: (6, 'SbcG')

What went wrong?

80

Speaker notes

GSA skips Sac because c was already explored, so it returns SbcG at cost 6. Ask students to explain the failure before proceeding.

Optimality of A*: GSA

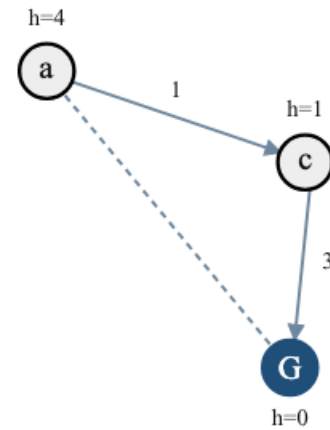
Slide 80 of 193 · Reveal 7

Consistency of Heuristic

Definition

For every arc $a \rightarrow c$

$$h(a) - h(c) \leq \text{cost}(a,c)$$



81

Speaker notes

Restore the original a – c – G triangle. Solid sides are actual edges of costs 1 and 3. The dashed a – G side represents the heuristic comparison, not an additional action.

Consistency of Heuristic

Slide 81 of 193 · Initial

Consistency of Heuristic

Definition

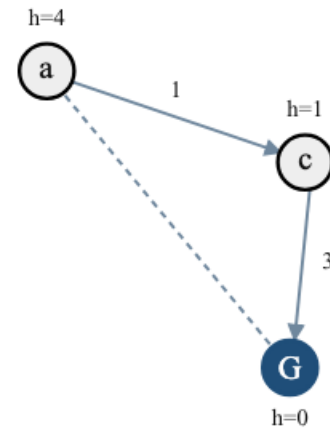
For every arc $a \rightarrow c$

$$h(a) - h(c) \leq \text{cost}(a,c)$$

Equivalently

$$h(a) \leq \text{cost}(a,c) + h(c)$$

Here: $4 \leq 1 + 1$ is false



81

Speaker notes

The estimate at a cannot exceed the actual first step plus the estimate at c.
The displayed heuristic violates this triangle inequality even though it is admissible.

Consistency of Heuristic

Slide 81 of 193 · Reveal 1

Consistency of Heuristic

Definition

For every arc $a \rightarrow c$

$$h(a) - h(c) \leq \text{cost}(a,c)$$

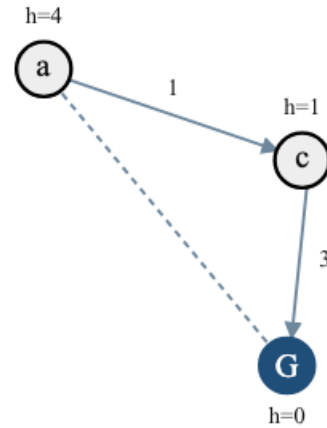
Equivalently

$$h(a) \leq \text{cost}(a,c) + h(c)$$

Here: $4 \leq 1 + 1$ is false

Consequence of consistency

The f value along a path never decreases



81

Speaker notes

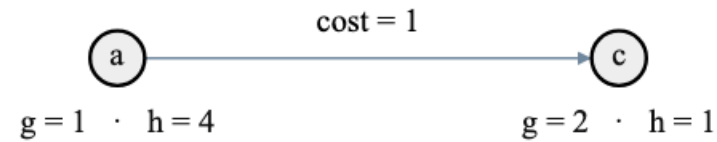
Ask the original follow-up: can you prove A*-GSA is optimal if f never decreases? The next reveal slide works through the relation.

Consistency of Heuristic

Slide 81 of 193 · Reveal 2

Consistency Keeps f from Decreasing

Recall the cheaper path $S \rightarrow a \rightarrow c$



82

Speaker notes

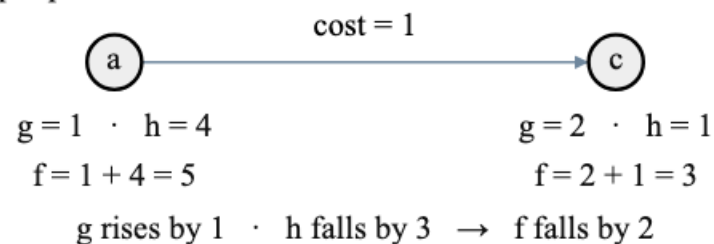
Recall the cheaper Sac path from the last run. Ask how g and h change across a to c , and what happens to f .

Consistency Keeps f from Decreasing

Slide 82 of 193 · Initial

Consistency Keeps f from Decreasing

Recall the cheaper path $S \rightarrow a \rightarrow c$



82

Speaker notes

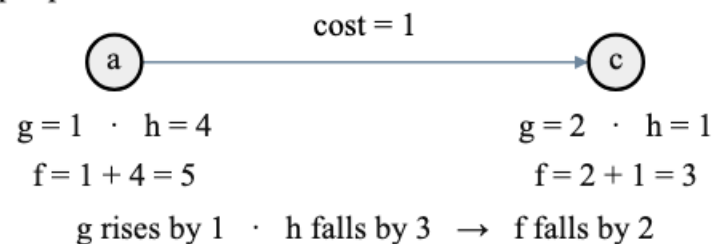
The heuristic drops by 3 although the step costs only 1. This makes the priority fall from 5 to 3; GSA had already closed c via the more expensive route.

Consistency Keeps f from Decreasing

Slide 82 of 193 · Reveal 1

Consistency Keeps f from Decreasing

Recall the cheaper path $S \rightarrow a \rightarrow c$



Consistency: h cannot fall by more than the step costs

$$f(v) - f(u) = \text{cost}(u, v) - (h(u) - h(v)) \geq 0$$

82

Speaker notes

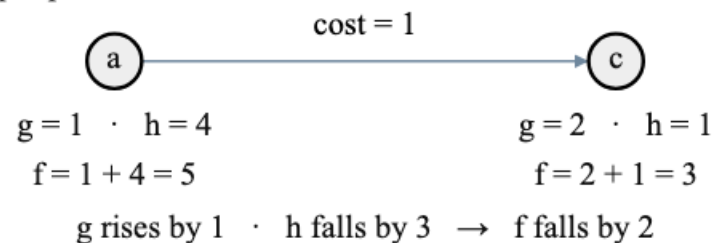
For any consecutive path nodes u and v , g increases by the step cost.
Consistency limits the drop in h to that cost, so their sum f cannot decrease.

Consistency Keeps f from Decreasing

Slide 82 of 193 · Reveal 2

Consistency Keeps f from Decreasing

Recall the cheaper path $S \rightarrow a \rightarrow c$



Consistency: h cannot fall by more than the step costs

$$f(v) - f(u) = \text{cost}(u, v) - (h(u) - h(v)) \geq 0$$

With consistent h , GSA first expands each state via a cheapest path
Closing the state cannot block a cheaper route later

82

Speaker notes

If a cheaper route existed, consistency would give its frontier prefix a lower f than the expensive arrival at the same state. That prefix must be processed first. This is why closing states is safe with consistent h .

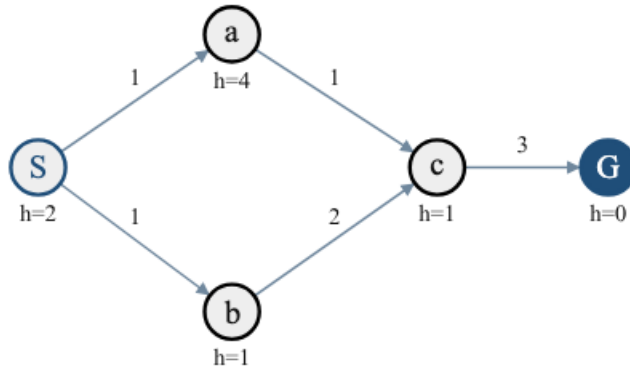
Consistency Keeps f from Decreasing

Slide 82 of 193 · Reveal 3

Consistency of Heuristic

Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$



83

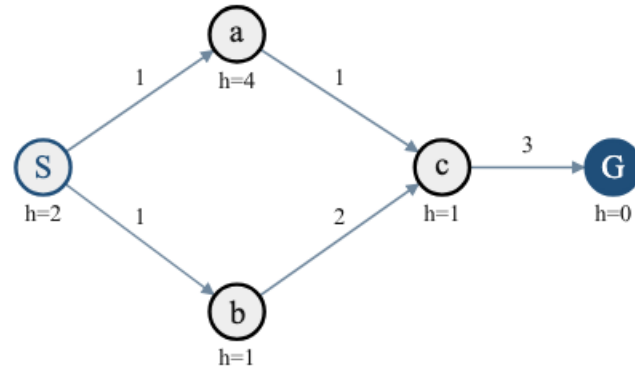
Speaker notes

Ask whether h is consistent. Let students check the directed edges before revealing the calculations.

Consistency of Heuristic

Slide 83 of 193 · Initial

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

83

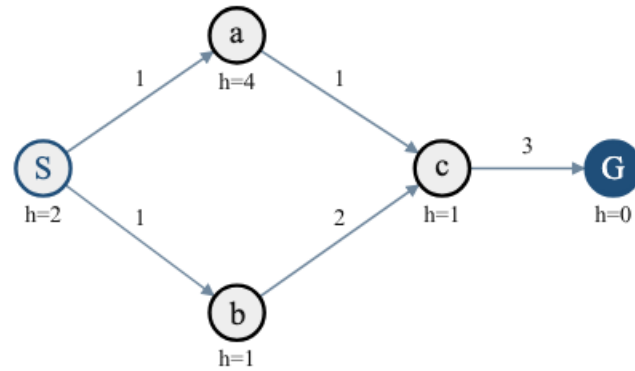
Consistency of Heuristic

Slide 83 of 193 · Reveal 1

Speaker notes

S to a passes: 2 is at most 5.

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

83

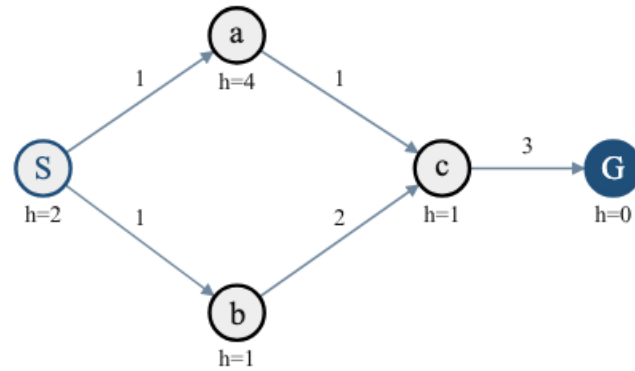
Consistency of Heuristic

Slide 83 of 193 · Reveal 2

Speaker notes

S to b passes with equality.

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

$$a \rightarrow c: 4 > 1 + 1$$

83

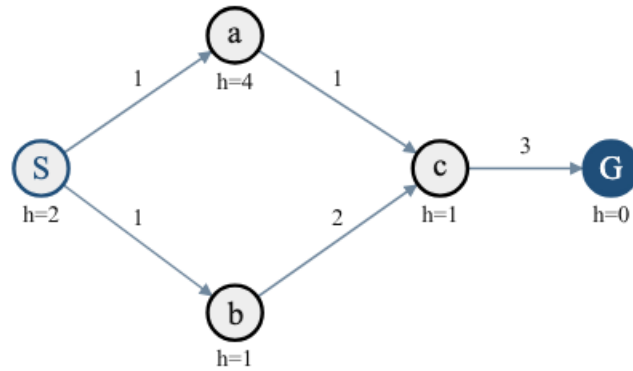
Speaker notes

a to c fails: the estimate drops by 3, but the edge costs only 1. Continue checking the remaining edges.

Consistency of Heuristic

Slide 83 of 193 · Reveal 3

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

$$a \rightarrow c: 4 > 1 + 1$$

$$b \rightarrow c: 1 \leq 2 + 1$$

83

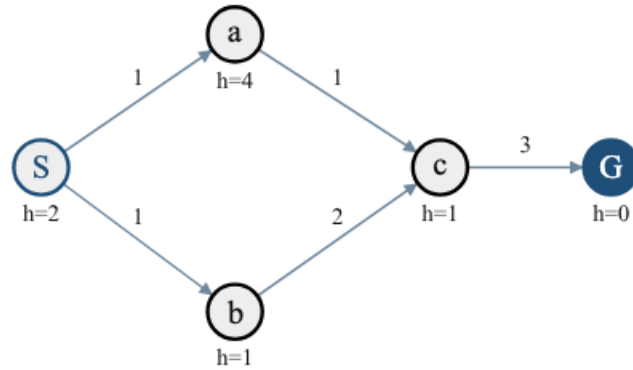
Consistency of Heuristic

Slide 83 of 193 · Reveal 4

Speaker notes

b to c passes: 1 is at most 3.

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

$$a \rightarrow c: 4 > 1 + 1$$

$$b \rightarrow c: 1 \leq 2 + 1$$

$$c \rightarrow G: 1 \leq 3 + 0$$

83

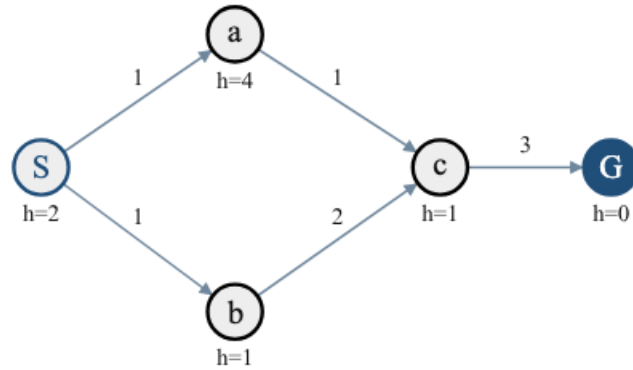
Consistency of Heuristic

Slide 83 of 193 · Reveal 5

Speaker notes

c to G passes: 1 is at most 3.

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

$$a \rightarrow c: 4 > 1 + 1$$

$$b \rightarrow c: 1 \leq 2 + 1$$

$$c \rightarrow G: 1 \leq 3 + 0$$

No: $a \rightarrow c$ violates consistency

83

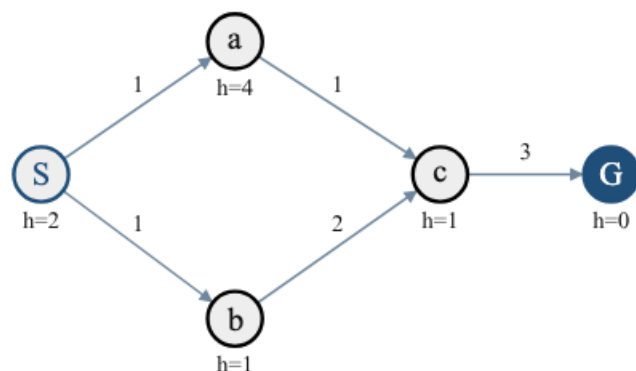
Speaker notes

One failed edge is enough: this heuristic is admissible but inconsistent.

Consistency of Heuristic

Slide 83 of 193 · Reveal 6

Consistency of Heuristic



Is h consistent for this graph?

$$h(u) \leq \text{cost}(u, v) + h(v)$$

$$S \rightarrow a: 2 \leq 1 + 4$$

$$S \rightarrow b: 2 \leq 1 + 1$$

$$a \rightarrow c: 4 > 1 + 1$$

$$b \rightarrow c: 1 \leq 2 + 1$$

$$c \rightarrow G: 1 \leq 3 + 0$$

No: $a \rightarrow c$ violates consistency

Keep all edge costs fixed

How could we repair h ?

83

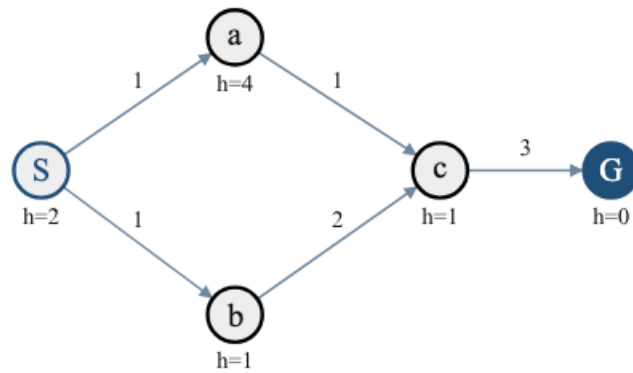
Speaker notes

Ask students to change as few heuristic values as possible. Could changing a alone work? Could changing c alone work? Check incoming and outgoing edges.

Consistency of Heuristic

Slide 83 of 193 · Reveal 7

Two Ways to Repair the Heuristic



Lower $h(a)$ from 4 to 2

$$a \rightarrow c: 2 \leq 1 + 1$$

$$S \rightarrow a: 2 \leq 1 + 2$$

84

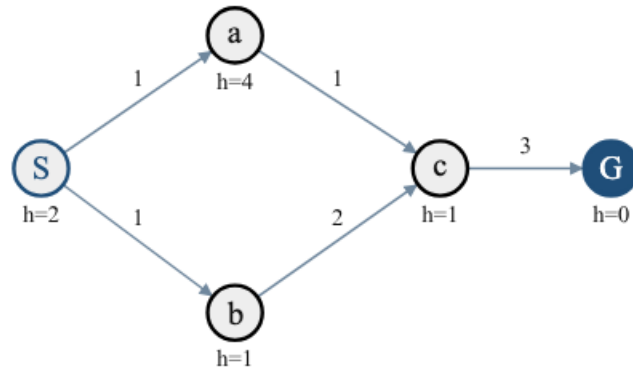
Speaker notes

Lowering $h(a)$ to 2 fixes the excessive drop on a to c without breaking the edge from S .

Two Ways to Repair the Heuristic

Slide 84 of 193 · Initial

Two Ways to Repair the Heuristic



Lower $h(a)$ from 4 to 2

$$a \rightarrow c: 2 \leq 1 + 1$$

$$S \rightarrow a: 2 \leq 1 + 2$$

Or raise $h(c)$ from 1 to 3

$$a \rightarrow c: 4 \leq 1 + 3$$

$$c \rightarrow G: 3 \leq 3 + 0$$

84

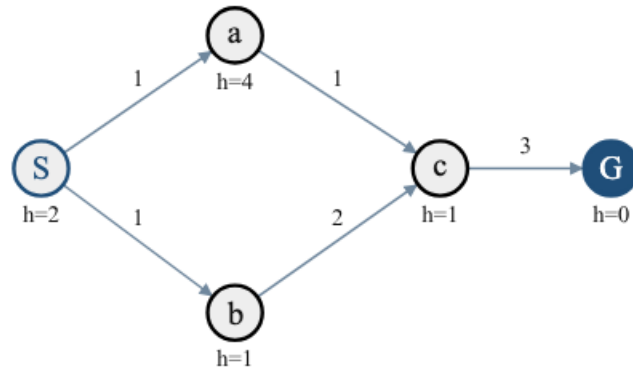
Two Ways to Repair the Heuristic

Slide 84 of 193 · Reveal 1

Speaker notes

Raising $h(c)$ to its true remaining cost also fixes the offending edge. Check b to c as well: 1 is at most $2+3$.

Two Ways to Repair the Heuristic



Lower $h(a)$ from 4 to 2

$a \rightarrow c: 2 \leq 1 + 1$

$S \rightarrow a: 2 \leq 1 + 2$

Or raise $h(c)$ from 1 to 3

$a \rightarrow c: 4 \leq 1 + 3$

$c \rightarrow G: 3 \leq 3 + 0$

Both repairs work

Other edges remain consistent

Both heuristics remain admissible

84

Speaker notes

These are two one-value repairs. Do not suggest there is a unique correct heuristic.

Two Ways to Repair the Heuristic

Slide 84 of 193 · Reveal 2

A*: Key Takeaways

A*-TSA

Returns a cheapest solution when h is admissible

85

A*: Key Takeaways

Slide 85 of 193 · Initial

Speaker notes

These optimality statements assume a solution is returned and use nonnegative h with zero at goals.

A*: Key Takeaways

A*-TSA

Returns a cheapest solution when h is admissible

Our A*-GSA

Returns a cheapest solution when h is consistent

85

A*: Key Takeaways

Slide 85 of 193 · Reveal 1

Speaker notes

A graph-search implementation that reopens states on improved g can instead work with admissibility; distinguish it from our current no-reopening GSA.

A*: Key Takeaways

A*-TSA

Returns a cheapest solution when h is admissible

Our A*-GSA

Returns a cheapest solution when h is consistent

Will it find a solution?

Yes, if one exists, with finite branching
and every step cost $\geq \epsilon > 0$

85

Speaker notes

Finite branching means finitely many successors per state. Every step costs at least the same positive epsilon; costs cannot shrink toward zero. These conditions ensure completeness for the admissible or consistent heuristic cases above.

A*: Key Takeaways

Slide 85 of 193 · Reveal 2

A*: Key Takeaways

A*-TSA

Returns a cheapest solution when h is admissible

Our A*-GSA

Returns a cheapest solution when h is consistent

Will it find a solution?

Yes, if one exists, with finite branching
and every step cost $\geq \epsilon > 0$

Time and memory

Can still grow exponentially

85

Speaker notes

Heuristics guide the search, but do not remove worst-case exponential time and memory requirements.

A*: Key Takeaways

Slide 85 of 193 · Reveal 3

Local Search

86

Local Search

Slide 86 of 193 · Initial

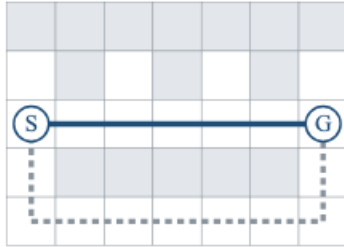
Speaker notes

Local search focuses on the quality of the current state

[Sources] - Preserves previous-offering COMP3270 2024–25
01_search.pptx, slide 85, and its PDF reference

Planning vs. Identification

Planning



The path matters

Find a sequence of actions

Compare path costs and depths

Use heuristics and a frontier

87

Speaker notes

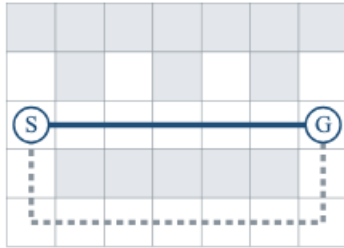
Both routes reach the same goal. Planning compares the paths: six moves versus ten

Planning vs. Identification

Slide 87 of 193 · Initial

Planning vs. Identification

Planning



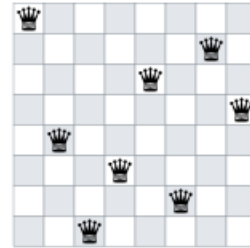
The path matters

Find a sequence of actions

Compare path costs and depths

Use heuristics and a frontier

Identification



The final arrangement matters

Find an assignment to variables

Example: assign a position to each queen

Local search can be faster for some problems

87

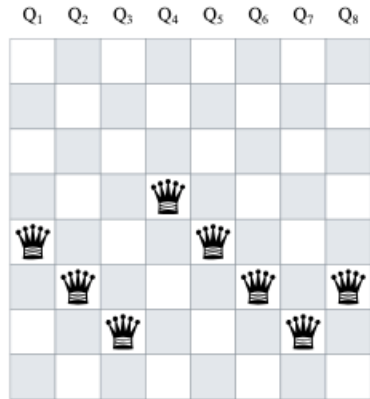
Speaker notes

For eight queens, the variables are queen positions. We want a valid arrangement; how we reached it is not part of the solution

Planning vs. Identification

Slide 87 of 193 · Reveal 1

Example: 8-Queens



State

Eight queens on the board, one per column

88

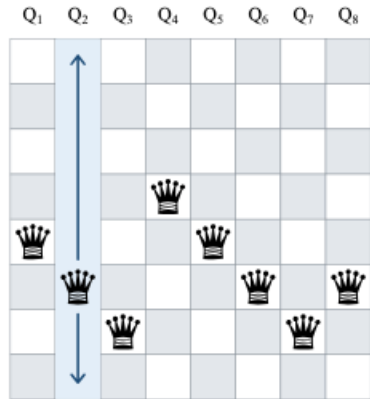
Speaker notes

Each column has one queen; the state records their row positions

Example: 8-Queens

Slide 88 of 193 · Initial

Example: 8-Queens



State

Eight queens on the board, one per column

Successors

Move one queen to another square
in the same column

Every such move gives a neighbour

88

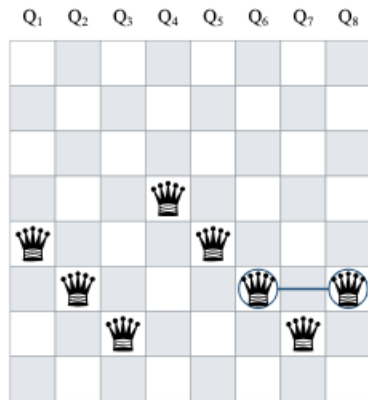
Speaker notes

A successor changes one queen's row within its column

Example: 8-Queens

Slide 88 of 193 · Reveal 1

Example: 8-Queens



Lower cost is better
Goal: zero attacking pairs

State

Eight queens on the board, one per column

Successors

Move one queen to another square
in the same column

Every such move gives a neighbour

Cost function

Count pairs of queens that attack each other

Same row or diagonal

Direct: Q₆ and Q₈

88

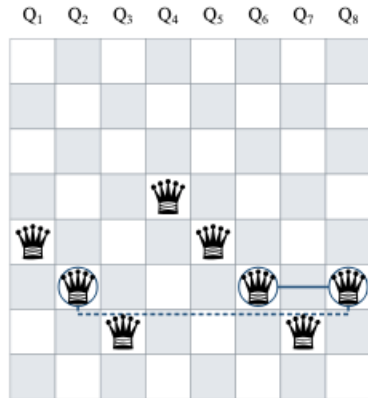
Speaker notes

Q6 and Q8 attack directly, with no queen between them

Example: 8-Queens

Slide 88 of 193 · Reveal 2

Example: 8-Queens



Lower cost is better
Goal: zero attacking pairs

State

Eight queens on the board, one per column

Successors

Move one queen to another square
in the same column

Every such move gives a neighbour

Cost function

Count pairs of queens that attack each other

Same row or diagonal

Direct: Q₆ and Q₈

Indirect: Q₂ and Q₈, with Q₆ between

88

Speaker notes

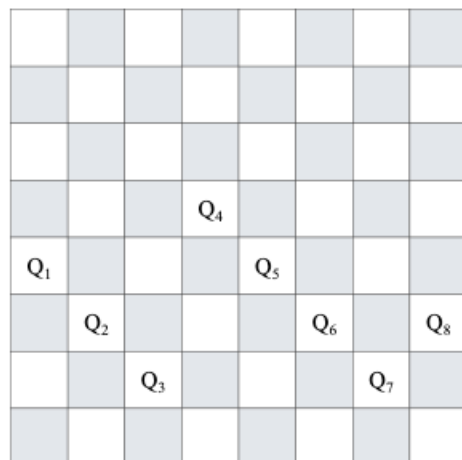
Q₂ and Q₈ are an indirect pair, with Q₆ between. This cost counts that pair too; the next slide asks for the total

Example: 8-Queens

Slide 88 of 193 · Reveal 3

Count Attacking Pairs

What is the cost of this state?



Count each pair once

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

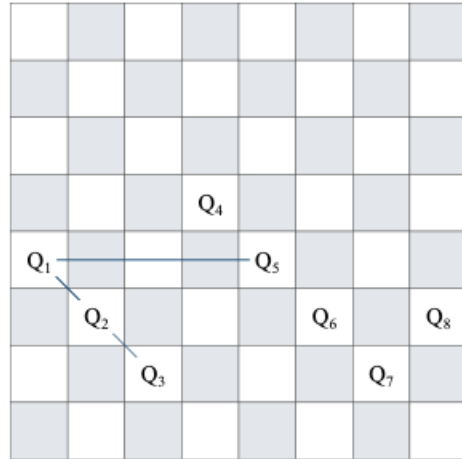
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Initial

Count Attacking Pairs

What is the cost of this state?



Count each pair once

Q₁ with Q₂, Q₃, Q₅

+3

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

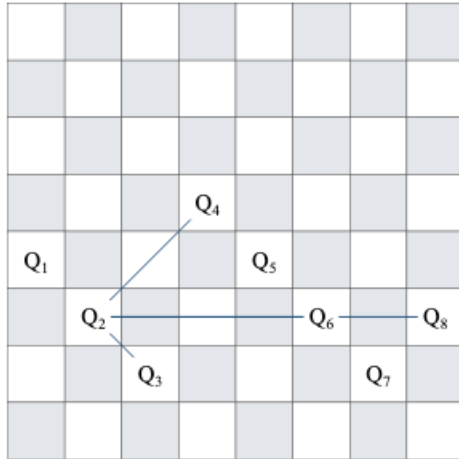
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 1

Count Attacking Pairs

What is the cost of this state?



Count each pair once

Q₁ with Q₂, Q₃, Q₅

+3

Q₂ with Q₃, Q₄, Q₆, Q₈

+4

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

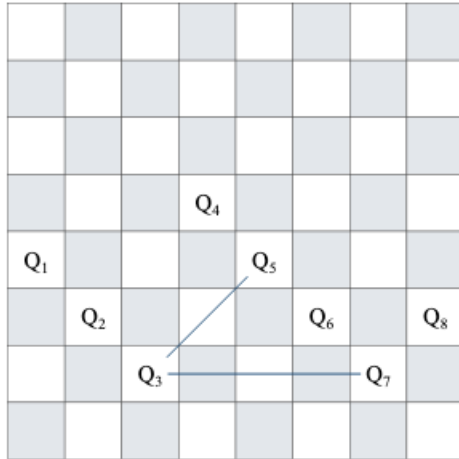
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 2

Count Attacking Pairs

What is the cost of this state?



Count each pair once

Q₁ with Q₂, Q₃, Q₅

+3

Q₂ with Q₃, Q₄, Q₆, Q₈

+4

Q₃ with Q₅, Q₇

+2

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

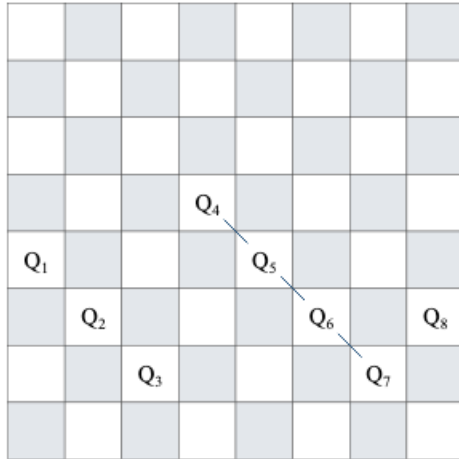
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 3

Count Attacking Pairs

What is the cost of this state?



Count each pair once

Q₁ with Q₂, Q₃, Q₅ +3

Q₂ with Q₃, Q₄, Q₆, Q₈ +4

Q₃ with Q₅, Q₇ +2

Q₄ with Q₅, Q₆, Q₇ +3

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

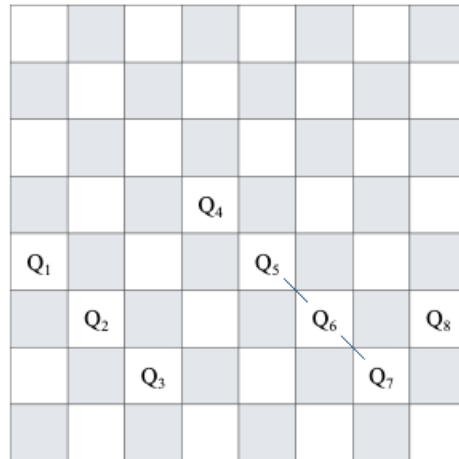
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 4

Count Attacking Pairs

What is the cost of this state?



Count each pair once

Q₁ with Q₂, Q₃, Q₅ +3

Q₂ with Q₃, Q₄, Q₆, Q₈ +4

Q₃ with Q₅, Q₇ +2

Q₄ with Q₅, Q₆, Q₇ +3

Q₅ with Q₆, Q₇ +2

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

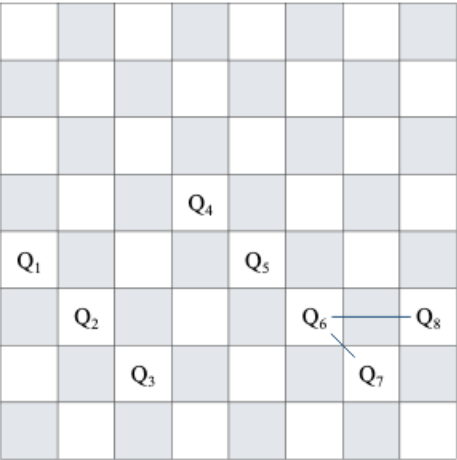
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 5

Count Attacking Pairs

What is the cost of this state?



- Count each pair once
- Q₁ with Q₂, Q₃, Q₅ +3
- Q₂ with Q₃, Q₄, Q₆, Q₈ +4
- Q₃ with Q₅, Q₇ +2
- Q₄ with Q₅, Q₆, Q₇ +3
- Q₅ with Q₆, Q₇ +2
- Q₆ with Q₇, Q₈ +2

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

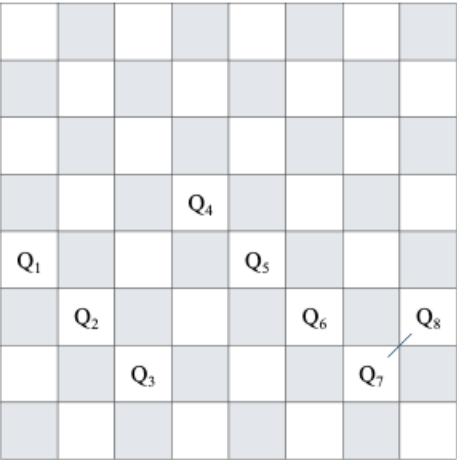
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 6

Count Attacking Pairs

What is the cost of this state?



Count each pair once	
Q ₁ with Q ₂ , Q ₃ , Q ₅	+3
Q ₂ with Q ₃ , Q ₄ , Q ₆ , Q ₈	+4
Q ₃ with Q ₅ , Q ₇	+2
Q ₄ with Q ₅ , Q ₆ , Q ₇	+3
Q ₅ with Q ₆ , Q ₇	+2
Q ₆ with Q ₇ , Q ₈	+2
Q ₇ with Q ₈	+1

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

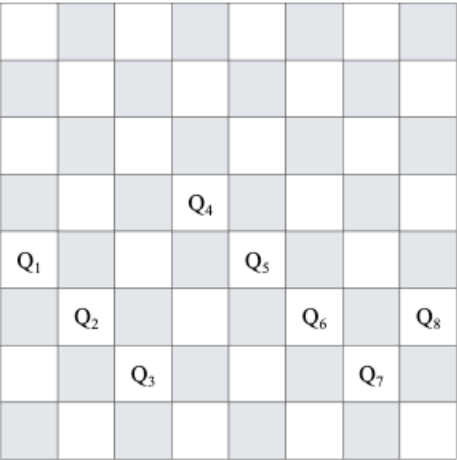
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 7

Count Attacking Pairs

What is the cost of this state?



Count each pair once	
Q ₁ with Q ₂ , Q ₃ , Q ₅	+3
Q ₂ with Q ₃ , Q ₄ , Q ₆ , Q ₈	+4
Q ₃ with Q ₅ , Q ₇	+2
Q ₄ with Q ₅ , Q ₆ , Q ₇	+3
Q ₅ with Q ₆ , Q ₇	+2
Q ₆ with Q ₇ , Q ₈	+2
Q ₇ with Q ₈	+1
$3 + 4 + 2 + 3 + 2 + 2 + 1 = 17$	

89

Speaker notes

Count only pairs with the second queen to the right, so none is counted twice

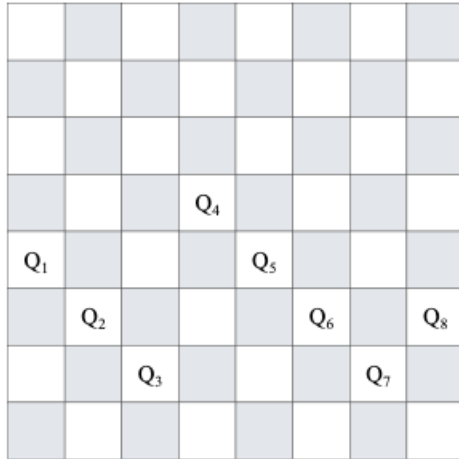
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count Attacking Pairs

Slide 89 of 193 · Reveal 8

Count the Neighbours

How many different one-queen moves are possible?



90

Count the Neighbours

Slide 90 of 193 · Initial

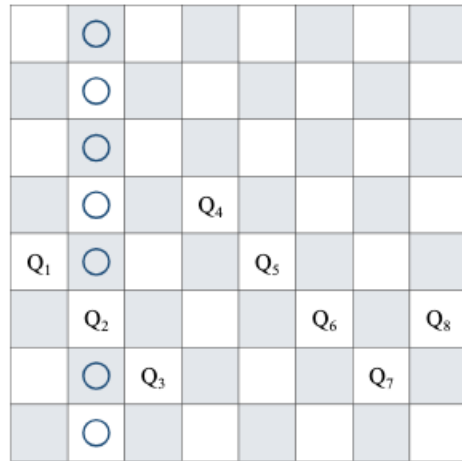
Speaker notes

The current square does not count as a move; each move changes exactly one column

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count the Neighbours

How many different one-queen moves are possible?



One queen

7 other rows

90

Count the Neighbours

Slide 90 of 193 · Reveal 1

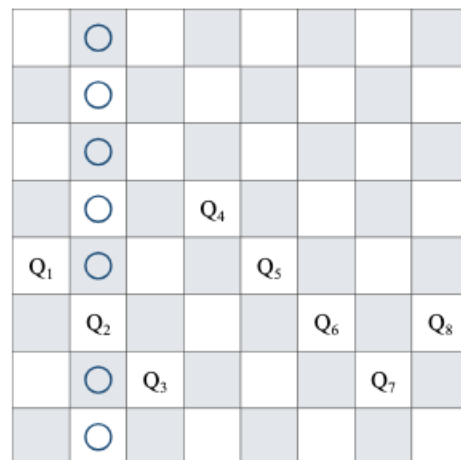
Speaker notes

The current square does not count as a move; each move changes exactly one column

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count the Neighbours

How many different one-queen moves are possible?



One queen

7 other rows

Eight queens

$8 \times 7 = 56$ neighbours

90

Speaker notes

The current square does not count as a move; each move changes exactly one column

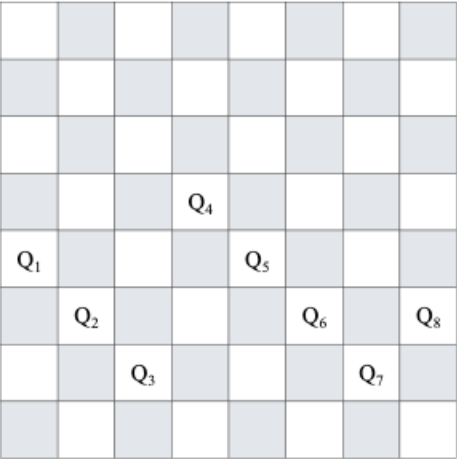
[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Count the Neighbours

Slide 90 of 193 · Reveal 2

Evaluate the Neighbours

What would the cost be after each move?



Each number is the cost of the whole board after moving that column's queen

91

Speaker notes

A square represents moving its column's queen there, keeping all other queens fixed

Evaluate the Neighbours

Slide 91 of 193 · Initial

Evaluate the Neighbours

What would the cost be after each move?

	12						
			Q ₄				
Q ₁				Q ₅			
	Q ₂			Q ₆		Q ₈	
		Q ₃			Q ₇		

Each number is the cost of the whole board after moving that column's queen

Example: move Q₂ to the top row
The resulting board costs 12

91

Speaker notes

The 12 is the cost of the entire resulting board, not the moved queen alone

Evaluate the Neighbours

Slide 91 of 193 · Reveal 1

Evaluate the Neighbours

What would the cost be after each move?

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	Q ₄	13	16	13	16
Q ₁	14	17	15	Q ₅	14	16	16
17	Q ₂	16	18	15	Q ₆	15	Q ₈
18	14	Q ₃	15	15	14	Q ₇	16
14	14	13	17	12	14	12	18

Each number is the cost of the whole board after moving that column's queen

Example: move Q₂ to the top row
The resulting board costs 12

The current board costs 17

Lowest of all 56 costs: 12

Several moves tie; choose one at random

91

Speaker notes

Evaluate every possible move. Red entries tie for the lowest cost

Evaluate the Neighbours

Slide 91 of 193 · Reveal 2

Improve the Current State

Cost = attacking pairs · Lower is better

Current state



Cost: 17

92

Speaker notes

Keep one current board. Its cost counts attacking pairs; lower is better

Improve the Current State

Slide 92 of 193 · Initial

Improve the Current State

Cost = attacking pairs · Lower is better

Compare neighbouring states

Current state

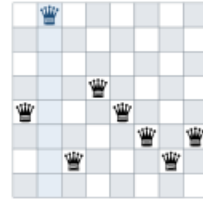


Cost: 17

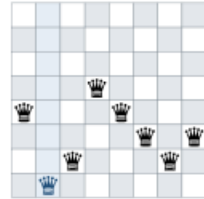
Three of the 56 neighbours; each moves the highlighted queen



Cost: 16



Cost: 12



Cost: 14

92

Speaker notes

These are three example neighbours, not all possible moves. Each changes the highlighted queen's row

Improve the Current State

Slide 92 of 193 · Reveal 1

Improve the Current State

Cost = attacking pairs · Lower is better

Compare neighbouring states

Current state



Cost: 17

Three of the 56 neighbours; each moves the highlighted queen

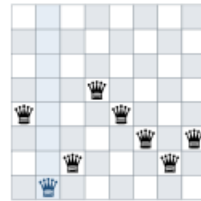


Cost: 16



Cost: 12

Choose this state



Cost: 14

92

Speaker notes

This example chooses the lowest-cost neighbour shown

Improve the Current State

Slide 92 of 193 · Reveal 2

Improve the Current State

Cost = attacking pairs · Lower is better

Compare neighbouring states

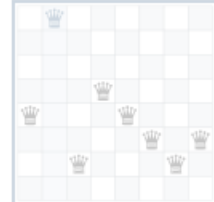
Three of the 56 neighbours; each moves the highlighted queen



Cost: 12

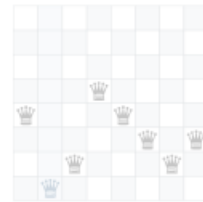


Cost: 16



Cost: 12

Choose this state



Cost: 14

Replace the current state, then repeat

92

Speaker notes

Replace the current state. The old board does not become a stored path or search tree

Improve the Current State

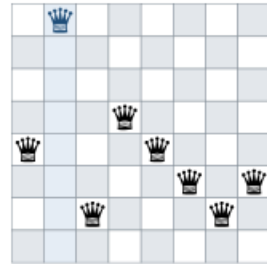
Slide 92 of 193 · Reveal 3

Improve the Current State

Cost = attacking pairs · Lower is better

Compare neighbouring states

Three of the 56 neighbours; each moves the highlighted queen



Cost: 12

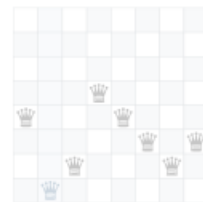


Cost: 16



Cost: 12

Choose this state



Cost: 14

Replace the current state, then repeat

Very little memory · Often finds good solutions in large spaces

92

Speaker notes

Local search can use very little memory and find good solutions in large spaces, without guaranteeing the best solution

Improve the Current State

Slide 92 of 193 · Reveal 4

Hill Climbing: One Attempt

Steepest descent for a cost we want to minimise

1. Start with a random state
2. If cost = 0, return the solution
3. Find the lowest-cost neighbour
 - Break ties at random
4. If its cost $\geq$ current cost, stop this attempt
5. Otherwise replace the current state; return to step 2

Accept only strictly lower costs

93

Speaker notes

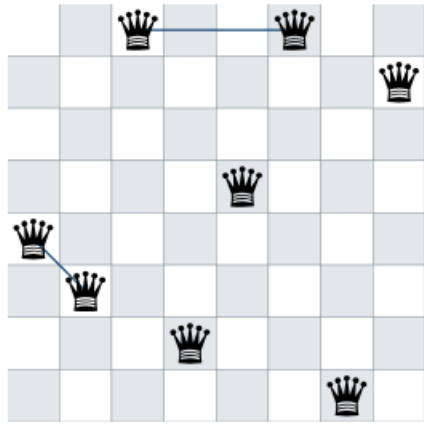
This is one hill-climbing attempt. Equal cost does not improve the state, so this version stops; restarts come later

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Hill Climbing: One Attempt

Slide 93 of 193 · Initial

When Improvement Stops



Current cost: 2

94

Speaker notes

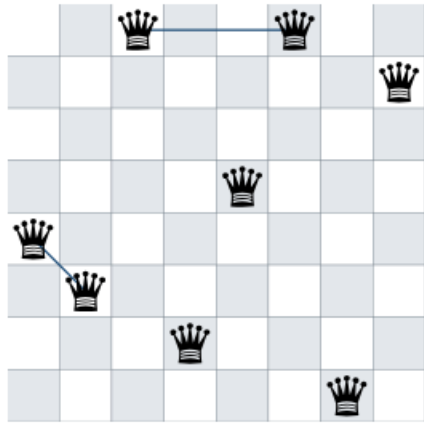
Computed state (4, 5, 0, 6, 3, 0, 7, 1); cost trace [5, 3, 2]. The minimum over all 56 neighbours is 2. A plateau also stops this strict version

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

When Improvement Stops

Slide 94 of 193 · Initial

When Improvement Stops



Current cost: 2

Lowest neighbour cost: 2

94

Speaker notes

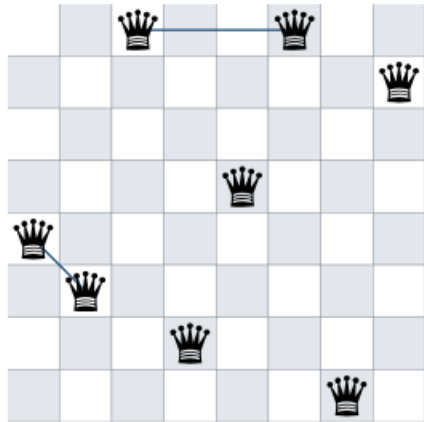
Computed state (4, 5, 0, 6, 3, 0, 7, 1); cost trace [5, 3, 2]. The minimum over all 56 neighbours is 2. A plateau also stops this strict version

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

When Improvement Stops

Slide 94 of 193 · Reveal 1

When Improvement Stops



Current cost: 2

Lowest neighbour cost: 2

No strictly better move

Stop, but not solved

An equal-cost move would be a sideways move

94

Speaker notes

Computed state (4, 5, 0, 6, 3, 0, 7, 1); cost trace [5, 3, 2]. The minimum over all 56 neighbours is 2. A plateau also stops this strict version

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

When Improvement Stops

Slide 94 of 193 · Reveal 2

main.py queens.py

```
1 from queens import experiment, format_mean as fmt
2 trials, seed = 1000, 1 # Change these and run again
3 result = experiment(trials, seed)
4 print(f"{trials} independent attempts; seed {seed}")
5 print(f"Solved: {result['success_rate']:.1%}")
6 print("Mean moves if solved:", fmt(result["mean_success_moves"]))
7 print("Mean moves if stopped:", fmt(result["mean_stopped_moves"]))
8
```

Download: comp3270.dirs.hk/eight_queens_hill_climbing.py

Try the Experiment

Slide 95 of 193 · Initial

Speaker notes

Run 1,000 attempts first. Open queens.py to inspect the algorithm. The next slide reports a larger recorded experiment, not this small live sample

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Does the Experiment Match?

Five seeds × 10,000 independent attempts

Solved	Mean moves if solved	Mean moves if stopped
14.07%	4.08	3.06

7,035 solutions in 50,000 attempts

Success across the five seeds: 13.23%–14.42%

One strict attempt, without sideways moves · Accepted moves only

96

Speaker notes

Uniform random rows, one queen per column; random tie breaking. Moves count accepted changes, not candidate evaluations. Textbook figures are rounded benchmarks, not guaranteed success rates

Recorded runs: `local_search/experiments.json`

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

Does the Experiment Match?

Slide 96 of 193 · Initial

Does the Experiment Match?

Five seeds × 10,000 independent attempts

Solved	Mean moves if solved	Mean moves if stopped
14.07%	4.08	3.06

7,035 solutions in 50,000 attempts

Success across the five seeds: 13.23%–14.42%

Close to AIMA's approximate 14% success and 4 / 3 moves

One strict attempt, without sideways moves · Accepted moves only

96

Speaker notes

Uniform random rows, one queen per column; random tie breaking. Moves count accepted changes, not candidate evaluations. Textbook figures are rounded benchmarks, not guaranteed success rates

Recorded runs: `local_search/experiments.json`

[Sources] - Original COMP3270 2024–25 local-search section; verified with the accompanying queens implementation - AIMA, Figures 4.11–4.12 and local-search discussion: <https://aima.cs.berkeley.edu/4th-ed/pdfs/newchap04.pdf>

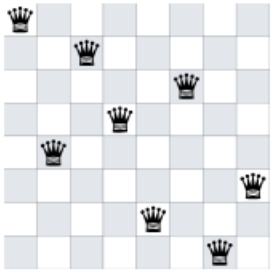
Does the Experiment Match?

Slide 96 of 193 · Reveal 1

Try a New Starting State

Random restart: begin each attempt from a fresh random board

Attempt 1



Stopped: cost 1

Repeat until solved, or a chosen attempt limit is reached

The 14% success rate describes one attempt

97

Speaker notes

These are final boards from three separate attempts, not successive moves.
The first stops at cost 1

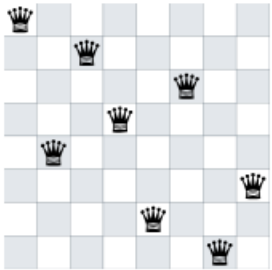
Try a New Starting State

Slide 97 of 193 · Initial

Try a New Starting State

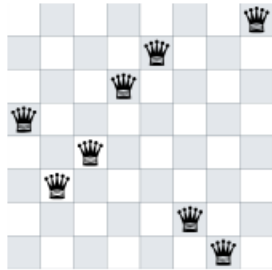
Random restart: begin each attempt from a fresh random board

Attempt 1



Stopped: cost 1

Attempt 2



Stopped: cost 3

Repeat until solved, or a chosen attempt limit is reached

The 14% success rate describes one attempt

97

Speaker notes

A fresh random start also stops, now at cost 3

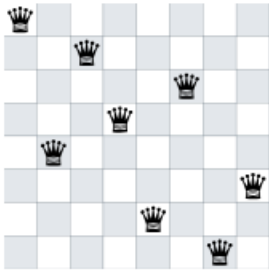
Try a New Starting State

Slide 97 of 193 · Reveal 1

Try a New Starting State

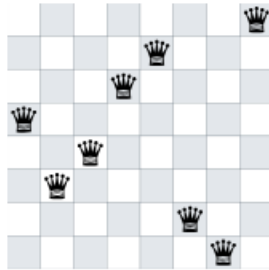
Random restart: begin each attempt from a fresh random board

Attempt 1



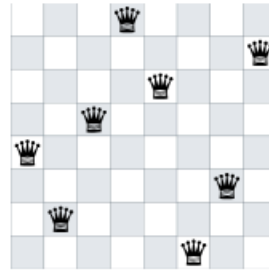
Stopped: cost 1

Attempt 2



Stopped: cost 3

Attempt 3



Solved: cost 0

Repeat until solved, or a chosen attempt limit is reached

The 14% success rate describes one attempt

97

Speaker notes

The third fresh start reaches cost zero. Three attempts means two additional restarts; this example is not a guarantee

Try a New Starting State

Slide 97 of 193 · Reveal 2

Task Preparation

Log in to Moodle → Click Ed

98

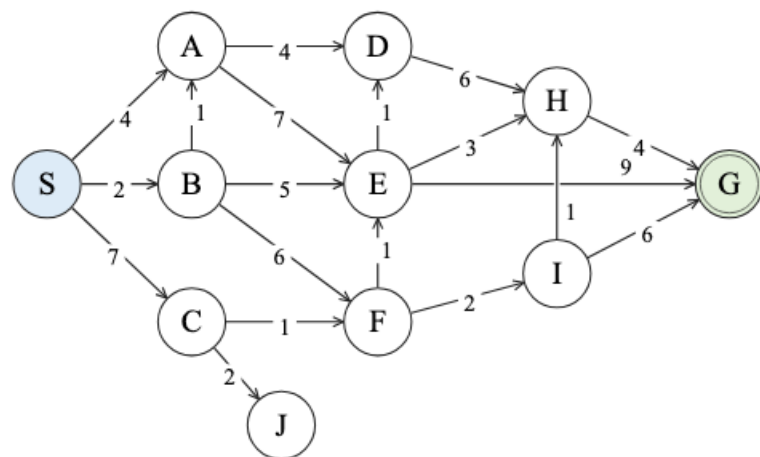
Task Preparation

Slide 98 of 193 · Initial

Speaker notes

Open Ed through Moodle before starting the tasks.

Task: BFS-GSA



TASK

Start at **S**, goal **G**

Frontier: Add alphabetically

1. Exploration Order

2. Solution path

Uppercase strings, no spaces

Example: SAG

[Submit on Ed](#)

99

Speaker notes

Students submit Task 1 on Ed. Use the lecture's graph-search conventions, including DFS stack order and whole-path priority ties.

Task: BFS-GSA

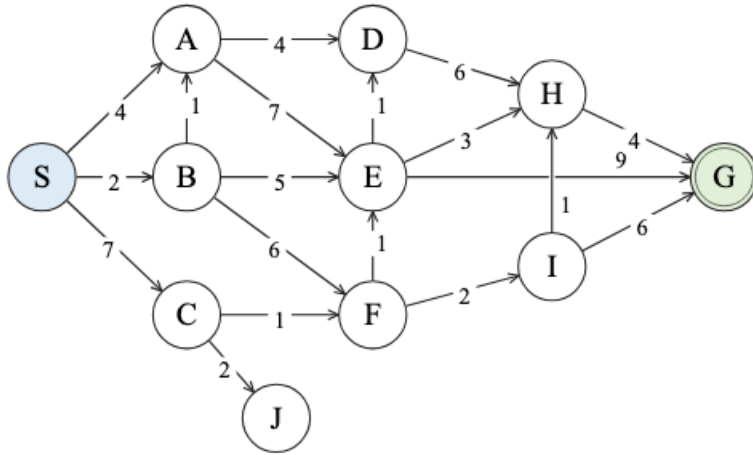
Slide 99 of 193 · Initial

Solution: BFS-GSA

SOLUTION

Explored
SABCDEFJH

Solution
SAEG



99

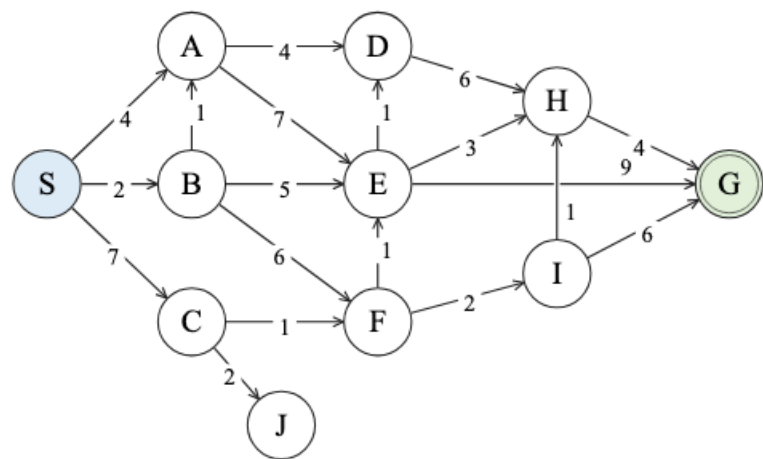
Task: BFS-GSA

Slide 99 of 193 · Reveal 1

Speaker notes

Reveal the explored order and path after submissions. The fewest-edge path need not have the lowest cost.

Task: DFS-TSA



TASK

Start at **S**, goal **G**

Frontier: Add alphabetically

1. Exploration Order

2. Solution path

Uppercase strings, no spaces

Example: SAG

[Submit on Ed](#)

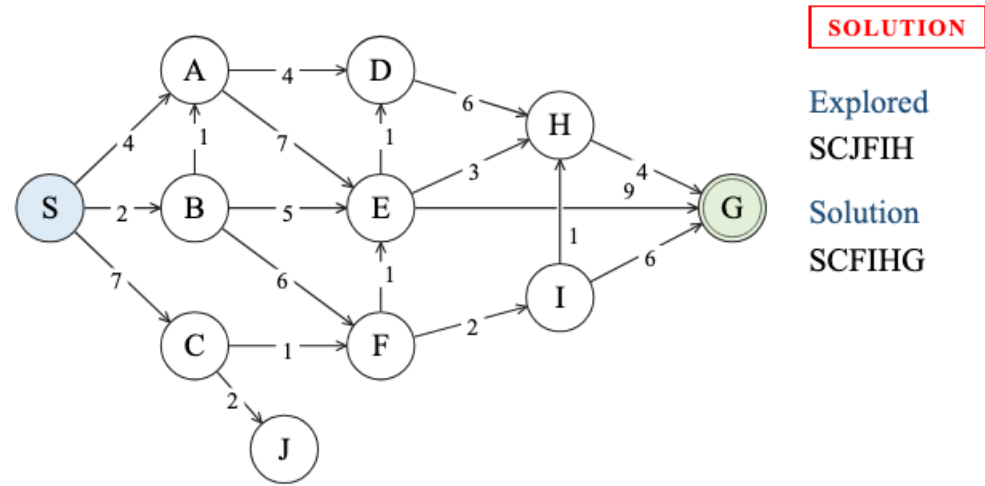
100

Speaker notes

Students submit Task 2 on Ed. Use tree search. Retain repeated states if encountered; push children alphabetically. This example reaches G after six explored states.

Task: DFS-TSA

Solution: DFS-TSA



100

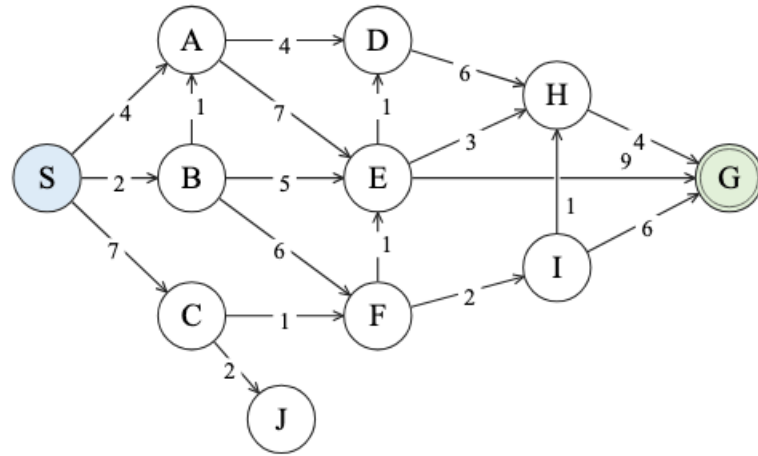
Task: DFS-TSA

Slide 100 of 193 · Reveal 1

Speaker notes

Reveal the explored order and path after submissions. DFS explores the last alphabetically pushed child first; J is a dead end.

Task: UCS-GSA



TASK

Start at **S**, goal **G**

Frontier: Add alphabetically

1. Exploration Order

2. Solution path

Uppercase strings, no spaces

Example: SAG

[Submit on Ed](#)

101

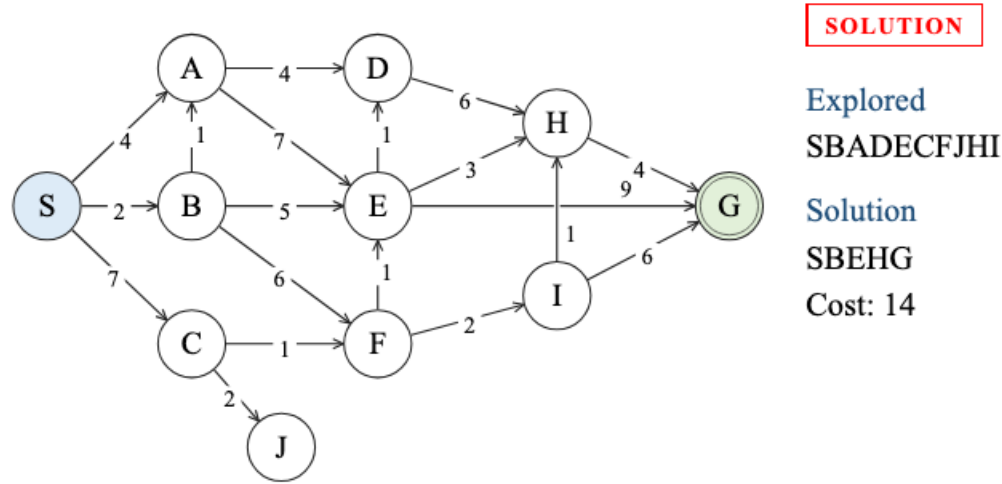
Speaker notes

Students submit Task 3 on Ed. Use the lecture's graph-search conventions, including DFS stack order and whole-path priority ties.

Task: UCS-GSA

Slide 101 of 193 · Initial

Solution: UCS-GSA

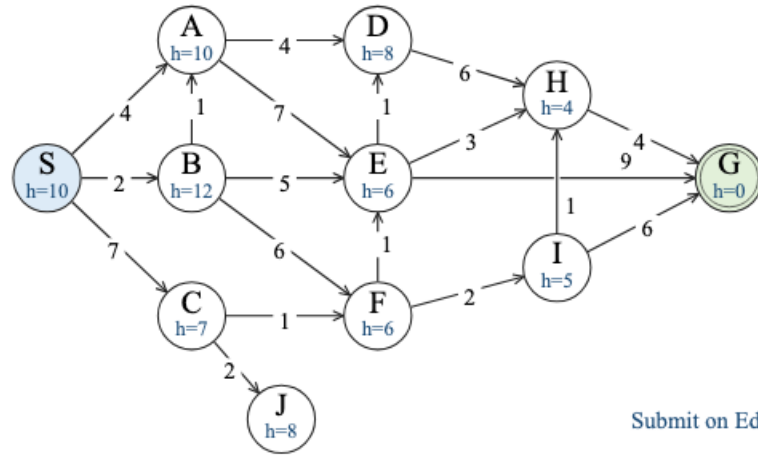


Task: UCS-GSA

Speaker notes

Reveal the explored order and path after submissions. A cheaper route to A is discovered through B.

Task: A*-GSA



Submit on Ed

TASK

Start at **S**, goal **G**

Frontier: Add alphabetically

1. Exploration Order
2. Solution path
3. Admissible?
4. Consistent?

Uppercase strings, no spaces
Example: SAG

102

Speaker notes

Students submit Task 4 on Ed. Use the lecture's graph-search conventions, including DFS stack order and whole-path priority ties.

Task: A*-GSA

Slide 102 of 193 · Initial

Solution: A*-GSA

SOLUTION

Explored: SABEH

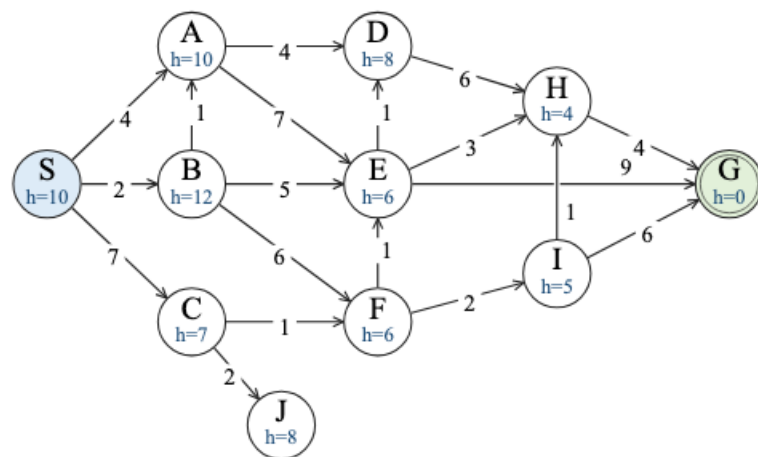
Solution: SBEHG

Cost: 14

Admissible: Yes

 $h(n) \leq h^*(n)$

Consistent: No

 $B \rightarrow A: 1 < 12 - 10$ $B \rightarrow E: 5 < 12 - 6$ 

102

Speaker notes

Reveal the explored order and path after submissions. A is explored at cost 4 before B. The improved route to A at cost 3 is skipped because this GSA never reopens explored states. It still finds the optimum here, but admissibility alone does not guarantee that.

Admissible at every state; $h^*(B)=12$. Consistency fails on B to A and B to E. J is a dead end with infinite true remaining cost.

Task: A*-GSA

Slide 102 of 193 · Reveal 1

Task: Local Search

	A	B	C	D	E
1	Q ₁	Q ₂			
2					
3				Q ₄	
4					
5			Q ₃		Q ₅

One queen per column · Move within its column

TASK

Minimise attacking pairs
No sideways moves

- 1. Sequence of moves
- 2. What is the cost after the last move?

Uppercase cells, separated by one space
Example: B4 E2

[Submit on Ed](#)

103

Speaker notes

Students submit the target-cell sequence and final cost.

Task: Local Search

Slide 103 of 193 · Initial

Solution: Local Search

	A	B	C	D	E
1		Q ₂			
2				Q ₄	
3					
4	Q ₁				
5			Q ₃		Q ₅

SOLUTION

1. A4 D2

2. 1

One queen per column · Move within its column

103

Speaker notes

A4 D2. Final cost 1: Q3 and Q5 still attack. No strictly improving move remains.

Task: Local Search

Slide 103 of 193 · Reveal 1

Constraint Satisfaction Problems

104

Constraint Satisfaction Problems

Slide 104 of 193 · Initial

Speaker notes

Now keep the structure inside each state: variables, domains and constraints.

[Sources] - Recovered COMP3270 2024–25 source slide 94:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Satisfaction Problems

A **factored state** records a value for each variable

A solution assigns every variable a value that satisfies all constraints

Use constraints to rule out combinations before exploring them

105

Constraint Satisfaction Problems

Slide 105 of 193 · Initial

Speaker notes

Contrast this with treating a state as a single opaque object.

[Sources] - Recovered COMP3270 2024–25 source slide 95:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Defining CSPs

Variables

$$X = \{X_1, \dots, X_n\}$$

Domains

D_i contains the allowed values of X_i

Constraints

Allowable combinations of values

A state is a partial or complete assignment

$$\{X_i = v_i, X_j = v_j, \dots\}$$

106

Defining CSPs

Slide 106 of 193 · Initial

Speaker notes

A domain lists possibilities. An assignment chooses a value.

[Sources] - Recovered COMP3270 2024–25 source slide 96:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Solutions to CSPs

Consistent: no constraint is violated

Complete: every variable has a value

A **solution** is both consistent and complete

107

Solutions to CSPs

Slide 107 of 193 · Initial

Speaker notes

A partial assignment can be consistent without being a solution.

[Sources] - Recovered COMP3270 2024–25 source slide 97:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Australia: Map Colouring

Variables

$X = \{WA, NT, Q, NSW, V, SA, T\}$

Domains

$D_i = \{\text{red, green, blue}\}$

Constraints

Neighbours have different colours



108

Speaker notes

Ask which pairs of regions share a border. Tasmania has no neighbours.

[Sources] - Recovered COMP3270 2024–25 source slide 98:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Australia: Map Colouring

Slide 108 of 193 · Initial

Australia: Map Colouring

Variables

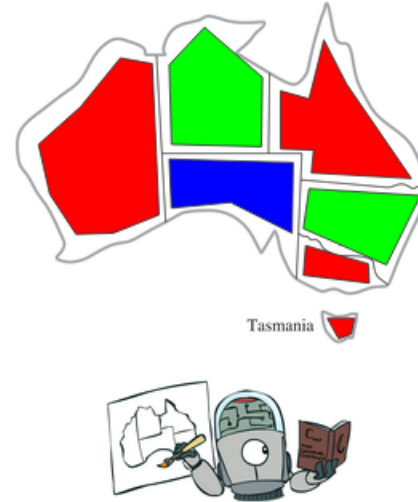
$X = \{\text{WA, NT, Q, NSW, V, SA, T}\}$

Domains

$D_i = \{\text{red, green, blue}\}$

Constraints

Neighbours have different colours



108

Speaker notes

Ask which pairs of regions share a border. Tasmania has no neighbours.

[Sources] - Recovered COMP3270 2024–25 source slide 98:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Australia: Map Colouring

Slide 108 of 193 · Reveal 1

Constraint Graph

Nodes represent variables

Edges represent constraints

Tasmania is independent
of the mainland variables



109

Speaker notes

Edges encode constraints, not routes to travel along.

[Sources] - Recovered COMP3270 2024–25 source slide 99:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

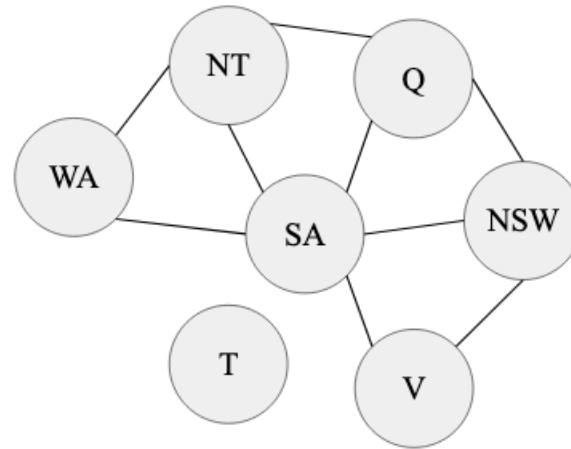
Constraint Graph It can be helpful to visualize a CSP as a constraint graph Nodes correspond to variables Arcs show existence of constraints 99 Q NSW V NT WA SA

Constraint Graph

Nodes represent variables

Edges represent constraints

Tasmania is independent
of the mainland variables



109

Speaker notes

Edges encode constraints, not routes to travel along.

[Sources] - Recovered COMP3270 2024–25 source slide 99:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Graph It can be helpful to visualize a CSP as a constraint graph. Nodes correspond to variables. Arcs show existence of constraints.

Slide 109 of 193 · Reveal 1
SA

Solving CSPs

State

The values assigned so far

Initial State

Empty assignment {}

Successor

Assign a value to an unassigned variable

Goal Test

Complete and consistent

110

Solving CSPs

Slide 110 of 193 · Initial

Speaker notes

This first formulation permits different variable orders. The next trees show the cost.

[Sources] - Recovered COMP3270 2024–25 source slide 100:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

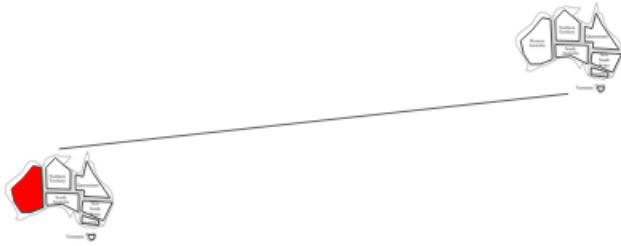
Slide 111 of 193 · Initial

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

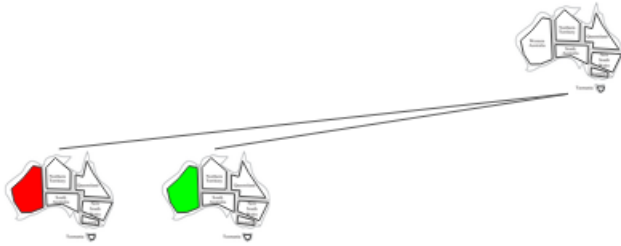
Slide 111 of 193 · Reveal 1

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

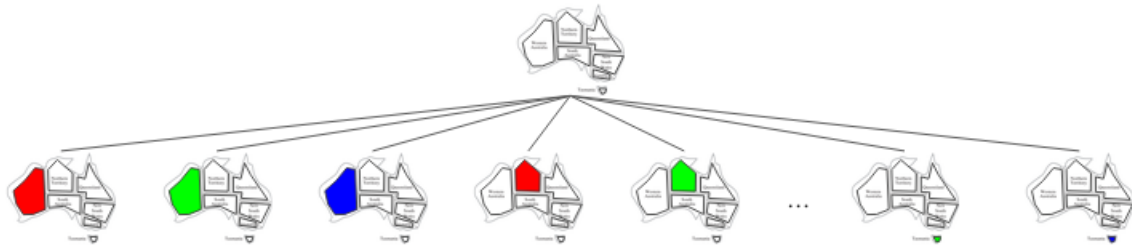
Slide 111 of 193 · Reveal 2

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

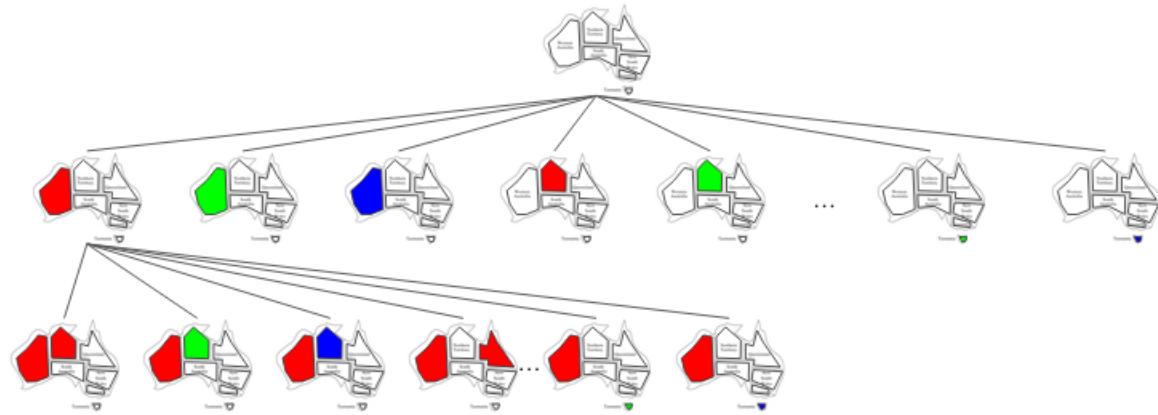
Slide 111 of 193 · Reveal 3

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

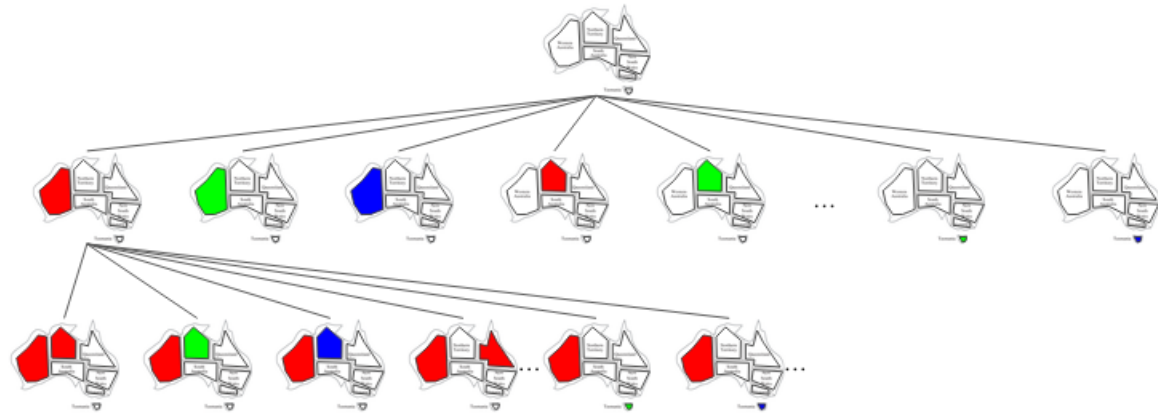
Slide 111 of 193 · Reveal 4

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024-25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

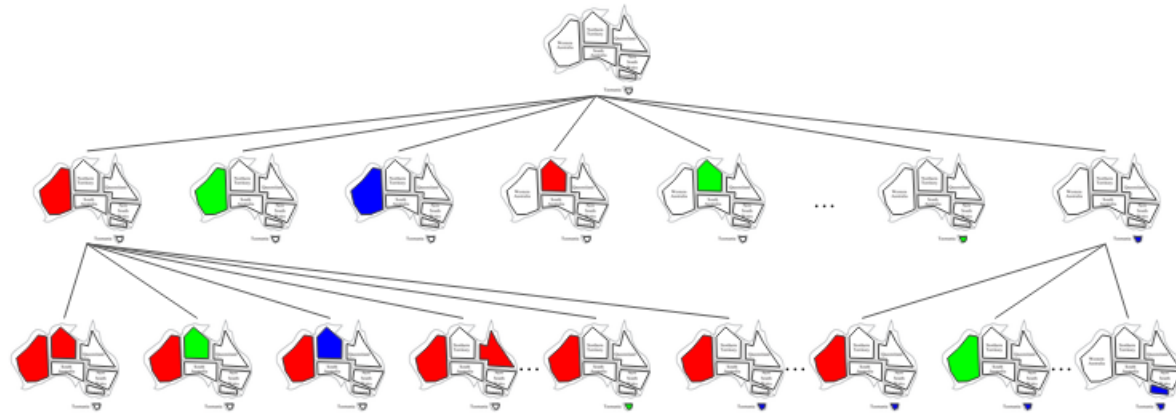
Slide 111 of 193 · Reveal 5

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024-25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



111

Breadth-First Search

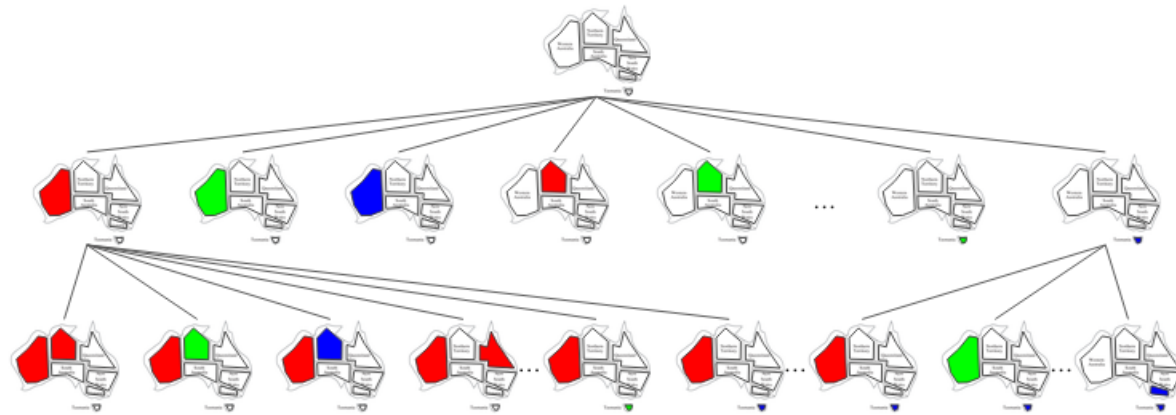
Slide 111 of 193 · Reveal 6

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search



All solutions are at depth 7

111

Speaker notes

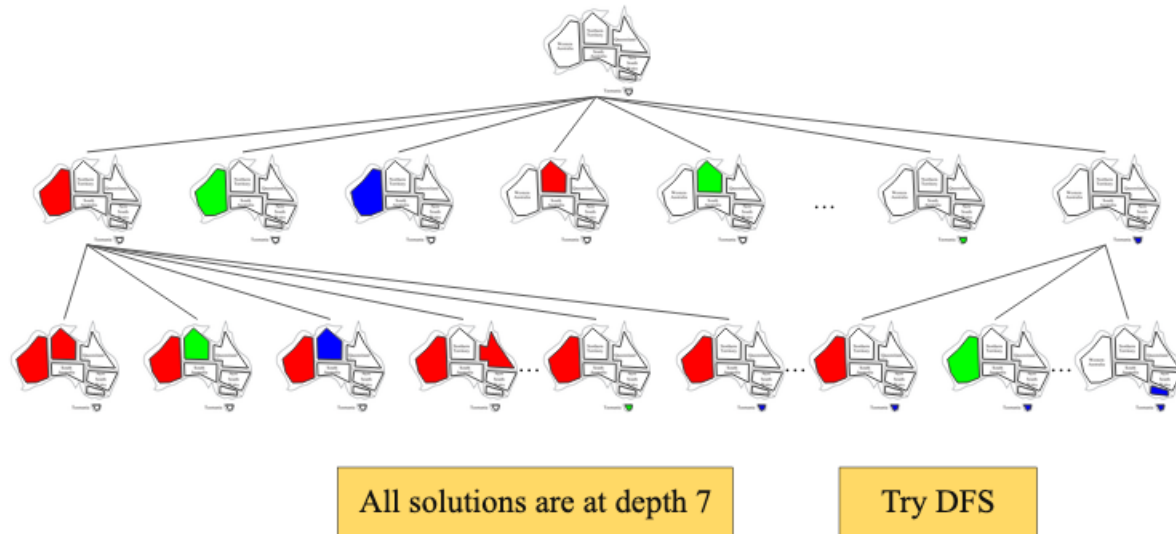
All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Breadth-First Search

Slide 111 of 193 · Reveal 7

Breadth-First Search



111

Breadth-First Search

Slide 111 of 193 · Reveal 8

Speaker notes

All complete assignments have the same depth. Breadth-first search spends time on partial assignments.

[Sources] - Recovered COMP3270 2024–25 source slide 101:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

Slide 112 of 193 · Initial

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

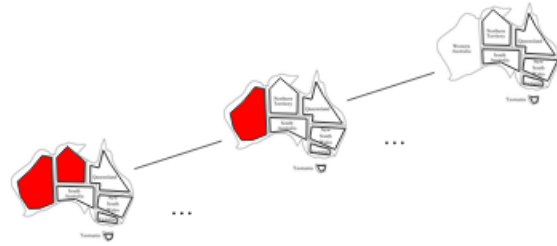
Slide 112 of 193 · Reveal 1

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

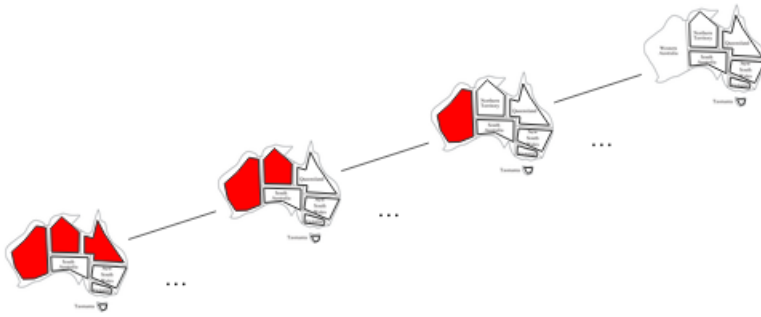
Slide 112 of 193 · Reveal 2

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

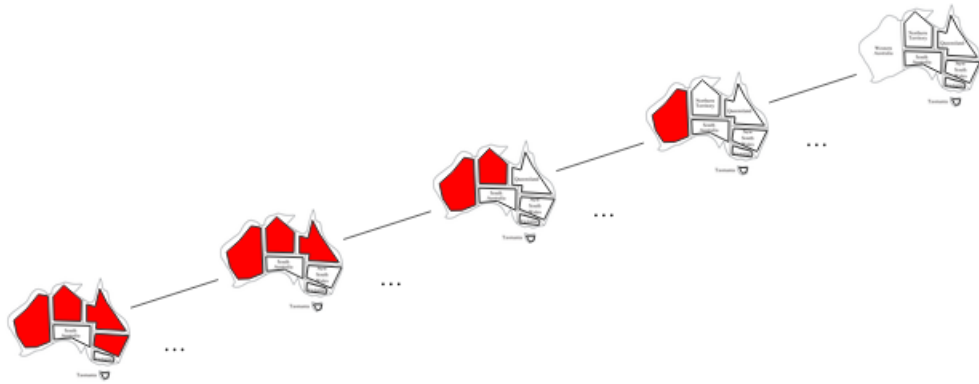
Slide 112 of 193 · Reveal 3

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

Speaker notes

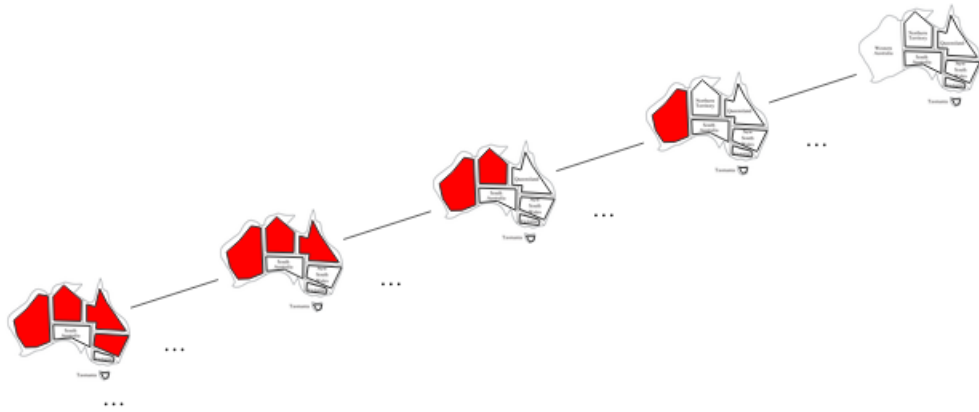
An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS

Slide 112 of 193 · Reveal 4

DFS



112

DFS

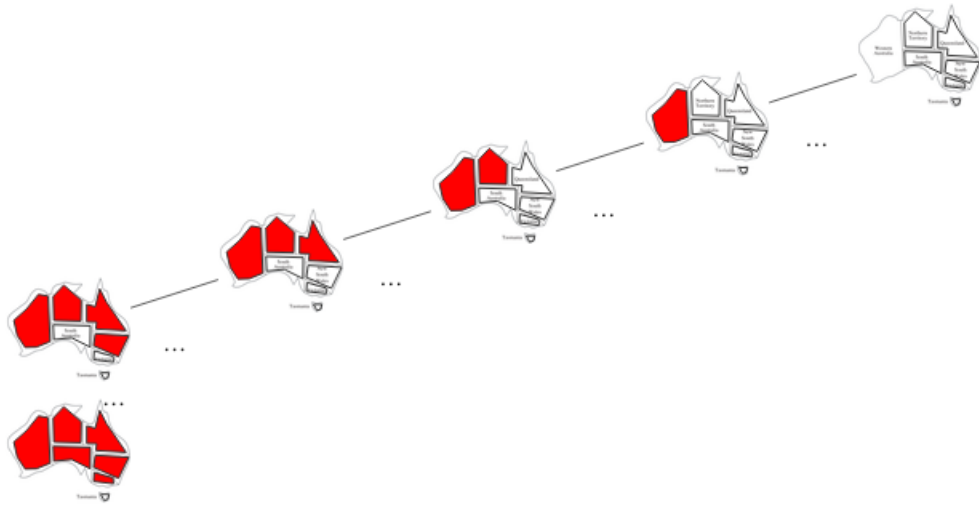
Slide 112 of 193 · Reveal 5

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

Speaker notes

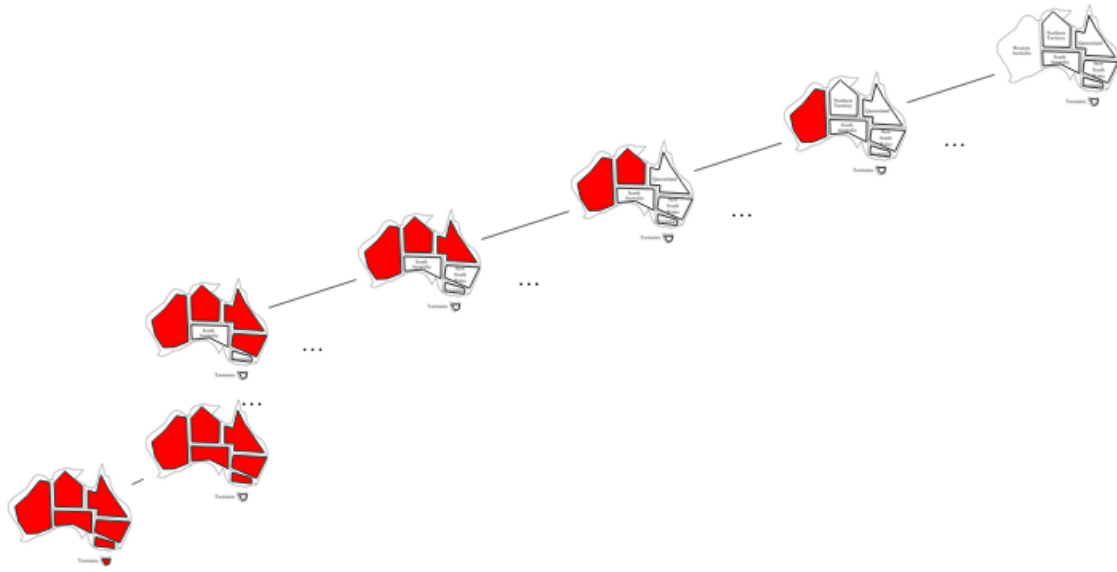
An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS

Slide 112 of 193 · Reveal 6

DFS



112

DFS

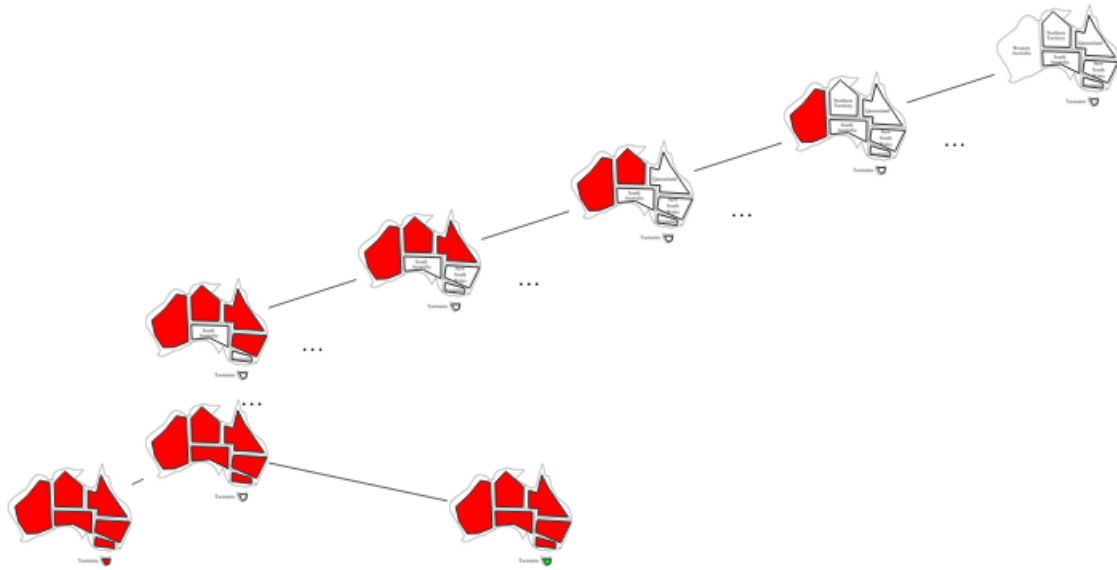
Slide 112 of 193 · Reveal 7

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

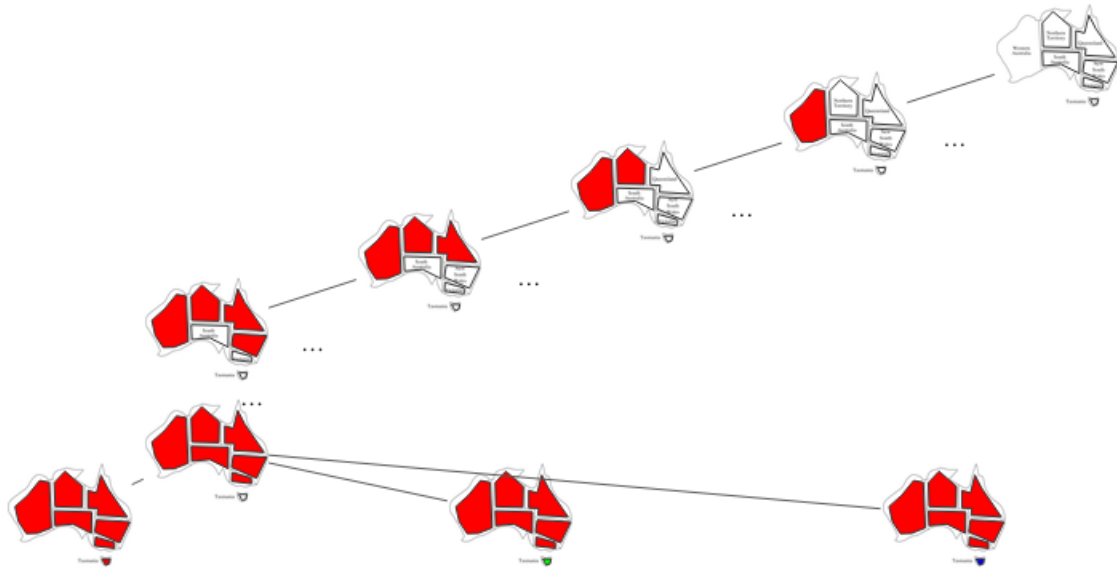
Slide 112 of 193 · Reveal 8

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

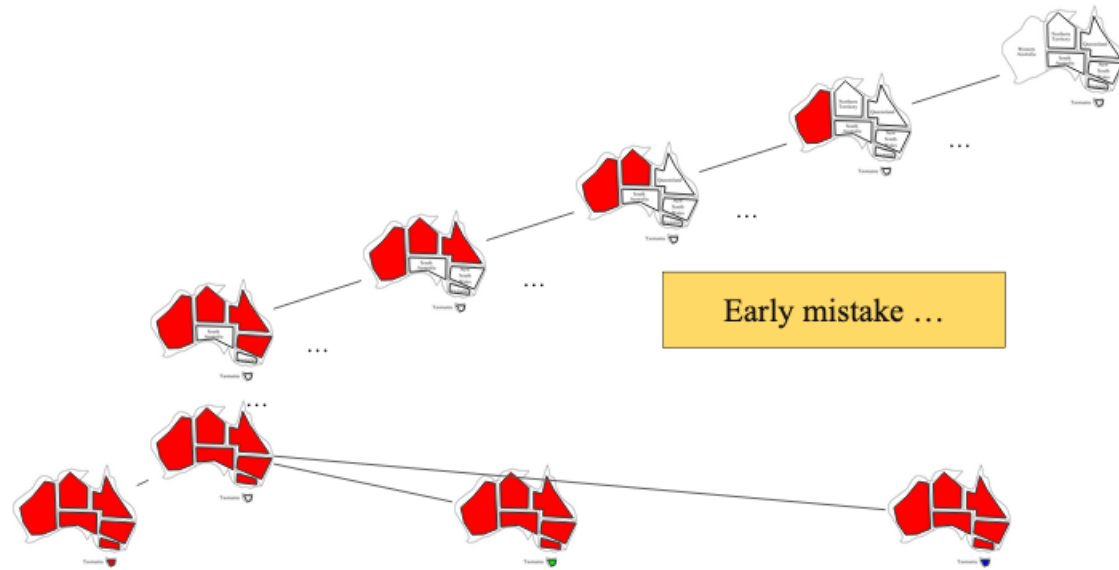
Slide 112 of 193 · Reveal 9

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

DFS



112

DFS

Slide 112 of 193 · Reveal 10

Speaker notes

An early conflicting assignment should be rejected immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 102:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Backtracking Search



Choose **one unassigned variable** at each level

Branch over its legal values

Try only values consistent with earlier assignments

If no legal value remains, **backtrack**

Depth-first search with these two ideas is backtracking search

113

Backtracking Search

Slide 113 of 193 · Initial

Speaker notes

Variable assignments commute. Fix one variable at each level and reject conflicts immediately.

[Sources] - Recovered COMP3270 2024–25 source slide 103:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Backtracking



114

Backtracking 104 $D = \{\text{red, green, blue}\}$ $X = \{\text{WA, NT, Q, NSW, V, SA, T}\} \dots$

Slide 114 of 193 · Initial

Speaker notes

Fix the variable order, then reject illegal values immediately. Keep only legal children. Continue the legal branch.

Backtracking



114

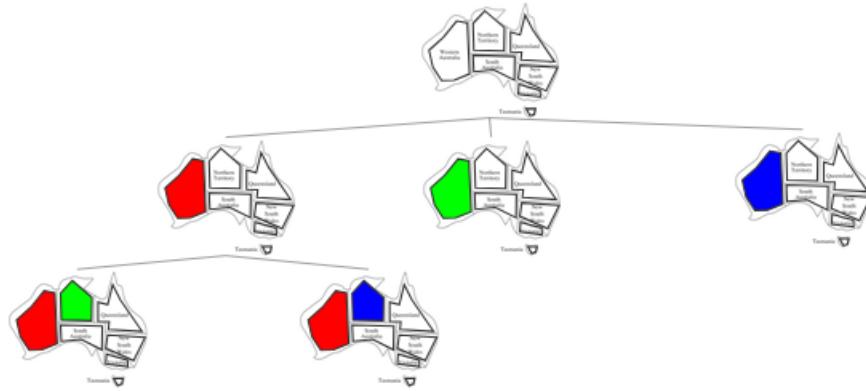
Backtracking 104 $D = \{\text{red, green, blue}\}$ $X = \{\text{WA, NT, Q, NSW, V, SA, T}\} \dots$

Slide 114 of 193 · Reveal 1

Speaker notes

Fix the variable order, then reject illegal values immediately. Keep only legal children. Continue the legal branch.

Backtracking



114

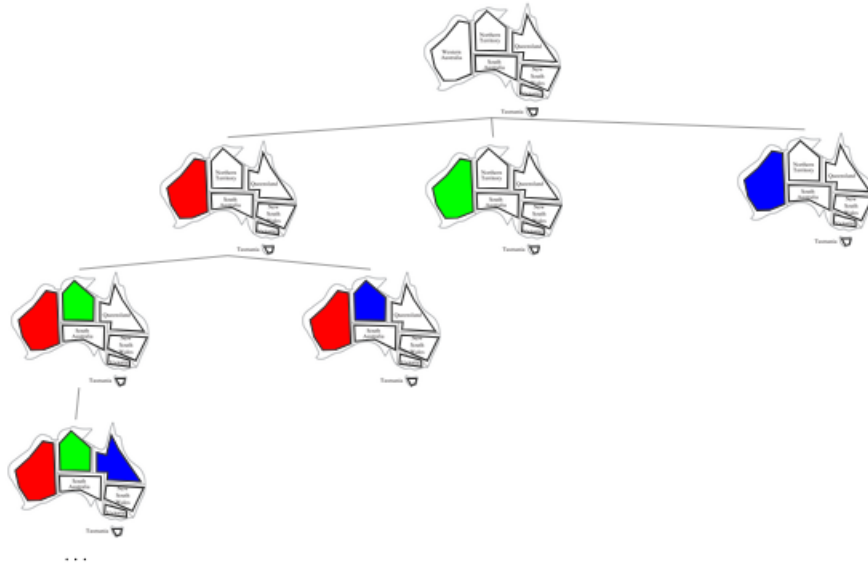
Backtracking 104 $D = \{\text{red, green, blue}\}$ $X = \{\text{WA, NT, Q, NSW, V, SA, T}\} \dots$

Slide 114 of 193 · Reveal 2

Speaker notes

Fix the variable order, then reject illegal values immediately. Keep only legal children. Continue the legal branch.

Backtracking



114

Backtracking 104 $D = \{\text{red, green, blue}\}$ $X = \{\text{WA, NT, Q, NSW, V, SA, T}\} \dots$

Slide 114 of 193 · Reveal 3

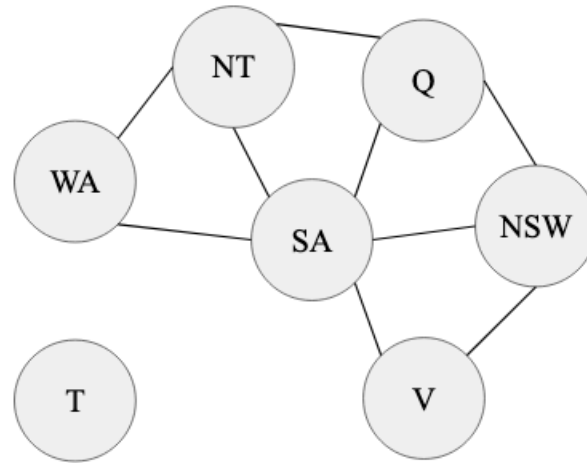
Speaker notes

Fix the variable order, then reject illegal values immediately. Keep only legal children. Continue the legal branch.

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Follow NSW, WA, NT, Q, SA, V, T; try red, green, blue. Each click shows a candidate, including rejected colours.

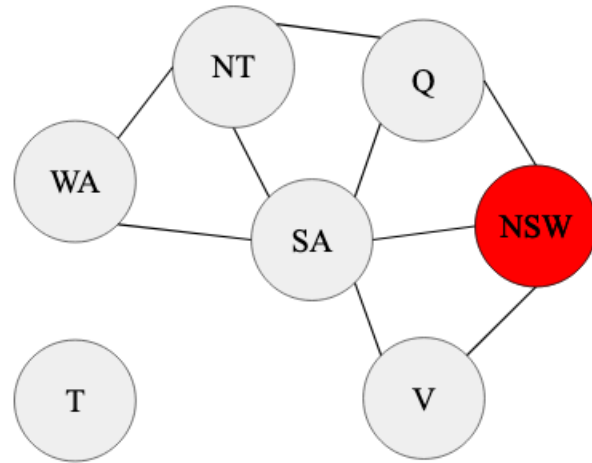
Backtracking

Slide 115 of 193 · Initial

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign NSW red.

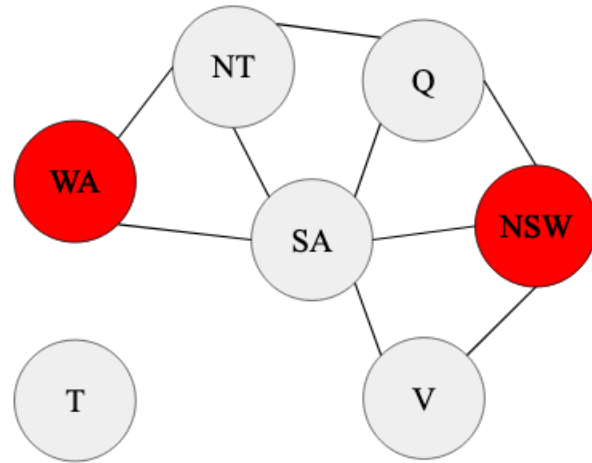
Backtracking

Slide 115 of 193 · Reveal 1

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

WA may also be red: WA and NSW are not neighbours.

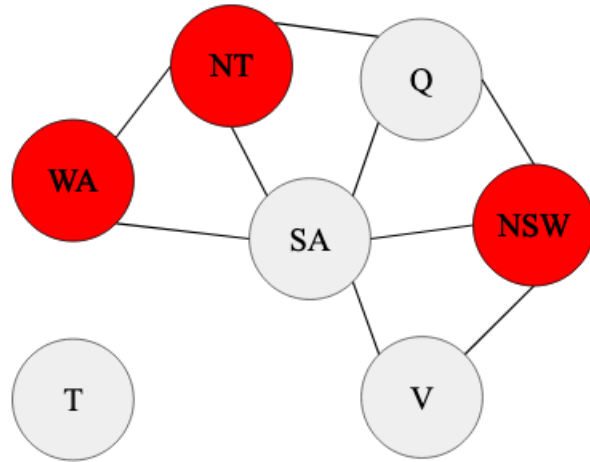
Backtracking

Slide 115 of 193 · Reveal 2

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject NT red: it conflicts with WA.

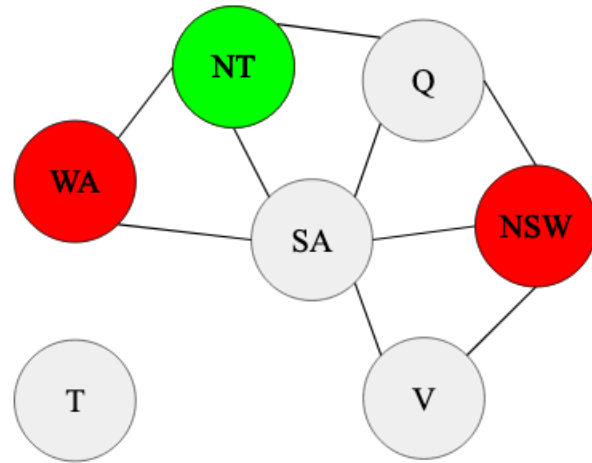
Backtracking

Slide 115 of 193 · Reveal 3

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign NT green.

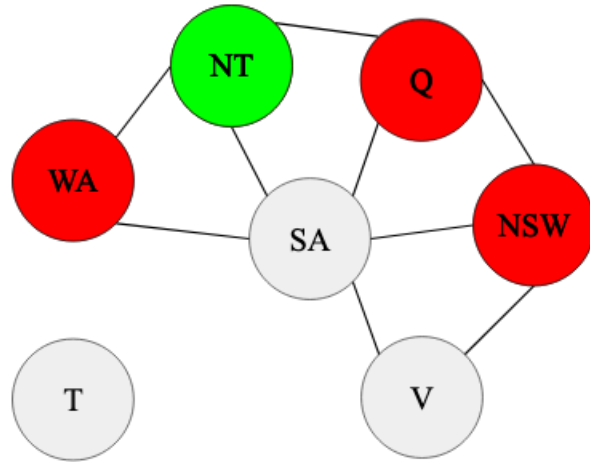
Backtracking

Slide 115 of 193 · Reveal 4

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject Q red: it conflicts with NSW.

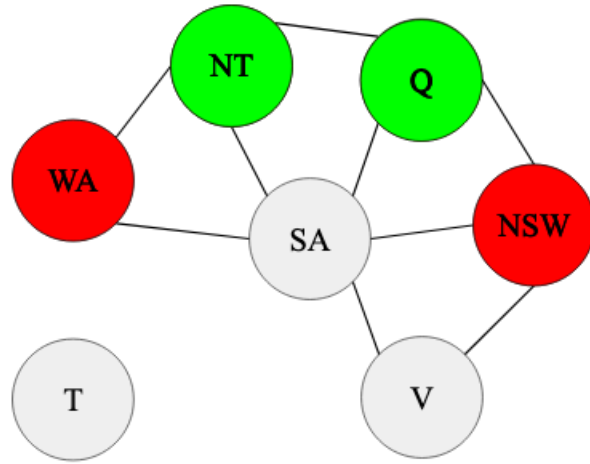
Backtracking

Slide 115 of 193 · Reveal 5

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject Q green: it conflicts with NT.

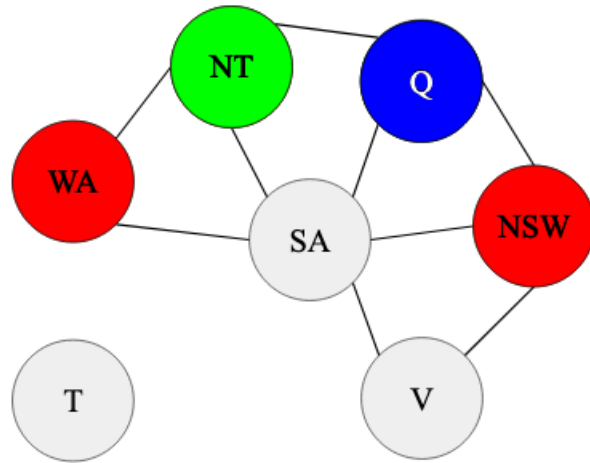
Backtracking

Slide 115 of 193 · Reveal 6

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign Q blue.

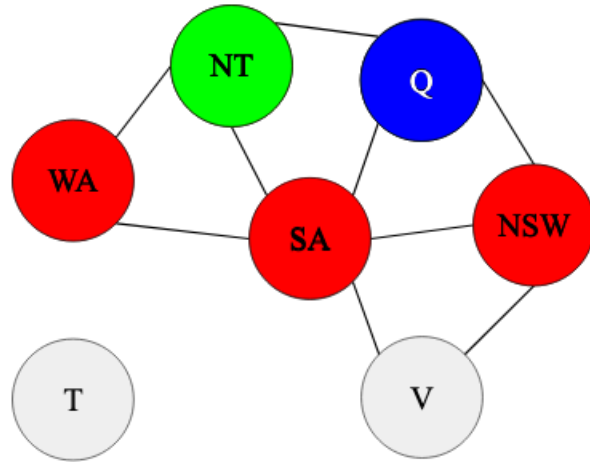
Backtracking

Slide 115 of 193 · Reveal 7

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA red: it conflicts with NSW and WA.

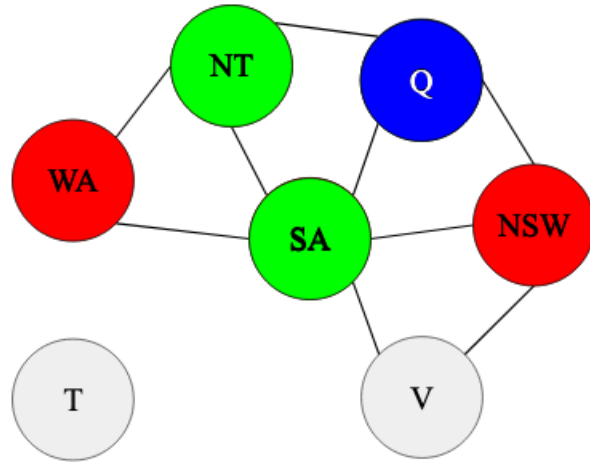
Backtracking

Slide 115 of 193 · Reveal 8

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA green: it conflicts with NT.

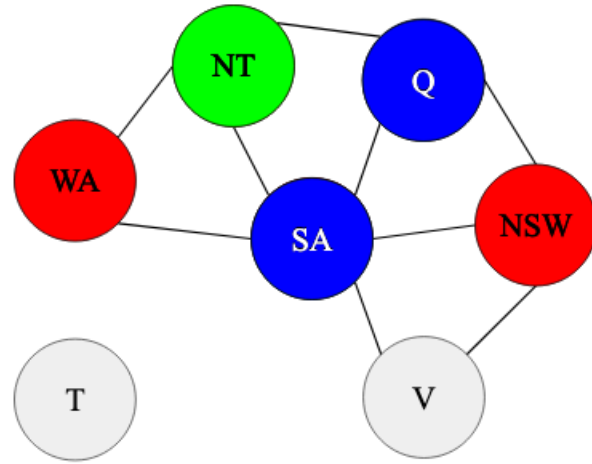
Backtracking

Slide 115 of 193 · Reveal 9

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA blue: it conflicts with Q. No SA value works; backtrack.

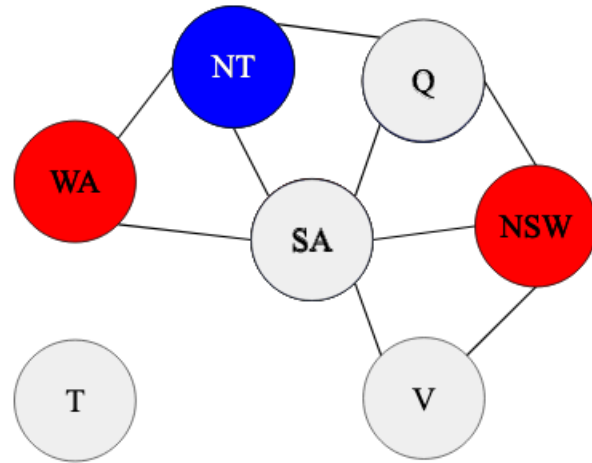
Backtracking

Slide 115 of 193 · Reveal 10

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Restore Q and SA to unassigned. Try NT blue.

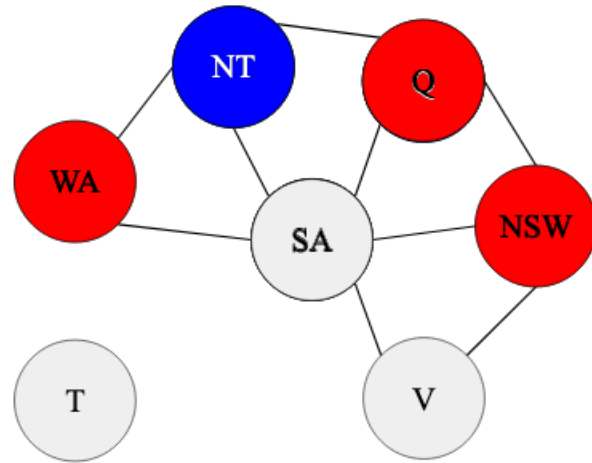
Backtracking

Slide 115 of 193 · Reveal 11

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject Q red: it conflicts with NSW.

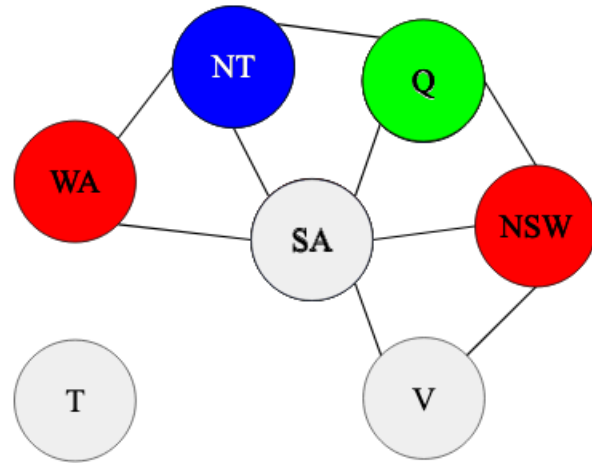
Backtracking

Slide 115 of 193 · Reveal 12

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign Q green.

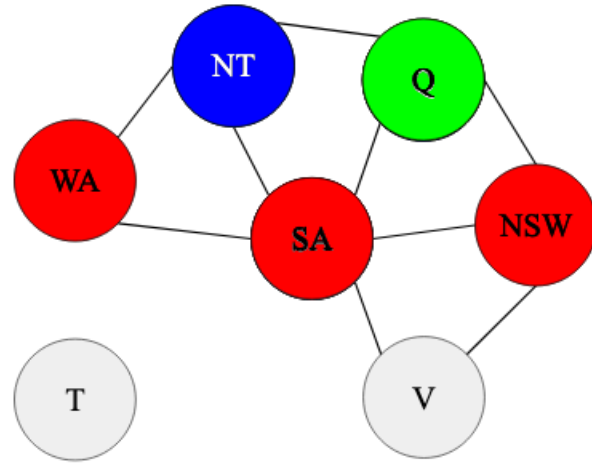
Backtracking

Slide 115 of 193 · Reveal 13

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA red: it conflicts with NSW and WA.

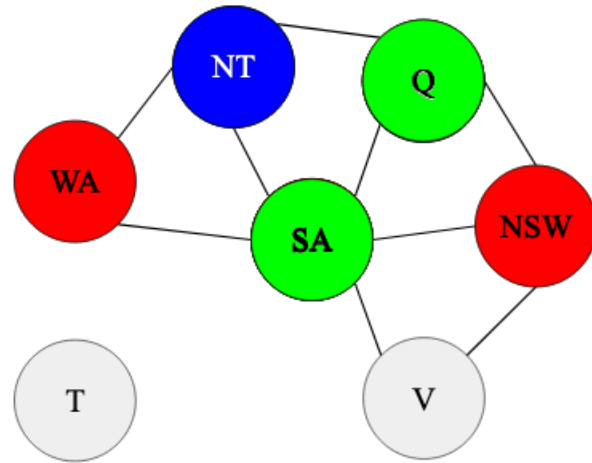
Backtracking

Slide 115 of 193 · Reveal 14

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA green: it conflicts with Q.

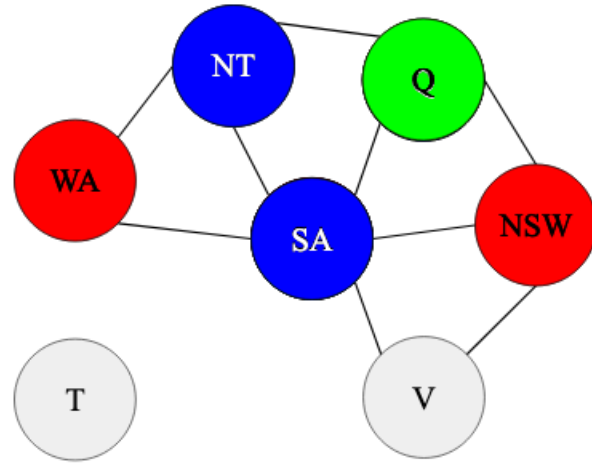
Backtracking

Slide 115 of 193 · Reveal 15

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA blue: it conflicts with NT. Backtrack to Q.

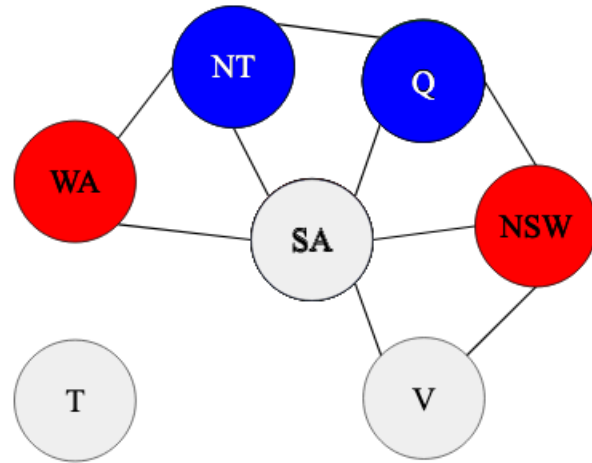
Backtracking

Slide 115 of 193 · Reveal 16

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject Q blue: it conflicts with NT. Q has no further value; backtrack to WA.

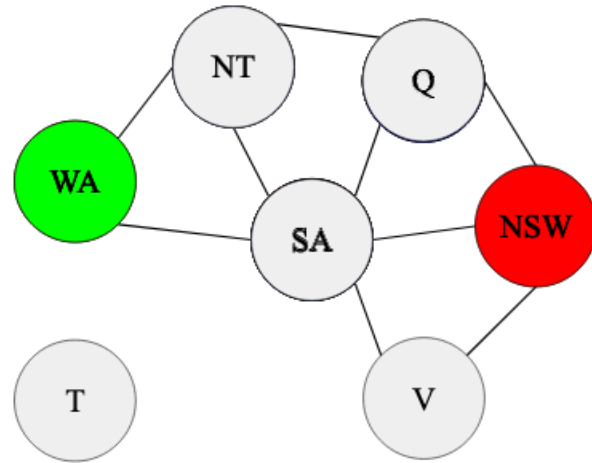
Backtracking

Slide 115 of 193 · Reveal 17

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Restore NT and Q to unassigned. Try WA green.

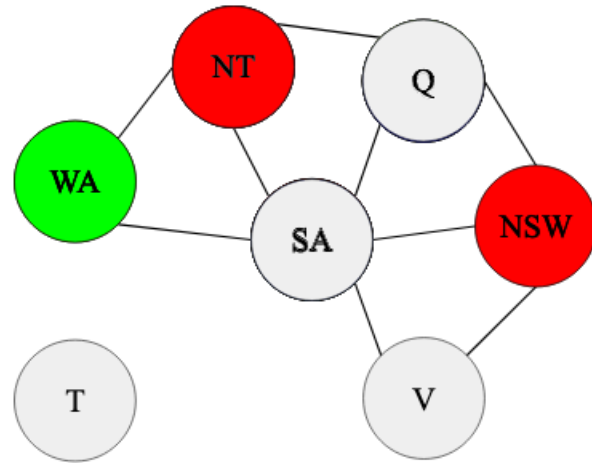
Backtracking

Slide 115 of 193 · Reveal 18

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign NT red.

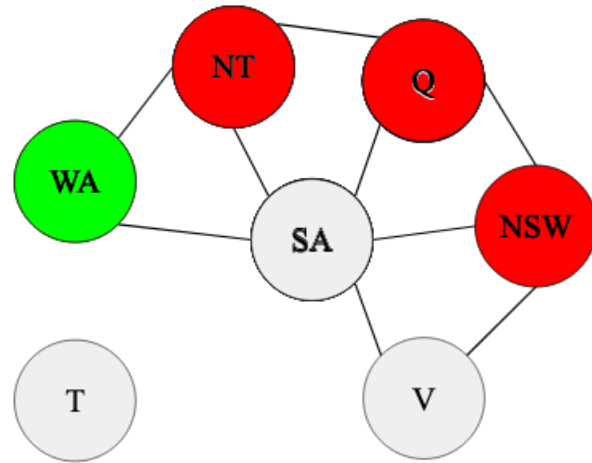
Backtracking

Slide 115 of 193 · Reveal 19

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject Q red: it conflicts with NSW and NT.

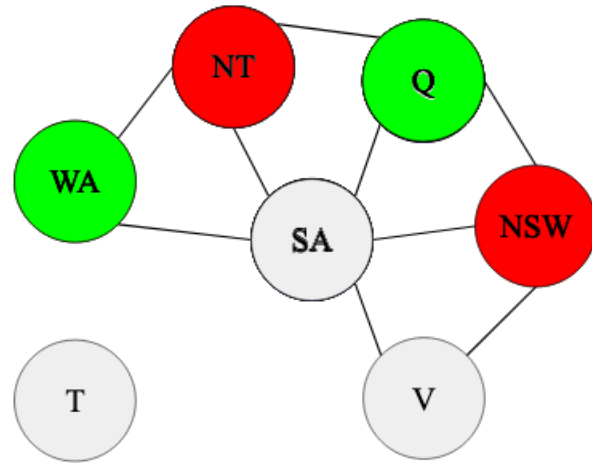
Backtracking

Slide 115 of 193 · Reveal 20

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign Q green.

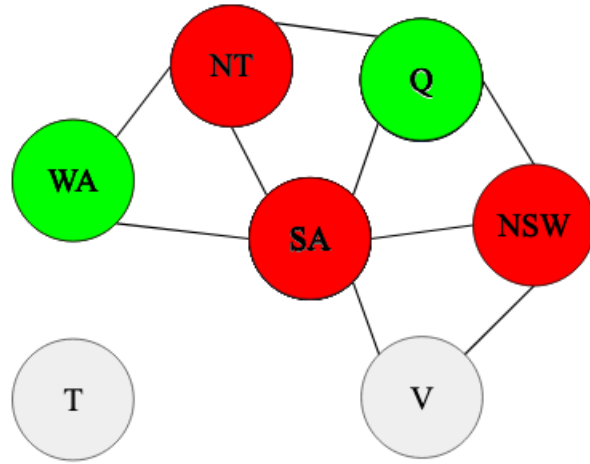
Backtracking

Slide 115 of 193 · Reveal 21

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA red: it conflicts with NSW and NT.

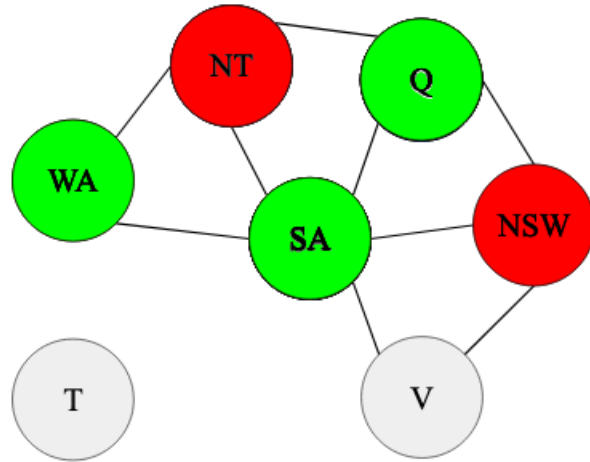
Backtracking

Slide 115 of 193 · Reveal 22

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject SA green: it conflicts with WA and Q.

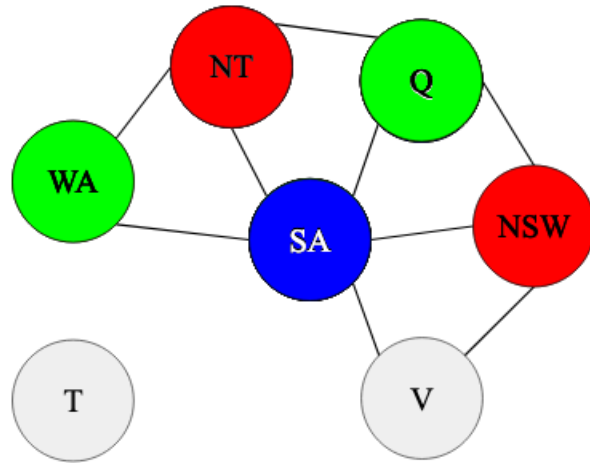
Backtracking

Slide 115 of 193 · Reveal 23

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign SA blue.

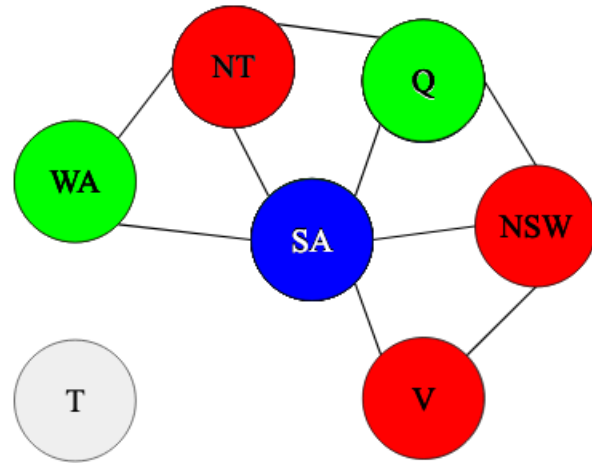
Backtracking

Slide 115 of 193 · Reveal 24

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Reject V red: it conflicts with NSW.

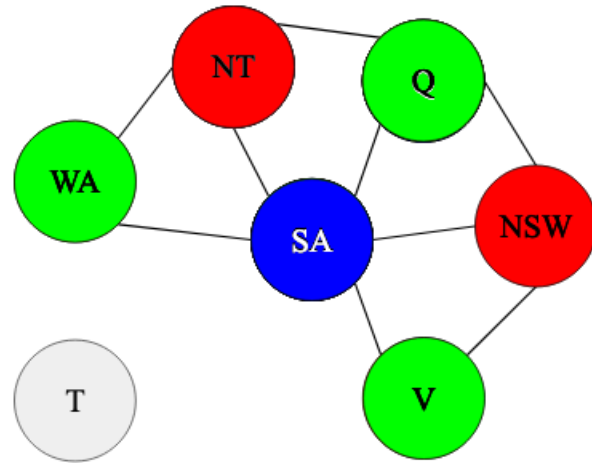
Backtracking

Slide 115 of 193 · Reveal 25

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign V green.

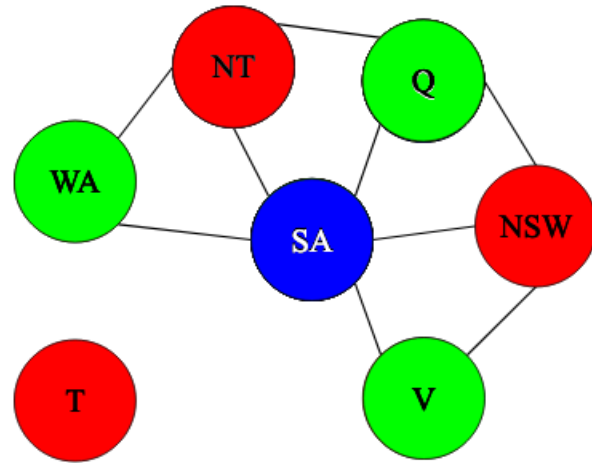
Backtracking

Slide 115 of 193 · Reveal 26

Backtracking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



115

Speaker notes

Assign T red. The complete assignment satisfies every constraint.

Backtracking

Slide 115 of 193 · Reveal 27

Backtracking Search

Choose a variable → Try a consistent value → Recurse → Undo

```

1 def solve(domains, neighbours, assignment=None):
2     if assignment is None: assignment = {}
3     if len(assignment) == len(domains): return assignment.copy()
4     x = next(x for x in domains if x not in assignment)
5     for colour in domains[x]:
6         if all(assignment.get(y) != colour for y in neighbours[x]):
7             assignment[x] = colour
8             result = solve(domains, neighbours, assignment)
9             if result is not None: return result
10            del assignment[x]
11    return None
12 neighbours = {
13     'WA': ('NT', 'SA'), 'NT': ('WA', 'SA', 'Q'),
14     'SA': ('WA', 'NT', 'Q', 'NSW', 'V'),
15     'Q': ('NT', 'SA', 'NSW'), 'NSW': ('SA', 'Q', 'V'),
16     'V': ('SA', 'NSW'), 'T': ()}
17 order = ('NSW', 'WA', 'NT', 'Q', 'SA', 'V', 'T')
18 domains = {x: ['red', 'green', 'blue'] for x in order}

```

Download: esp_backtracking.py

116

Speaker notes

Run the three-colour map. A failed branch removes its assignment before trying the next colour. Two colours produce no solution.

[Sources] - Adapted from recovered COMP3270 source slide 106; runnable Australia map-colouring example

Backtracking Search

Slide 116 of 193 · Initial

Improving Backtracking

Can we detect inevitable failure earlier?

Forward checking: remove conflicting values from neighbours

Constraint propagation: repeat these deductions with AC-3

117

Improving Backtracking

Slide 117 of 193 · Initial

Speaker notes

Predict a failure before committing to the remaining variables.

[Sources] - Recovered COMP3270 2024–25 source slide 107:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>						

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>					

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>				

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>			

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>		

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>
<div><div></div></div>	<div><div></div></div>	<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Forward Checking

Forward Checking

After each assignment, remove conflicting values from the domains of unassigned neighbours



Detect failure earlier

WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

118

Speaker notes

Remove values only from neighbours of the newly assigned variable. An empty domain means failure.

[Sources] - Recovered COMP3270 2024–25 source slide 108: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

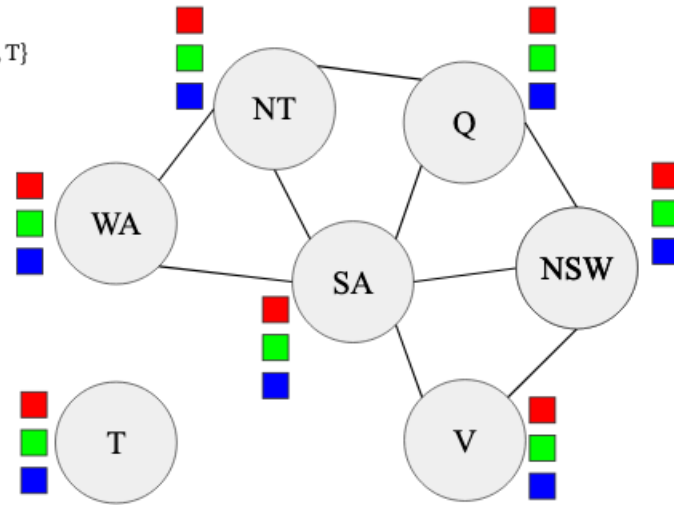
Forward Checking

Slide 118 of 193 · Reveal 13

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Follow the fixed variable and colour order. Chips show domains of unassigned variables only.

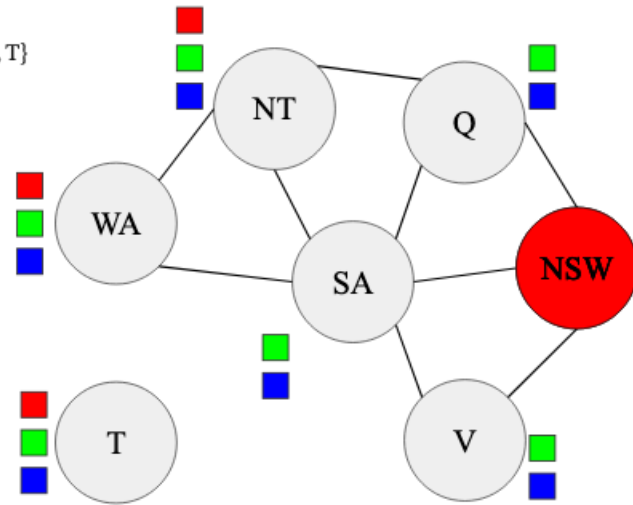
Backtracking with Forward Checking

Slide 119 of 193 · Initial

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign NSW red; remove red from Q, SA and V.

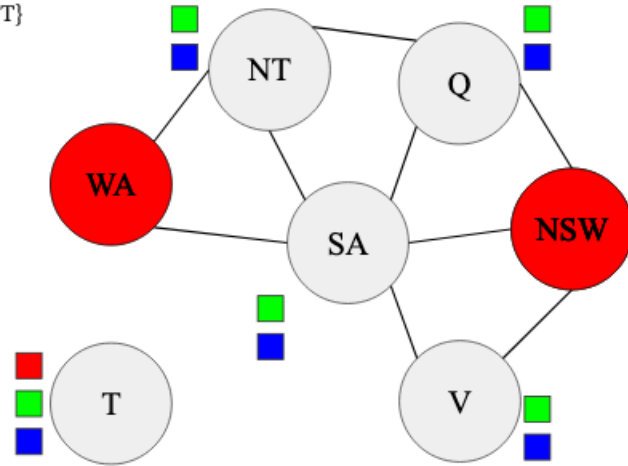
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 1

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign WA red; remove red from NT.

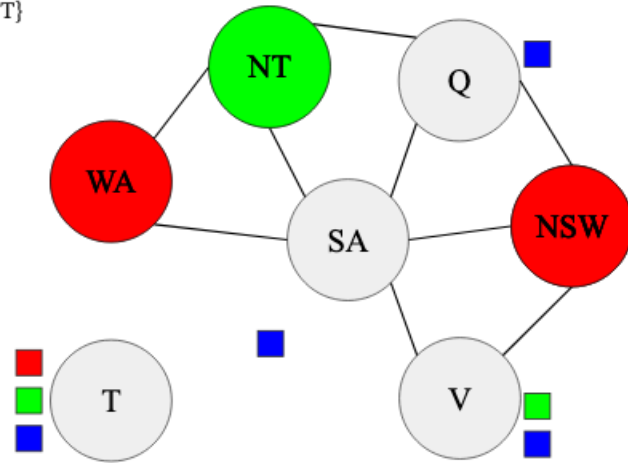
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 2

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign NT green. Q and SA now each have only blue.

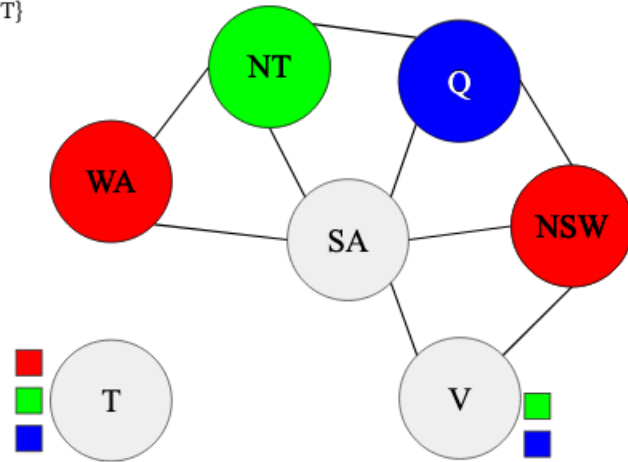
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 3

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Try Q blue. SA loses its last value, so this branch fails.

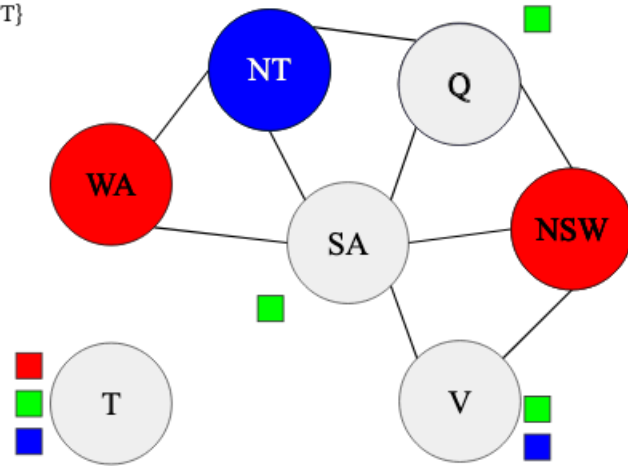
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 4

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Restore the failed branch. Try NT blue; Q and SA now each have only green.

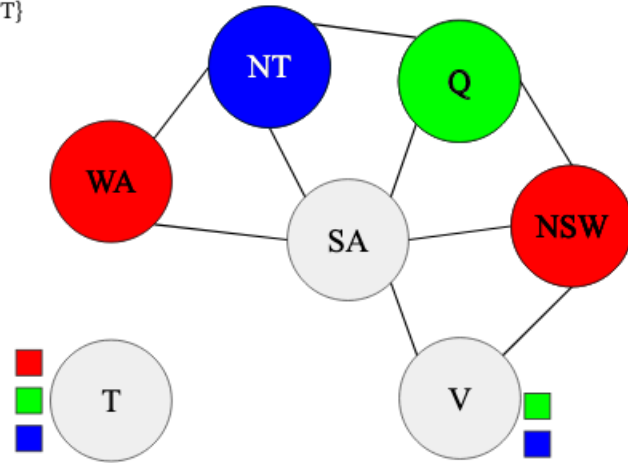
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 5

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Try Q green. SA loses its last value, so backtrack again.

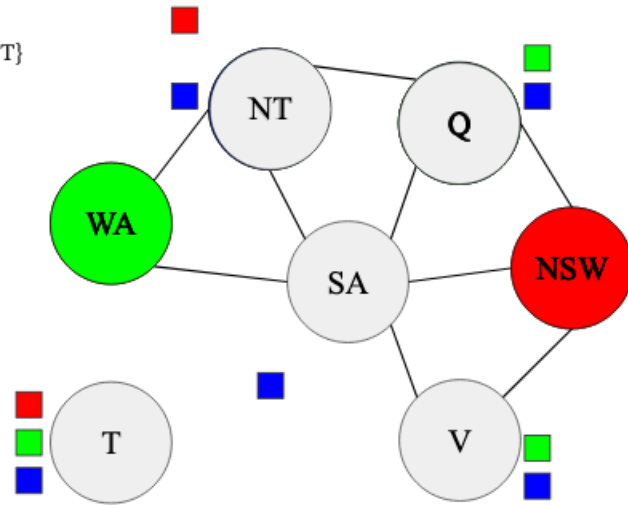
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 6

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Restore domains and unassign NT and Q. Try WA green.

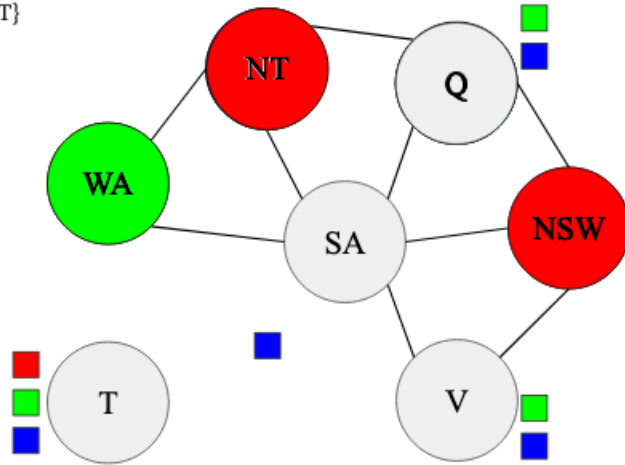
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 7

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Backtracking with Forward Checking

Slide 119 of 193 · Reveal 8

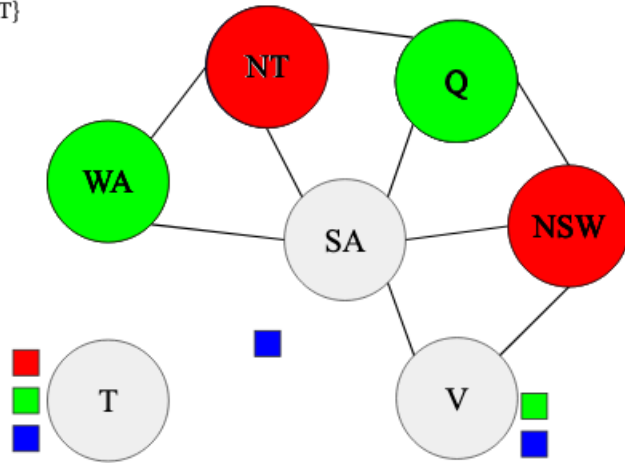
Speaker notes

Assign NT red. Q still has green or blue; SA has only blue.

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign Q green. SA remains blue.

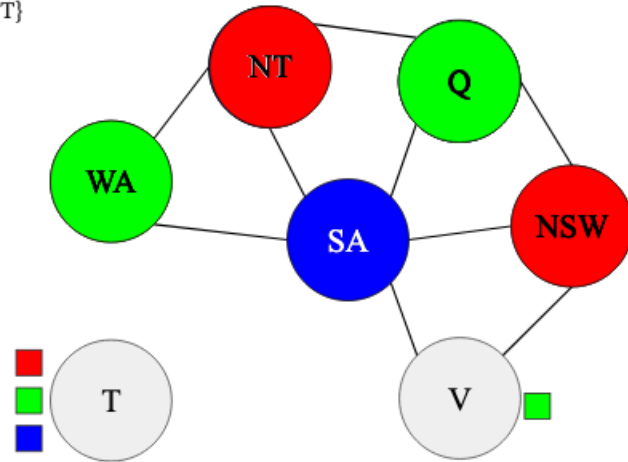
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 9

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign SA blue; V is now forced to green.

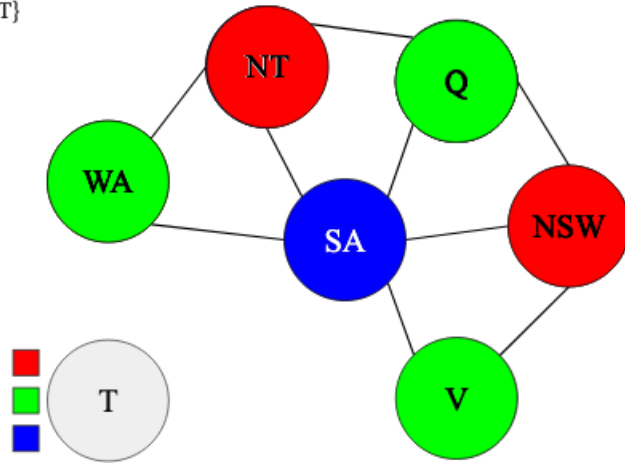
Backtracking with Forward Checking

Slide 119 of 193 · Reveal 10

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Backtracking with Forward Checking

Slide 119 of 193 · Reveal 11

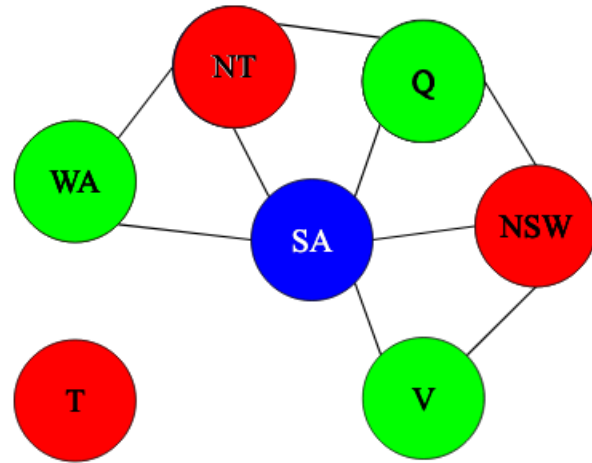
Speaker notes

Assign V green.

Backtracking with Forward Checking

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



119

Speaker notes

Assign T red. All constraints are satisfied.

Backtracking with Forward Checking

Slide 119 of 193 · Reveal 12

When Forward Checking Misses Failure

After each assignment, remove conflicting values from the domains of unassigned neighbours



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

120

Speaker notes

NT and SA both have only blue left. Their domains are nonempty, but their mutual constraint cannot be satisfied.

[Sources] - Recovered COMP3270 2024–25 source slide 110: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

When Forward Checking Misses Failure

When Forward Checking Misses Failure

After each assignment, remove conflicting values from the domains of unassigned neighbours



Doomed to fail

WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

120

Speaker notes

NT and SA both have only blue left. Their domains are nonempty, but their mutual constraint cannot be satisfied.

[Sources] - Recovered COMP3270 2024–25 source slide 110: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

When Forward Checking Misses Failure

Slide 120 of 193 · Reveal 1

When Forward Checking Misses Failure

After each assignment, remove conflicting values from the domains of unassigned neighbours



Doomed to fail

Can we detect this earlier?

WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

120

Speaker notes

NT and SA both have only blue left. Their domains are nonempty, but their mutual constraint cannot be satisfied.

[Sources] - Recovered COMP3270 2024–25 source slide 110: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

When Forward Checking Misses Failure

Slide 120 of 193 · Reveal 2

Consistency of an Arc

An arc $X \rightarrow Y$ is consistent if every $x \in D_X$ has a compatible $y \in D_Y$

$\forall x \in D_X, \exists y \in D_Y: C_{XY}(x, y)$

WA	NT	Q	NSW	V	SA	T
	  	  	  	  	  	  

121

Speaker notes

For each tail value, find support in the head. Remove a tail value with no support. The reverse arc is a separate check.

Consistency of an Arc An arc $X \rightarrow Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constrain 1

Consistency of an Arc

An arc $X \rightarrow Y$ is consistent if every $x \in D_X$ has a compatible $y \in D_Y$

$$\forall x \in D_X, \exists y \in D_Y: C_{XY}(x, y)$$

WA	NT	Q	NSW	V	SA	T
	  	  	  	  	  	  



121

Consistency of an Arc An arc $X \rightarrow Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constrain 1

Slide 121 of 193 · Reveal 1

Speaker notes

For each tail value, find support in the head. Remove a tail value with no support. The reverse arc is a separate check.

Consistency of an Arc

An arc $X \rightarrow Y$ is consistent if every $x \in D_X$ has a compatible $y \in D_Y$

$$\forall x \in D_X, \exists y \in D_Y: C_{XY}(x, y)$$

WA	NT	Q	NSW	V	SA	T
Red	Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue

121

Consistency of an Arc An arc $X \rightarrow Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constrain 1

Slide 121 of 193 · Reveal 2

Speaker notes

For each tail value, find support in the head. Remove a tail value with no support. The reverse arc is a separate check.

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

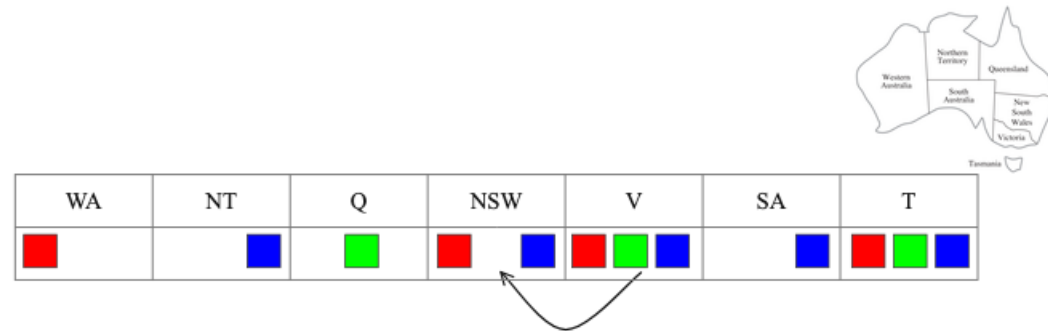
When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Initial

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 1

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

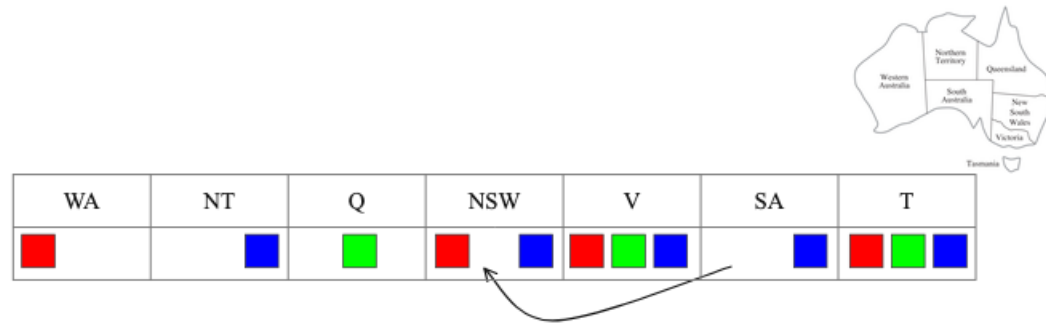
Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 3

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

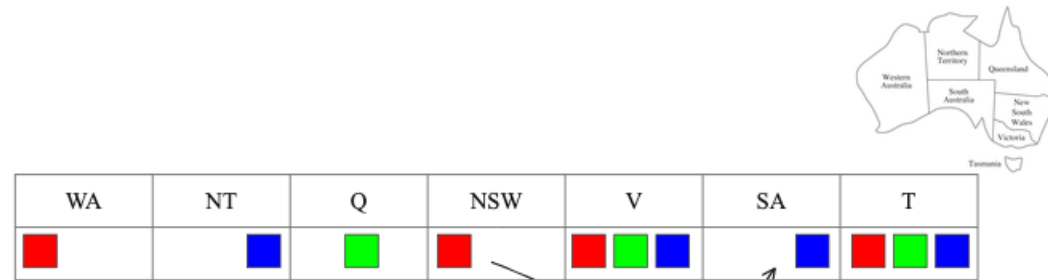
When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 5

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 6

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
■	■	■	■	■ ■ ■	■	■ ■ ■

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

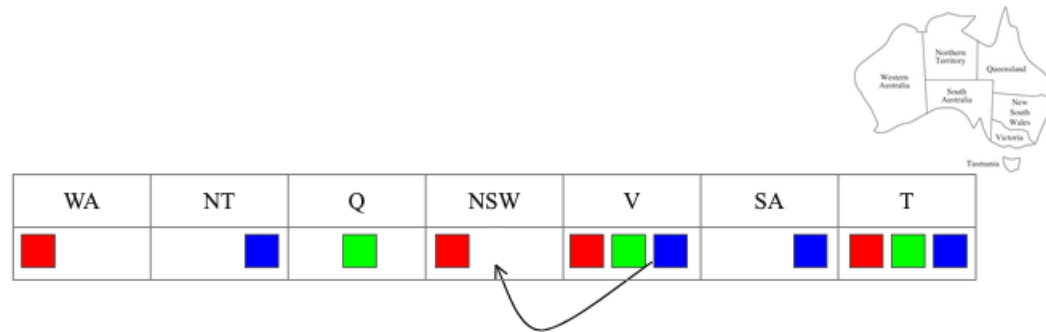
Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

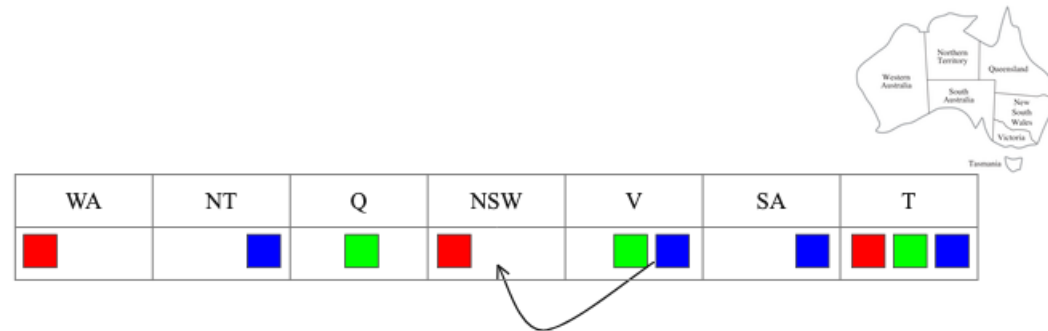
When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 8

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 9

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
Red	Blue	Green	Red	Green, Blue	Blue	Red, Green, Blue

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

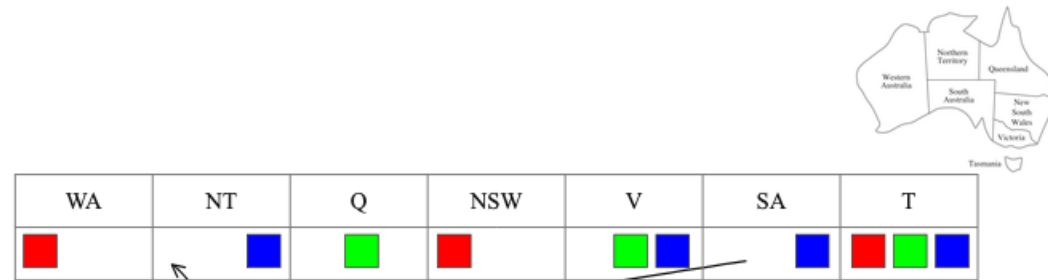
Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

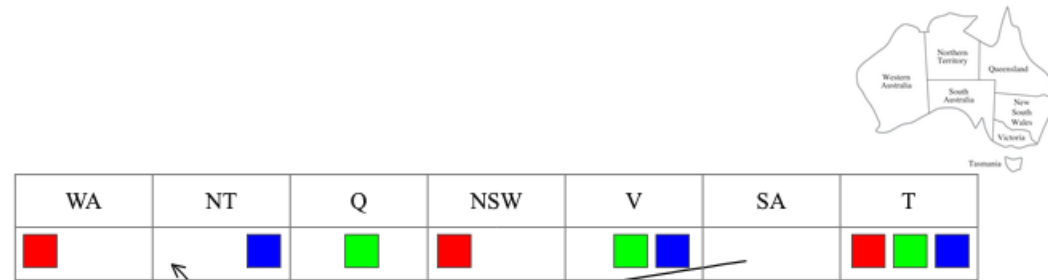
When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 11

Constraint Propagation



$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 12

Constraint Propagation



WA	NT	Q	NSW	V	SA	T
■	■	■	■	■ ■		■ ■ ■

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 13

Constraint Propagation

When a domain shrinks, recheck arcs pointing into it
Repeat until no domain changes, or a domain is empty



WA	NT	Q	NSW	V	SA	T

$X \rightarrow Y$: every value in D_X needs a compatible value in D_Y

122

Speaker notes

When a domain shrinks, recheck arcs pointing into that variable. Repeat until stable or a domain is empty.

[Sources] - Recovered COMP3270 2024–25 source slide 112:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Constraint Propagation

Slide 122 of 193 · Reveal 14

AC-3

Revise an arc → Revisit affected incoming arcs

```

1 from collections import deque
2 def ac3(domains, neighbours):
3     if any(not values for values in domains.values()): return False
4     queue = deque((x, y) for x in neighbours for y in neighbours[x])
5     while queue:
6         x, y = queue.popleft()
7         if revise(domains, x, y):
8             if not domains[x]: return False
9             queue.extend((z, x) for z in neighbours[x] if z != y)
10    return True
11 def revise(domains, x, y):
12     kept = [a for a in domains[x]
13             if any(a != b for b in domains[y])]
14     changed = kept != domains[x]
15     domains[x] = kept
16     return changed
17 neighbours = {
18     'WA': ('NT', 'SA'), 'NT': ('WA', 'SA', 'Q'),

```

Download: csp_ac3.py

123

Speaker notes

Run the restricted WA and NT domains. When x shrinks, revisit (z, x), not (x, z). True means no empty domain was found, not that a solution is guaranteed.

[Sources] - Adapted from recovered COMP3270 source slide 113; runnable Australia map-colouring example

AC-3

Slide 123 of 193 · Initial

AC-3

Why exclude
the reverse arc?

Revise an arc → Revisit affected incoming arcs

```

1 from collections import deque
2 def ac3(domains, neighbours):
3     if any(not values for values in domains.values()): return False
4     queue = deque((x, y) for x in neighbours for y in neighbours[x])
5     while queue:
6         x, y = queue.popleft()
7         if revise(domains, x, y):
8             if not domains[x]: return False
9             queue.extend((z, x) for z in neighbours[x] if z != y)
10    return True
11 def revise(domains, x, y):
12     kept = [a for a in domains[x]
13             if any(a != b for b in domains[y])]
14     changed = kept != domains[x]
15     domains[x] = kept
16     return changed
17 neighbours = {
18     'WA': ('NT', 'SA'), 'NT': ('WA', 'SA', 'Q'),

```

Download: csp_ac3.py

123

Speaker notes

Run the restricted WA and NT domains. When x shrinks, revisit (z, x), not (x, z). True means no empty domain was found, not that a solution is guaranteed.

[Sources] - Adapted from recovered COMP3270 source slide 113; runnable Australia map-colouring example

AC-3

Slide 123 of 193 · Reveal 1

Why Exclude the Reverse Arc?

124

Why Exclude the Reverse Arc?

Slide 124 of 193 · Initial

Speaker notes

A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?



124

Why Exclude the Reverse Arc?

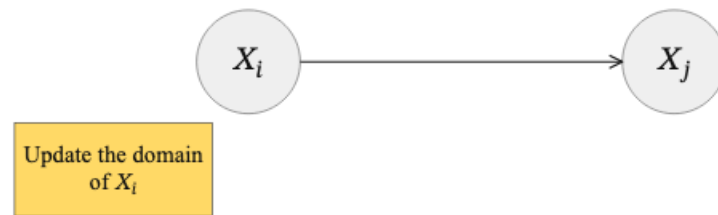
Slide 124 of 193 · Reveal 1

Speaker notes

A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?



124

Speaker notes

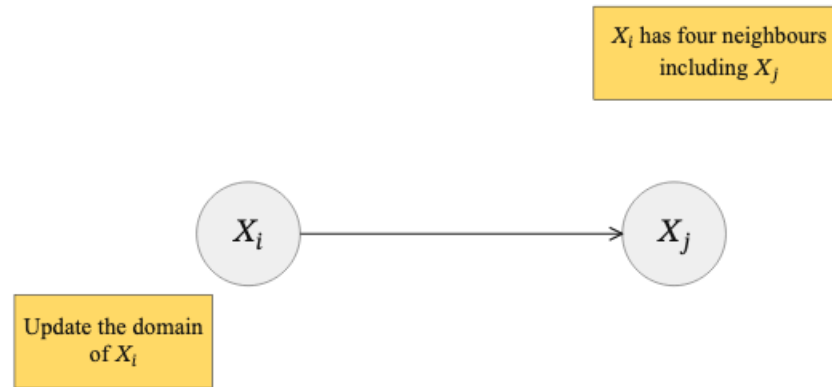
A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?

Slide 124 of 193 · Reveal 2

Why Exclude the Reverse Arc?



124

Why Exclude the Reverse Arc?

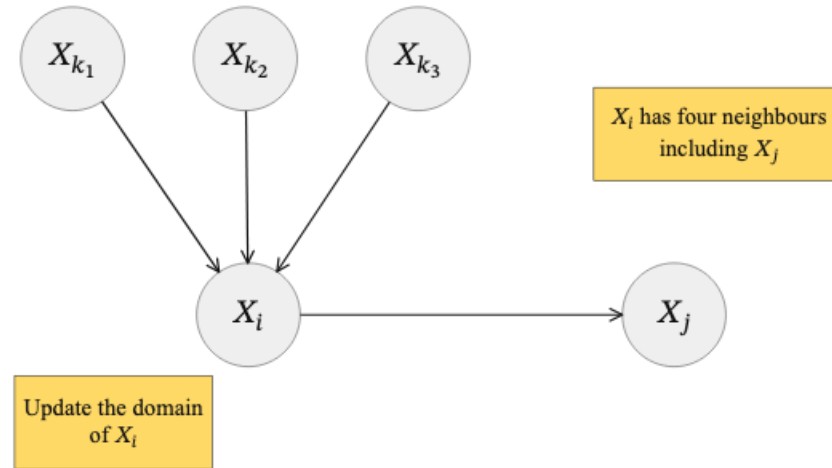
Slide 124 of 193 · Reveal 3

Speaker notes

A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?



124

Why Exclude the Reverse Arc?

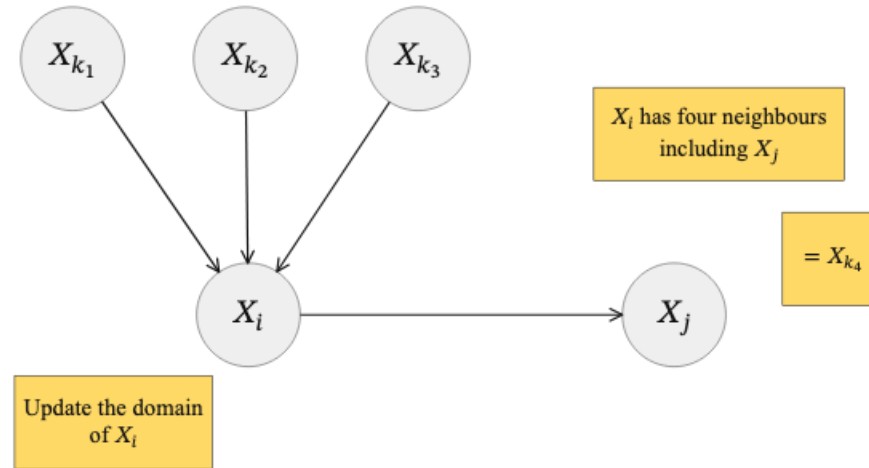
Slide 124 of 193 · Reveal 4

Speaker notes

A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?



124

Speaker notes

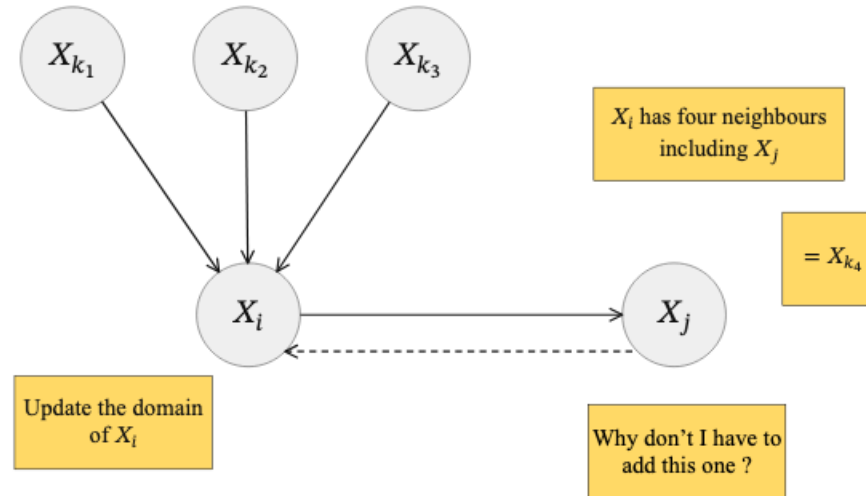
A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?

Slide 124 of 193 · Reveal 5

Why Exclude the Reverse Arc?



124

Speaker notes

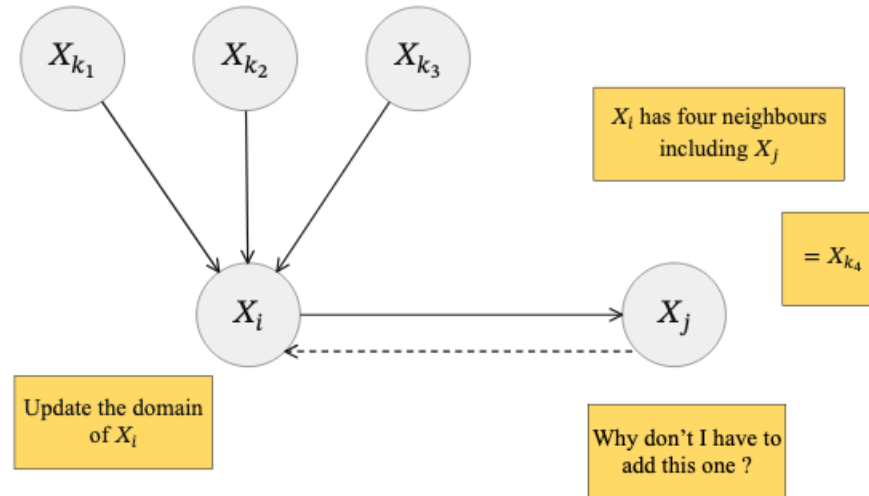
A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?

Slide 124 of 193 · Reveal 6

Why Exclude the Reverse Arc?



124

Speaker notes

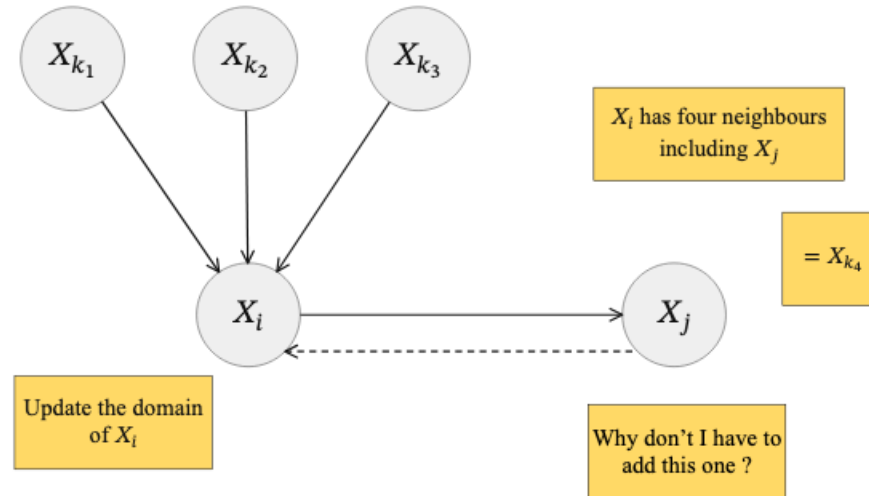
A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

Why Exclude the Reverse Arc?

Slide 124 of 193 · Reveal 7

Why Exclude the Reverse Arc?



124

Speaker notes

A value removed from X_i had no support in X_j . It therefore could not support a value of X_j under the same binary constraint.

[Sources] - Recovered COMP3270 2024–25 source slide 114: 01_search.pptx. - Canonical visual layer: 01_search.pdf.

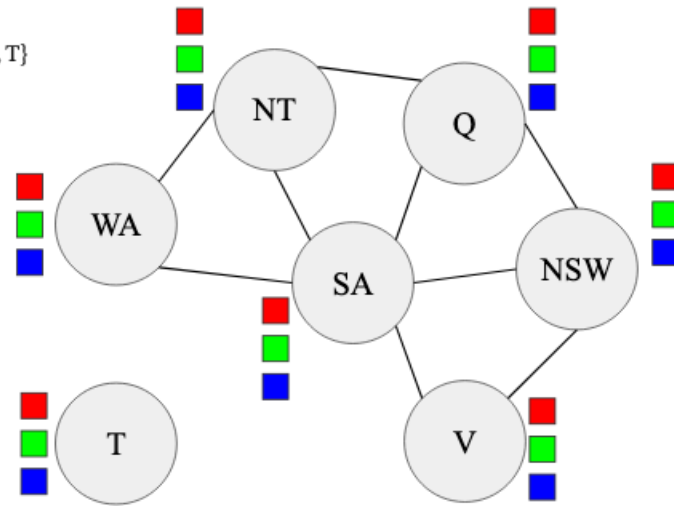
Why Exclude the Reverse Arc?

Slide 124 of 193 · Reveal 8

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Use the same variable and colour order. Chips show AC-3 domains for unassigned variables; restore them on backtracking.

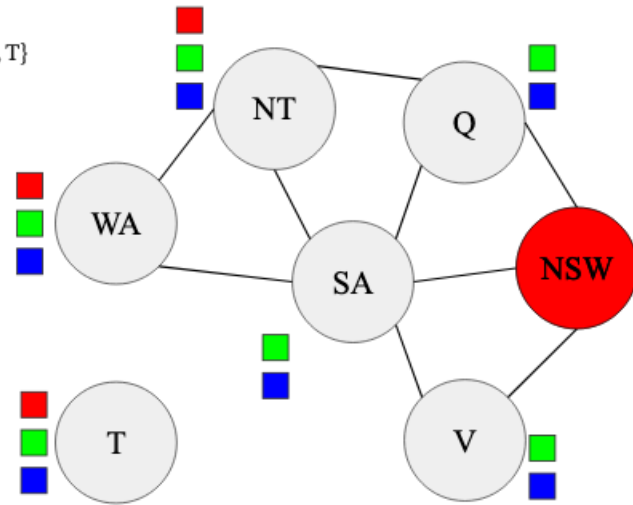
Backtracking with AC-3

Slide 125 of 193 · Initial

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign NSW red. AC-3 removes red from its neighbours.

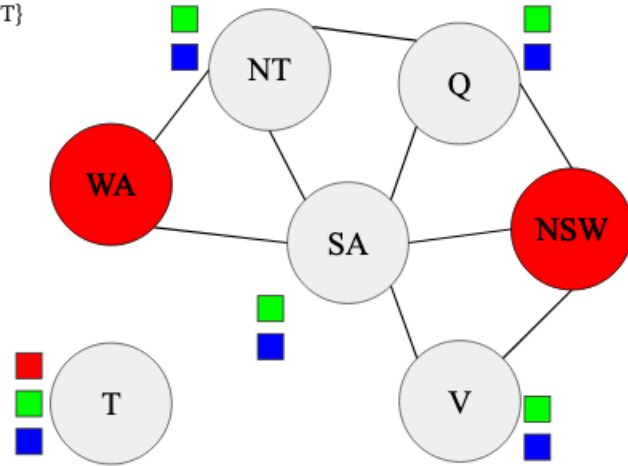
Backtracking with AC-3

Slide 125 of 193 · Reveal 1

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign WA red. NT, Q and SA retain green or blue; arc consistency has not detected the failure.

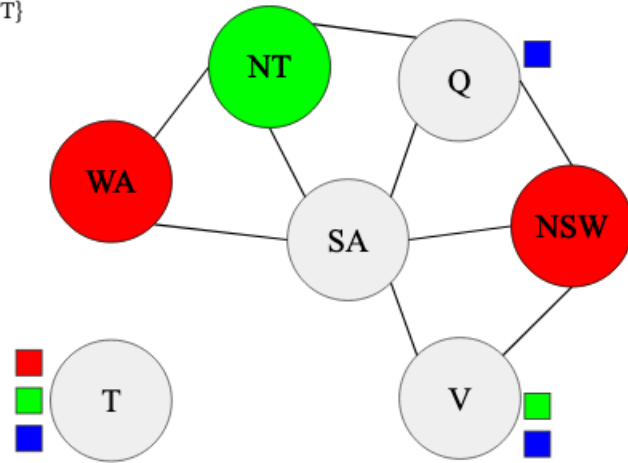
Backtracking with AC-3

Slide 125 of 193 · Reveal 2

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Try NT green. Propagation empties a domain before Q is assigned; backtrack.

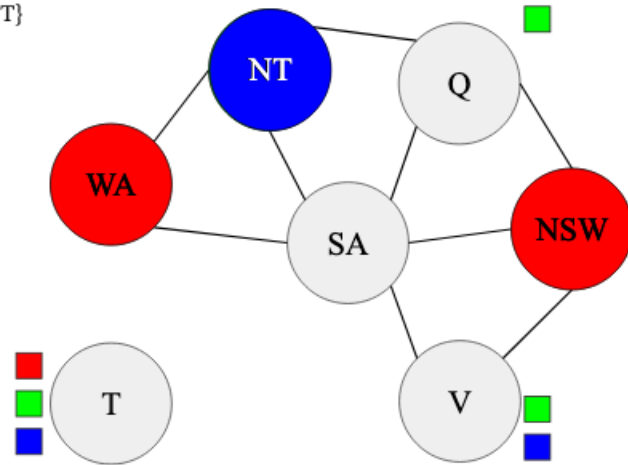
Backtracking with AC-3

Slide 125 of 193 · Reveal 3

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Restore domains and try NT blue. Propagation again empties a domain.

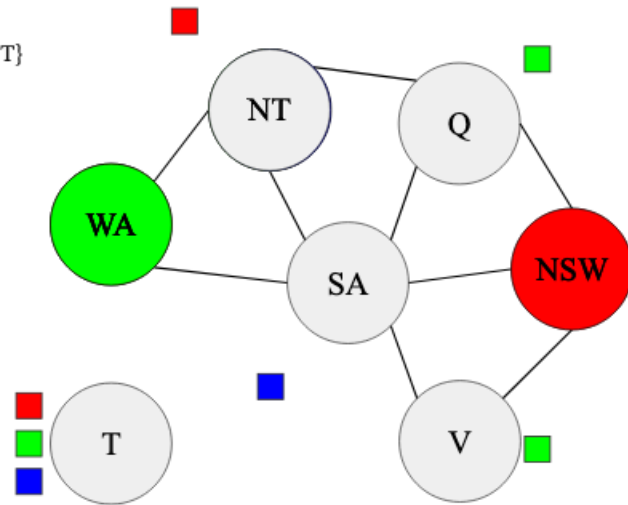
Backtracking with AC-3

Slide 125 of 193 · Reveal 4

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Restore domains, unassign NT and try WA green. AC-3 forces the remaining mainland colours.

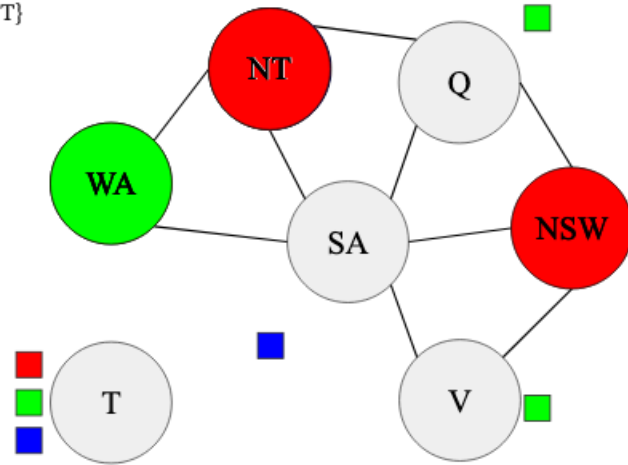
Backtracking with AC-3

Slide 125 of 193 · Reveal 5

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign the forced red value to NT.

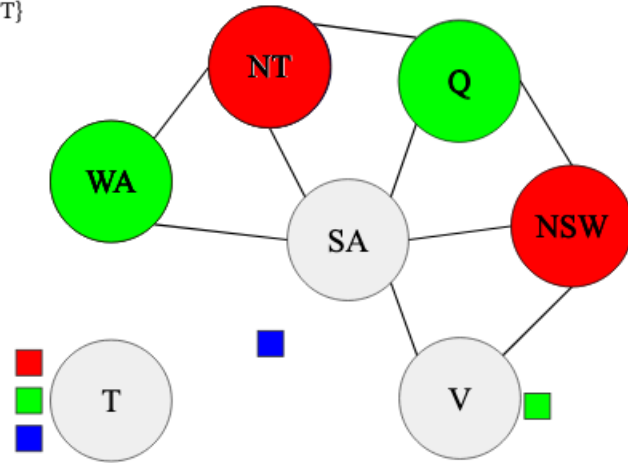
Backtracking with AC-3

Slide 125 of 193 · Reveal 6

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign the forced green value to Q.

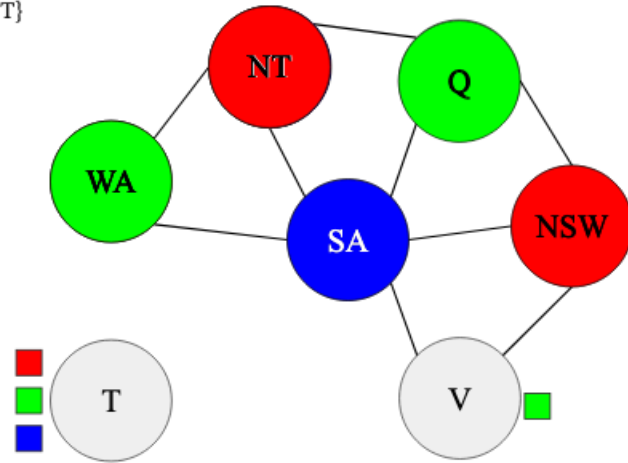
Backtracking with AC-3

Slide 125 of 193 · Reveal 7

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign the forced blue value to SA.

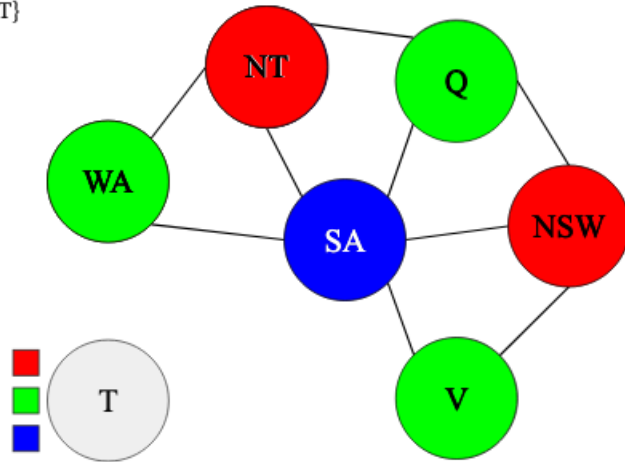
Backtracking with AC-3

Slide 125 of 193 · Reveal 8

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Backtracking with AC-3

Slide 125 of 193 · Reveal 9

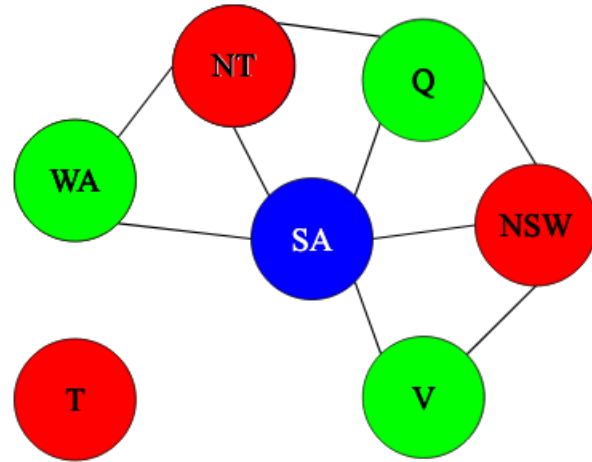
Speaker notes

Assign the forced green value to V.

Backtracking with AC-3

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



125

Speaker notes

Assign T red. The complete assignment is consistent.

Backtracking with AC-3

Slide 125 of 193 · Reveal 10

Variable and Value Ordering

126

Variable and Value Ordering

Slide 126 of 193 · Initial

Speaker notes

MRV chooses a variable, degree breaks ties, and LCV chooses a value.

[Sources] - Recovered COMP3270 2024–25 source slide 116:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Variable and Value Ordering

Variable Ordering

Minimum Remaining Values (MRV)

Choose the variable with the fewest legal values

Most constrained variable

Fail-first heuristic

126

Speaker notes

MRV chooses a variable, degree breaks ties, and LCV chooses a value.

[Sources] - Recovered COMP3270 2024–25 source slide 116:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Variable and Value Ordering

Slide 126 of 193 · Reveal 1

Variable and Value Ordering

Variable Ordering

Minimum Remaining Values (MRV)

- Choose the variable with the fewest legal values

 - Most constrained variable

 - Fail-first heuristic

Break MRV Ties

Degree Heuristic

- Most constraints on unassigned variables

126

Speaker notes

MRV chooses a variable, degree breaks ties, and LCV chooses a value.

[Sources] - Recovered COMP3270 2024–25 source slide 116:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Variable and Value Ordering

Slide 126 of 193 · Reveal 2

Variable and Value Ordering

Variable Ordering

Minimum Remaining Values (MRV)

- Choose the variable with the fewest legal values

 - Most constrained variable

 - Fail-first heuristic

Break MRV Ties

- Degree Heuristic

 - Most constraints on unassigned variables

Value Ordering

Least Constraining Value (LCV)

- Rules out the fewest values in unassigned neighbours

126

Speaker notes

MRV chooses a variable, degree breaks ties, and LCV chooses a value.

[Sources] - Recovered COMP3270 2024–25 source slide 116:
01_search.pptx. - Canonical visual layer: 01_search.pdf.

Variable and Value Ordering

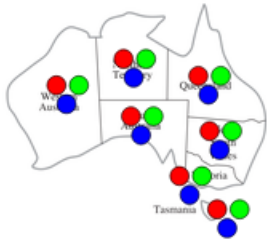
Slide 126 of 193 · Reveal 3

Variable Ordering (MRV)

Minimum Remaining Values

Choose the unassigned variable with the fewest legal values

Most constrained variable · Fail-first heuristic



Using Forward Checking

127

Speaker notes

Which unassigned variable has the fewest legal colours? After WA, NT and SA tie on domain size. After NT, SA has only one legal colour.

Variable Ordering (MRV) Minimum remaining values Choose the variable with the fewest legal left values in its domain Most constrained variable Fail-first heuristic

Slide 127 of 193 · Initial

Variable Ordering (MRV)

Minimum Remaining Values

Choose the unassigned variable with the fewest legal values

Most constrained variable · Fail-first heuristic



Using Forward Checking

127

Speaker notes

Which unassigned variable has the fewest legal colours? After WA, NT and SA tie on domain size. After NT, SA has only one legal colour.

Variable Ordering (MRV) Minimum remaining values Choose the variable with the fewest legal left values in its domain Most constrained variable Fail-first heuris

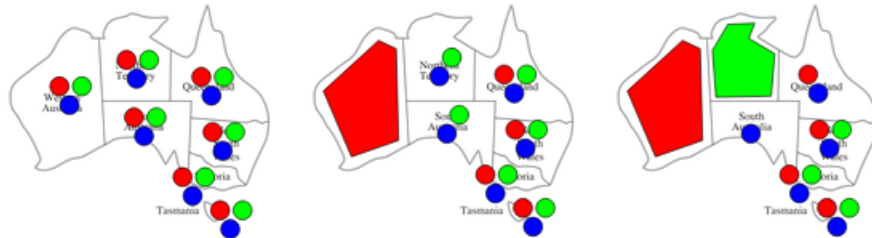
Slide 127 of 193 · Reveal 1

Variable Ordering (MRV)

Minimum Remaining Values

Choose the unassigned variable with the fewest legal values

Most constrained variable · Fail-first heuristic



Using Forward Checking

127

Speaker notes

Which unassigned variable has the fewest legal colours? After WA, NT and SA tie on domain size. After NT, SA has only one legal colour.

Variable Ordering (MRV) Minimum remaining values Choose the variable with the fewest legal left values in its domain Most constrained variable Fail-first heuristic

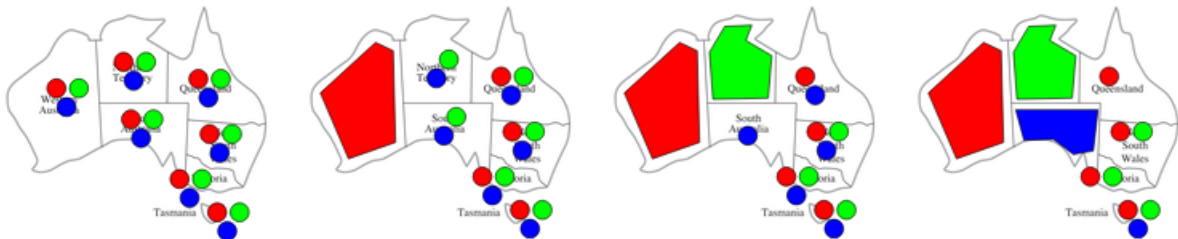
Slide 127 of 193 · Reveal 2

Variable Ordering (MRV)

Minimum Remaining Values

Choose the unassigned variable with the fewest legal values

Most constrained variable · Fail-first heuristic



Using Forward Checking

127

Speaker notes

Which unassigned variable has the fewest legal colours? After WA, NT and SA tie on domain size. After NT, SA has only one legal colour.

Variable Ordering (MRV) Minimum remaining values Choose the variable with the fewest legal left values in its domain Most constrained variable Fail-first heuris

Variable Ordering (MRV + Deg)

Degree Heuristic

Among MRV ties, choose the variable with the most constraints on unassigned variables

128

Speaker notes

Start with equal domains. Which variable constrains the most others? SA has five unassigned neighbours.

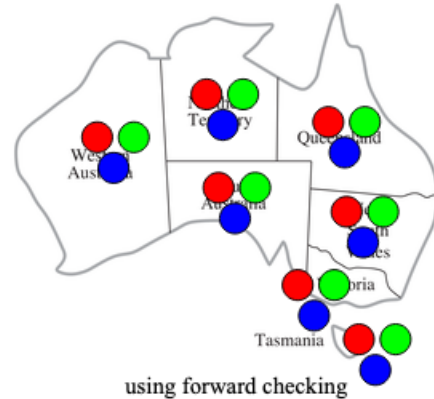
Variable Ordering (MRV + Deg) Degree Heuristic (Deg) Choose the variable with the most constraints on remaining variables 118 using forward checking D = {red, g

Slide 128 of 193 · Initial

Variable Ordering (MRV + Deg)

Degree Heuristic

Among MRV ties, choose the variable with the most constraints on unassigned variables



128

Speaker notes

Start with equal domains. Which variable constrains the most others? SA has five unassigned neighbours.

Variable Ordering (MRV + Deg) Degree Heuristic (Deg) Choose the variable with the most constraints on remaining variables 118 using forward checking D = {red, g

Variable Ordering (MRV + Deg)

Degree Heuristic

Among MRV ties, choose the variable with the most constraints on unassigned variables



128

Speaker notes

Start with equal domains. Which variable constrains the most others? SA has five unassigned neighbours.

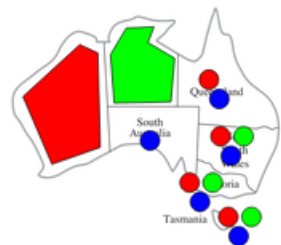
Variable Ordering (MRV + Deg) Degree Heuristic (Deg) Choose the variable with the most constraints on remaining variables 118 using forward checking $D = \{\text{red, g}\}$

Value Ordering (LCV)

Least Constraining Value

Try the value that removes the fewest options from unassigned neighbours

Assign Q Next
Which Colour First?



129

Speaker notes

Choose Q next. Which colour leaves more options? Compare the domains left by each value. Choose the colour that removes fewer neighbour values.

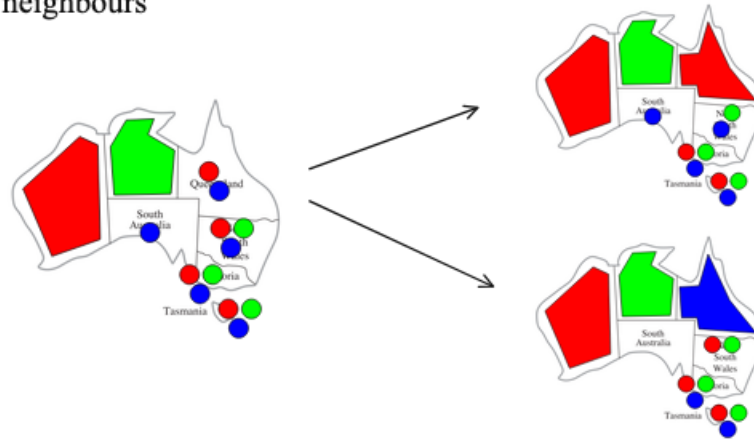
Value Ordering (LCV) Choose the value that rules out the fewest values in the remaining variables
Least constraining value (LCV) 119 Suppose we have decided to

Value Ordering (LCV)

Least Constraining Value

Try the value that removes the fewest options from unassigned neighbours

Assign Q Next
Which Colour First?



129

Speaker notes

Choose Q next. Which colour leaves more options? Compare the domains left by each value. Choose the colour that removes fewer neighbour values.

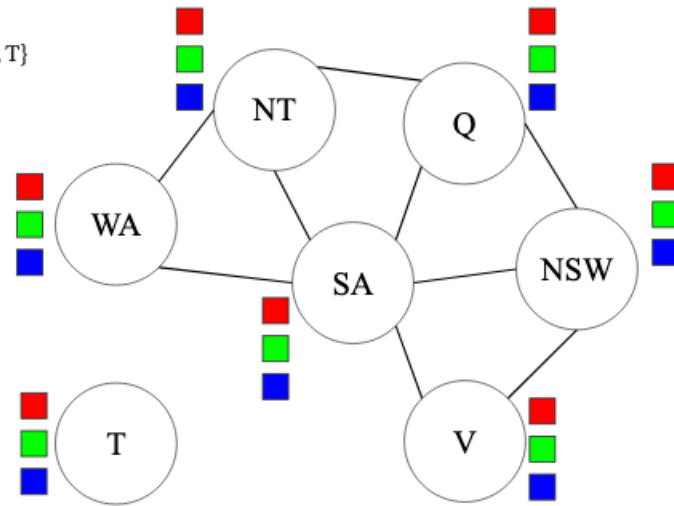
Value Ordering (LCV) Choose the value that rules out the fewest values in the remaining variables. Least constraining value (LCV) 119 Suppose we have decided to

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

Choose with MRV, then remaining degree, then the listed tie order. LCV chooses the colour; colour ties use red, green, blue.

Backtracking with All Heuristics

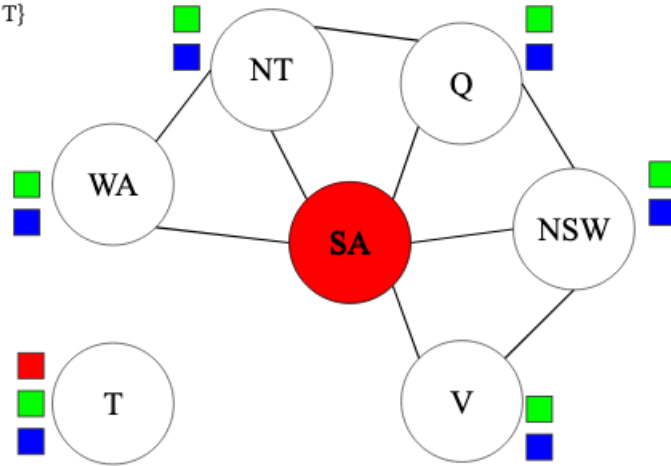
Slide 130 of 193 · Initial

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



Speaker notes

SA has degree 5. All colours tie under LCV, so choose red.

Backtracking with All Heuristics

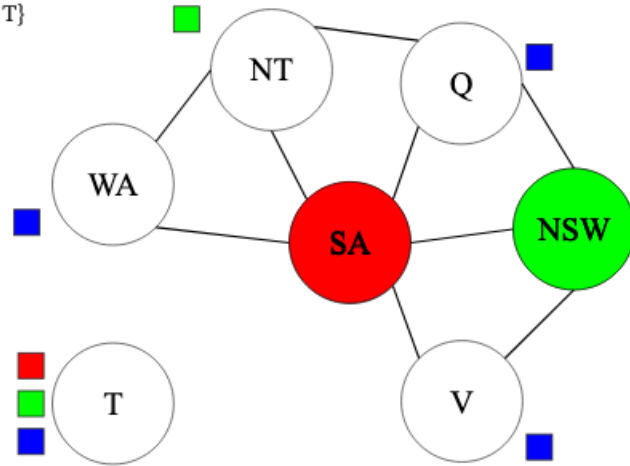
Slide 130 of 193 · Reveal 1

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

NSW wins the remaining degree tie. Green and blue tie under LCV. Choose green; AC-3 forces the other mainland domains.

Backtracking with All Heuristics

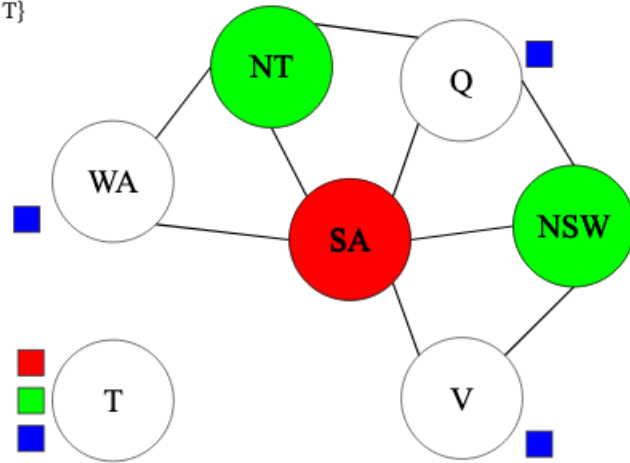
Slide 130 of 193 · Reveal 2

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

NT has two unassigned neighbours, WA and Q. MRV and degree therefore select NT next.

Backtracking with All Heuristics

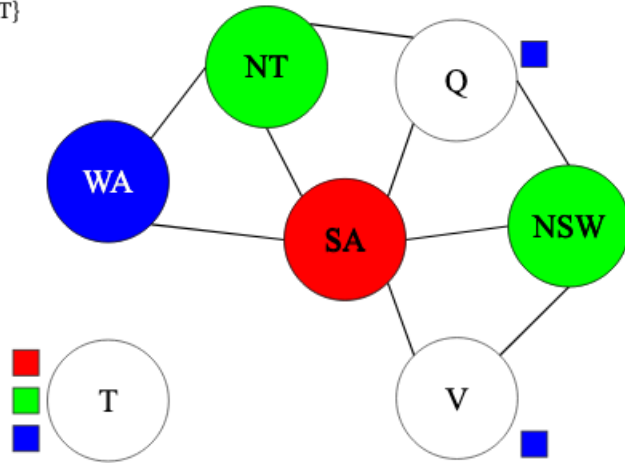
Slide 130 of 193 · Reveal 3

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

WA, Q and V are singleton domains with degree zero. The listed order selects WA.

Backtracking with All Heuristics

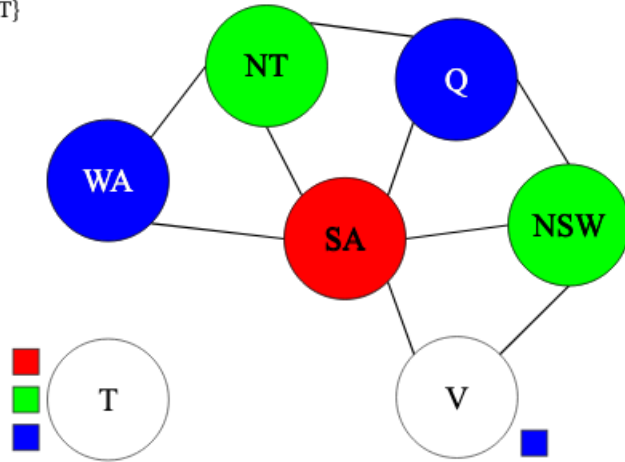
Slide 130 of 193 · Reveal 4

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

Q is next in the remaining singleton tie.

Backtracking with All Heuristics

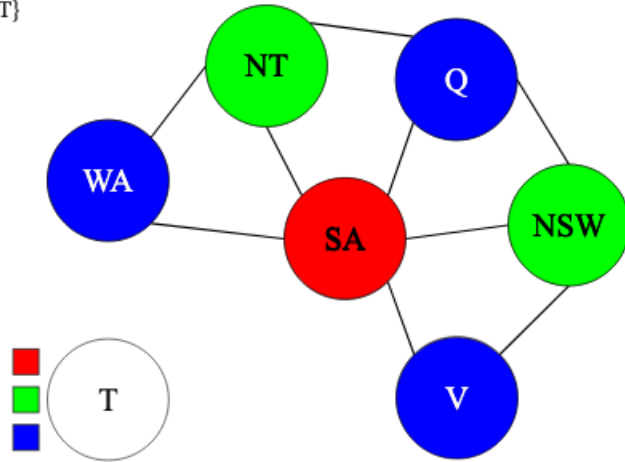
Slide 130 of 193 · Reveal 5

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

Speaker notes

Assign the forced blue value to V.

Backtracking with All Heuristics

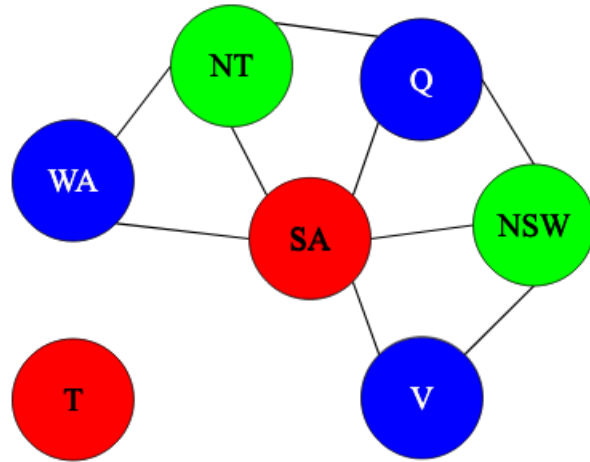
Slide 130 of 193 · Reveal 6

Backtracking with All Heuristics

AC-3 · MRV · Degree · LCV

Variables: {NSW, WA, NT, Q, SA, V, T}

Colors: {red, green, blue}



130

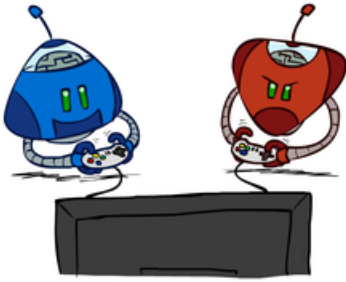
Speaker notes

T has no constraints. All colours tie, so choose red. The assignment is complete and consistent.

Backtracking with All Heuristics

Slide 130 of 193 · Reveal 7

Adversarial Search



131

Adversarial Search

Slide 131 of 193 · Initial

Speaker notes

So far, the environment did not choose actions against us. Now another player gets a turn.

Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*

132

Other Agents Choose Their Own Moves

Slide 132 of 193 · Initial

Speaker notes

Another agent chooses its own response to our move.

Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



132

Other Agents Choose Their Own Moves

Slide 132 of 193 · Reveal 1

Speaker notes

In chess, consider the opponent's reply before choosing a move.

Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



X	O	
	X	
	O	

132

Other Agents Choose Their Own Moves

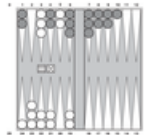
Slide 132 of 193 · Reveal 2

Speaker notes

Tic-tac-toe has the same alternating-turn structure on a smaller board.

Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



Chance: Later

X	O	
	X	
	O	

132

Speaker notes

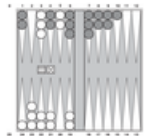
Backgammon adds chance. We will handle that later.

Other Agents Choose Their Own Moves

Slide 132 of 193 · Reveal 3

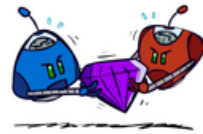
Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



Chance: Later

X	O	
	X	
	O	



132

Speaker notes

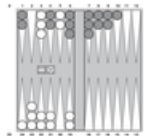
The agents' goals can conflict: they are competing for the same outcome.

Other Agents Choose Their Own Moves

Slide 132 of 193 · Reveal 4

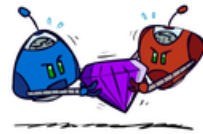
Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



Chance: Later

X	O	
	X	
	O	



Here: Two Players · Alternating Turns · Deterministic · Fully Observable

132

Speaker notes

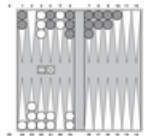
For now, focus on deterministic, fully observable games with two players taking turns.

Other Agents Choose Their Own Moves

Slide 132 of 193 · Reveal 5

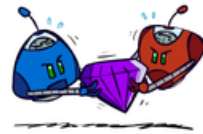
Other Agents Choose Their Own Moves

Our next state depends on our move *and their response*



Chance: Later

X	O	
	X	
	O	



Here: Two Players · Alternating Turns · Deterministic · Fully Observable

Zero-Sum: What Helps MAX Hurts MIN

132

Speaker notes

In a zero-sum game, what helps MAX hurts MIN. Other multi-agent problems need not be zero-sum.

Other Agents Choose Their Own Moves

Slide 132 of 193 · Reveal 6

Defining a Game

State s ; Initial State s_0	Board configuration <i>and player to move</i>
Player(s)	Whose turn is it?
Actions(s); Result(s, a)	Legal moves; state after a move
TerminalTest(s)	Has the game ended?
Utility(s, p)	Final payoff for player p

133

Speaker notes

Compare with our search-problem definition. The player to move belongs to the state. Utility is a terminal payoff, not a path cost.

Defining a Game

Slide 133 of 193 · Initial

Defining a Game

State s ; Initial State s_0	Board configuration <i>and player to move</i>
Player(s)	Whose turn is it?
Actions(s); Result(s, a)	Legal moves; state after a move
TerminalTest(s)	Has the game ended?
Utility(s, p)	Final payoff for player p

Game Tree: Nodes Are States; Edges Are Moves

133

Speaker notes

A strategy tells us what to do for each opponent response; a single path does not.

Defining a Game

Slide 133 of 193 · Reveal 1

Tic-Tac-Toe Game Tree



Players

X is MAX

O is MIN

Utility for X

Win: +1

Draw: 0

Loss: -1

134

Speaker notes

The empty board is the root. X moves first.

Tic-Tac-Toe Game Tree

Slide 134 of 193 · Initial

Tic-Tac-Toe Game Tree

Players

X is MAX

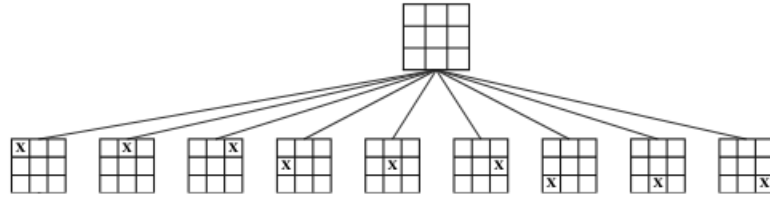
O is MIN

Utility for X

Win: +1

Draw: 0

Loss: -1



134

Speaker notes

Reveal X's nine possible opening moves. The diagram follows only the left branch after this level.

Tic-Tac-Toe Game Tree

Slide 134 of 193 · Reveal 1

Tic-Tac-Toe Game Tree

Players

X is MAX

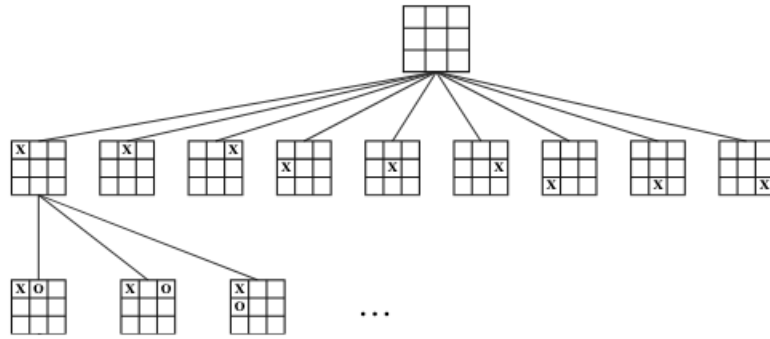
O is MIN

Utility for X

Win: +1

Draw: 0

Loss: -1



134

Speaker notes

O chooses a reply, so this level belongs to MIN.

Tic-Tac-Toe Game Tree

Slide 134 of 193 · Reveal 2

Tic-Tac-Toe Game Tree

Players

X is MAX

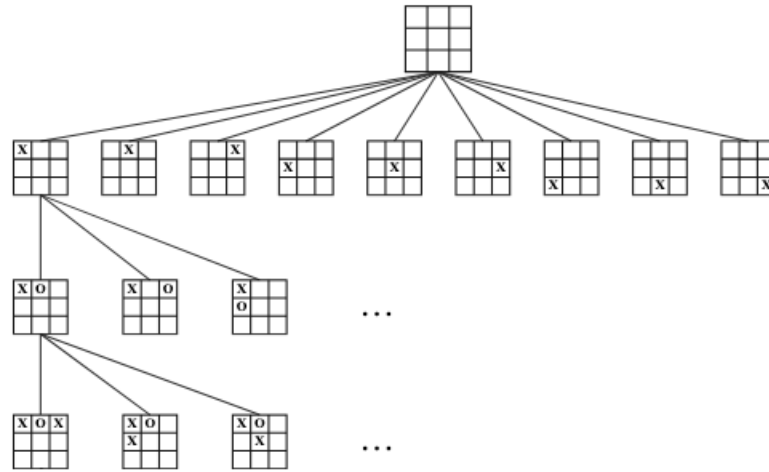
O is MIN

Utility for X

Win: +1

Draw: 0

Loss: -1



134

Speaker notes

X moves again. Each level includes the player whose turn comes next.

Tic-Tac-Toe Game Tree

Slide 134 of 193 · Reveal 3

Tic-Tac-Toe Game Tree

Players

X is MAX

O is MIN

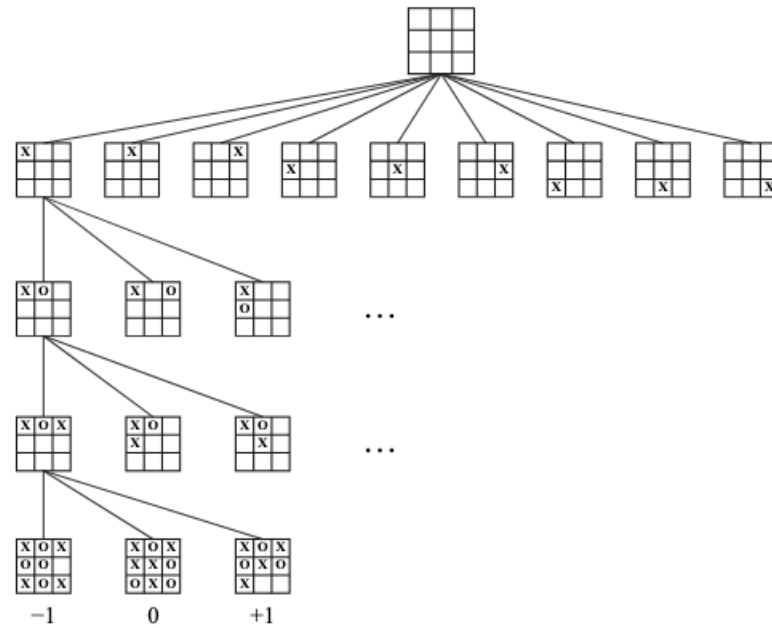
Utility for X

Win: +1

Draw: 0

Loss: -1

State Includes
the Player to
Move



134

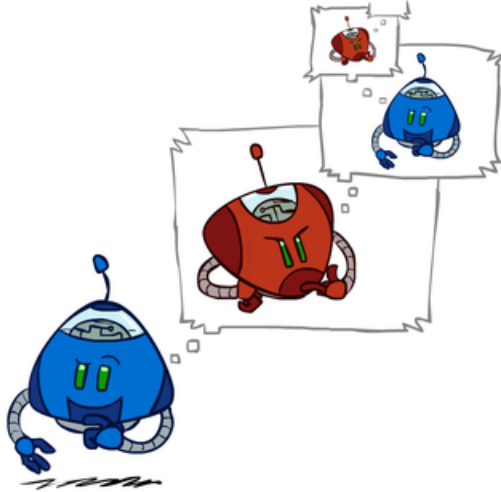
Speaker notes

The final row shows three possible terminal situations. Read every utility from X's perspective: loss, draw and win.

Tic-Tac-Toe Game Tree

Slide 134 of 193 · Reveal 4

Think About Their Reply



135

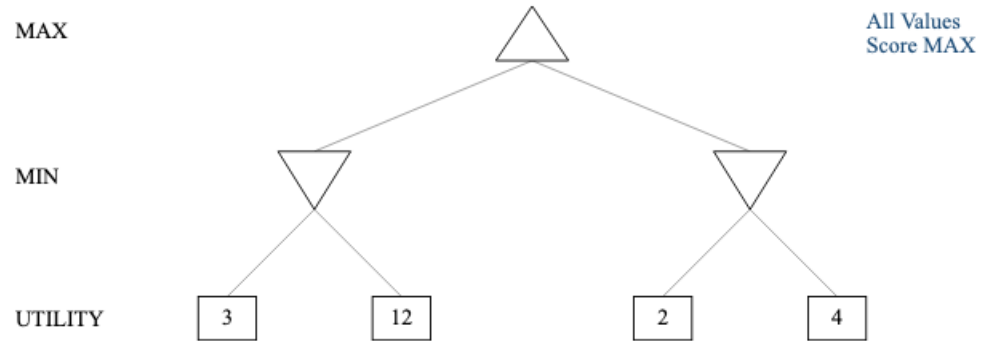
Think About Their Reply

Slide 135 of 193 · Initial

Speaker notes

If I move here, what can the opponent do? Then what can I do? The recursion alternates players.

Back Up Values from the Leaves



136

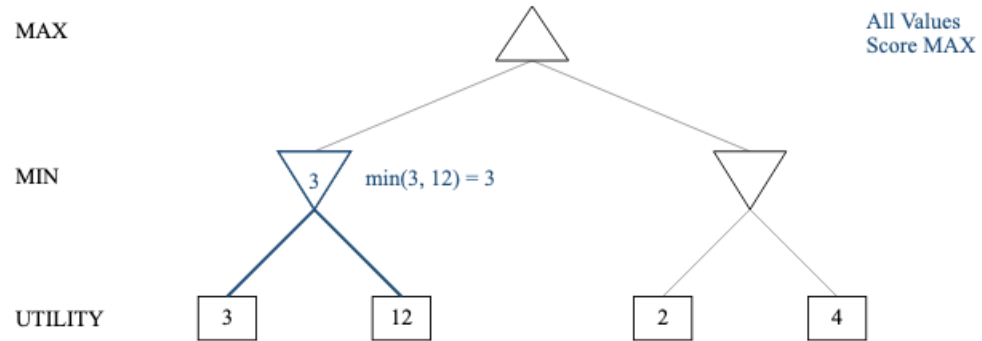
Back Up Values from the Leaves

Slide 136 of 193 · Initial

Speaker notes

Start at the terminal utilities. Every number scores the outcome for MAX.

Back Up Values from the Leaves



136

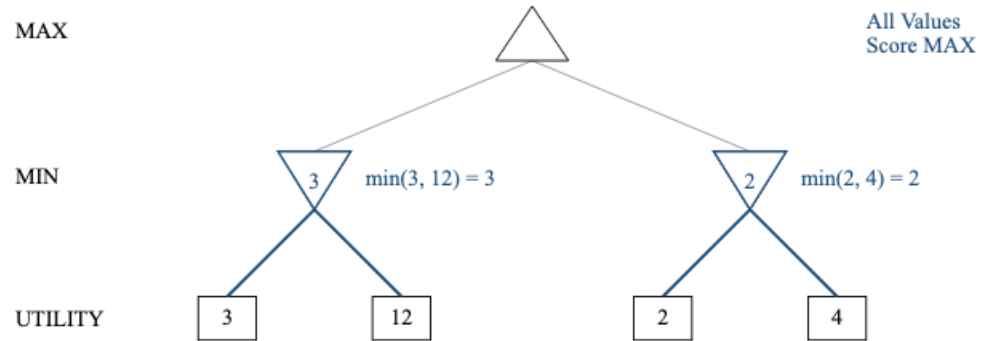
Speaker notes

The left MIN node chooses the smaller child: $\min(3, 12) = 3$.

Back Up Values from the Leaves

Slide 136 of 193 · Reveal 1

Back Up Values from the Leaves



136

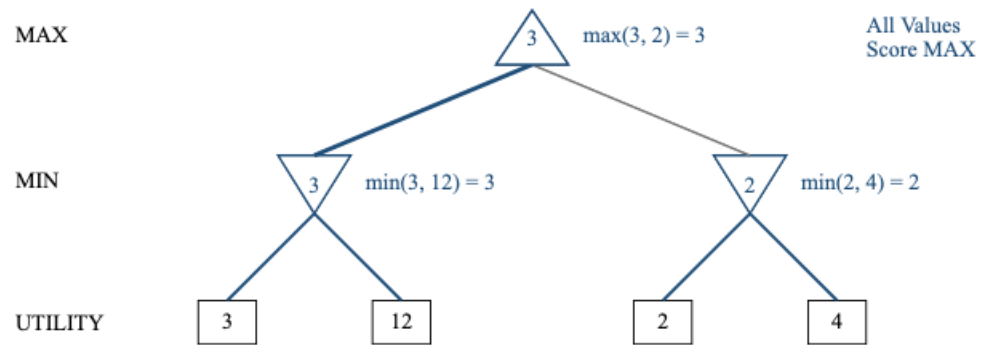
Speaker notes

The right MIN node also chooses the smaller child: $\min(2, 4) = 2$.

Back Up Values from the Leaves

Slide 136 of 193 · Reveal 2

Back Up Values from the Leaves



MAX Chooses Left: 3 Is Larger Than 2

136

Speaker notes

MAX compares the two backed-up values. Since 3 is larger than 2, MAX chooses the left action.

Back Up Values from the Leaves

Slide 136 of 193 · Reveal 3

Minimax Follows the Tree Recursively

```
def minimax(tree, maximizing=True):
```

137

Minimax Follows the Tree Recursively

Slide 137 of 193 · Initial

Speaker notes

The function mirrors the game tree. Each call evaluates one state.

Minimax Follows the Tree Recursively

```
def minimax(tree, maximizing=True):  
  
    if not isinstance(tree, list): return tree
```

Base Case

A leaf returns its utility

137

Speaker notes

A leaf has no children, so the base case returns its utility immediately.

Minimax Follows the Tree Recursively

Slide 137 of 193 · Reveal 1

Minimax Follows the Tree Recursively

```
def minimax(tree, maximizing=True):  
  
    if not isinstance(tree, list): return tree  
  
    values = [minimax(child, not maximizing) for child in tree]
```

Base Case

A leaf returns its utility

Recursive Step

Visit every child and switch the player

137

Minimax Follows the Tree Recursively

Slide 137 of 193 · Reveal 2

Speaker notes

For an internal state, recurse on every child. Passing not maximizing alternates MAX and MIN at each level.

Minimax Follows the Tree Recursively

```
def minimax(tree, maximizing=True):  
  
    if not isinstance(tree, list): return tree  
  
    values = [minimax(child, not maximizing) for child in tree]  
  
    return max(values) if maximizing else min(values)
```

Base Case

A leaf returns its utility

Recursive Step

Visit every child and switch the player

Back Up

MAX returns max; MIN returns min

One Recursive Call per State in the Game Tree

137

Speaker notes

After the child calls return, back up their values with max at a MAX node or min at a MIN node.

Minimax Follows the Tree Recursively

Slide 137 of 193 · Reveal 3

Run Minimax on a Deeper Tree

main.py

```
1 def minimax(tree, maximizing=True):
2     if not isinstance(tree, list): return tree
3     values = [minimax(child, not maximizing) for child in tree]
4     return max(values) if maximizing else min(values)
5 def node(*children): return list(children)
6 left = node(node(3, 5), node(6, 9))
7 middle = node(node(1, 2), node(0, -1))
8 right = node(node(7, 4), node(5, 8))
9 tree = node(left, middle, right)
10 values = [minimax(child, False) for child in tree]
11 best = max(range(len(tree)), key=lambda i: values[i])
12 print("Child values:", values, "Root value:", minimax(tree))
13 print("Best move:", ["Left", "Middle", "Right"][best])
14
```

Download: minimax.py

138

Run Minimax on a Deeper Tree

Slide 138 of 193 · Initial

Speaker notes

The node helper makes the nested structure read like the diagram: each call creates one internal tree node.

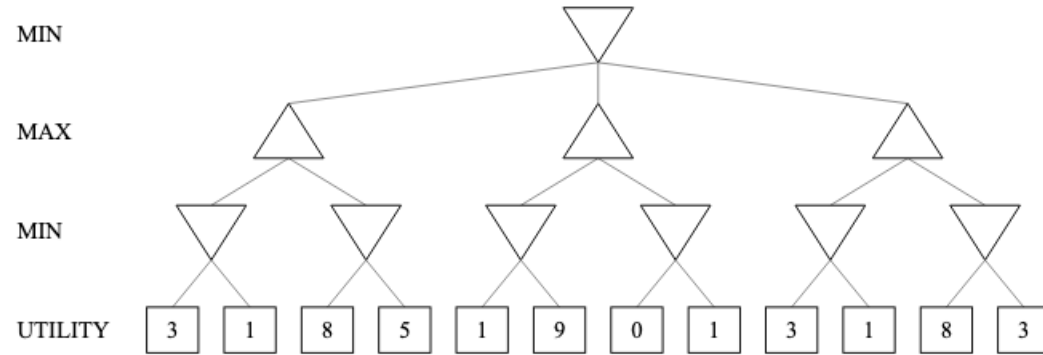
The tree has three decision levels: the root is MAX, its children are MIN, and their children are MAX before the utilities.

The three root-child values are 5, 0 and 7. The root value is 7, so MAX chooses Right.

The function returns a value. The final lines recover the corresponding root action; ties choose the first branch.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

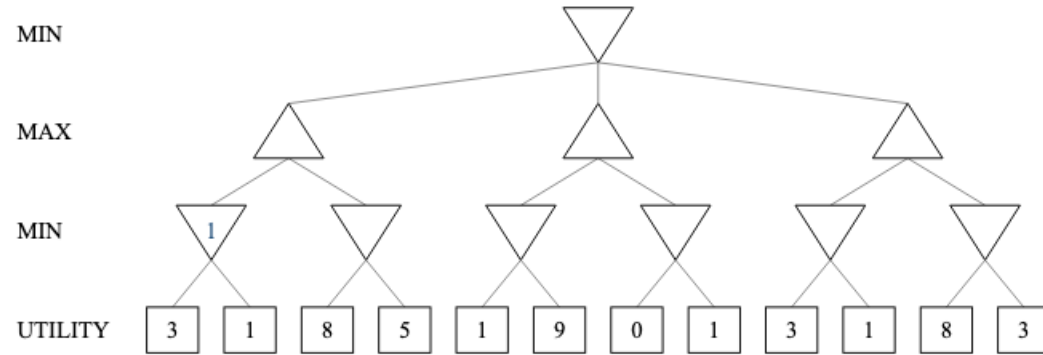
Slide 139 of 193 · Initial

Speaker notes

Work from the bottom level upward. All scores still belong to MAX.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

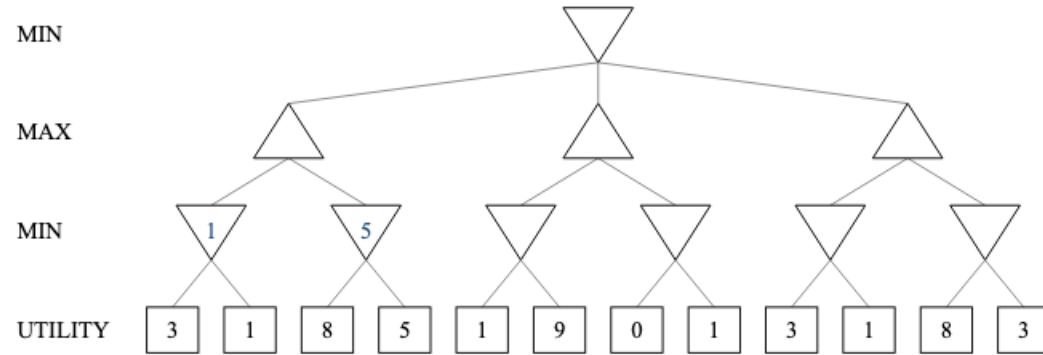
Slide 139 of 193 · Reveal 1

Speaker notes

Left pair: MIN chooses 1 from 3 and 1.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

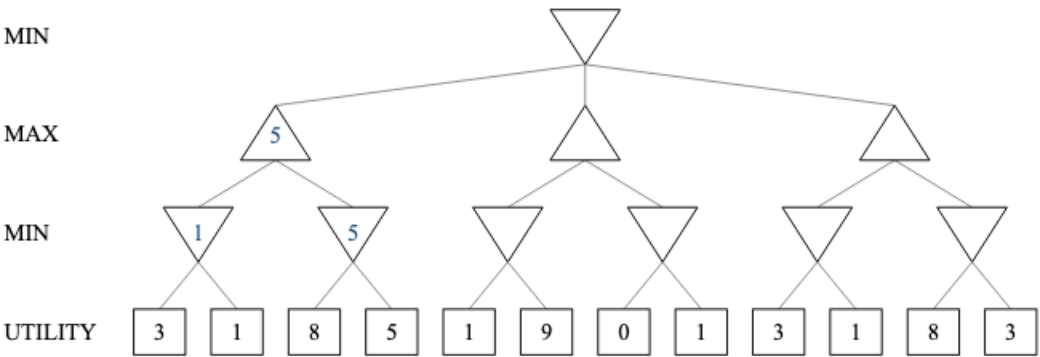
Slide 139 of 193 · Reveal 2

Speaker notes

Next pair: MIN chooses 5 from 8 and 5.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Speaker notes

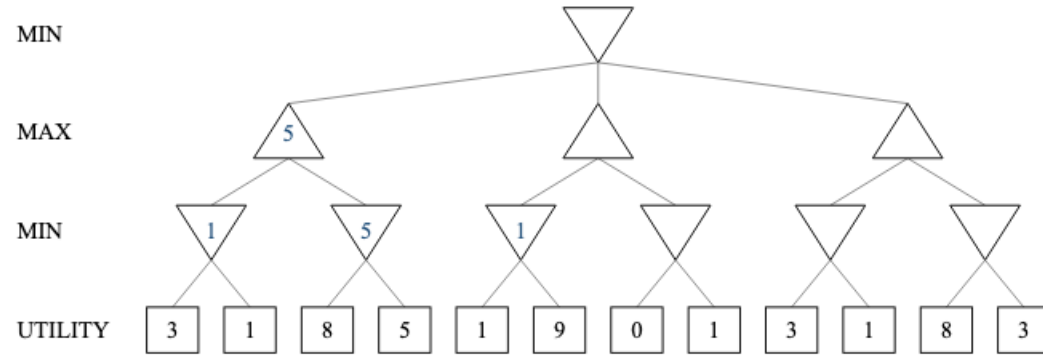
Left MAX chooses 5 from 1 and 5.

Minimax Practice

Slide 139 of 193 · Reveal 3

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Speaker notes

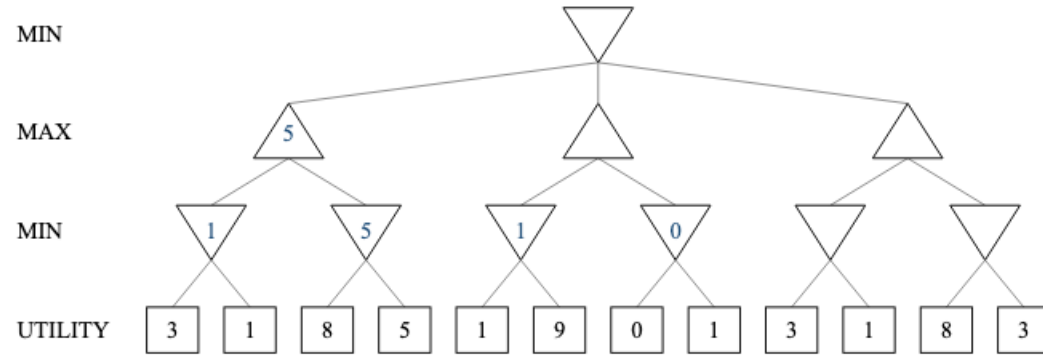
Middle-left pair: MIN chooses 1 from 1 and 9.

Minimax Practice

Slide 139 of 193 · Reveal 4

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Speaker notes

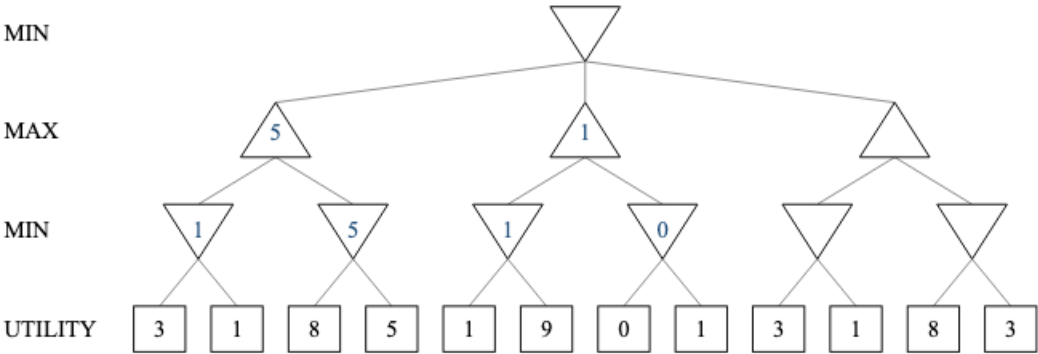
Middle-right pair: MIN chooses 0 from 0 and 1.

Minimax Practice

Slide 139 of 193 · Reveal 5

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

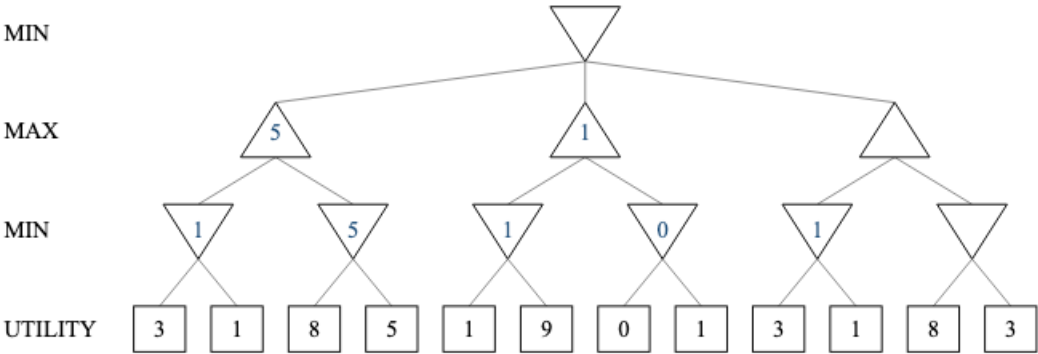
Slide 139 of 193 · Reveal 6

Speaker notes

Middle MAX chooses 1 from 1 and 0.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

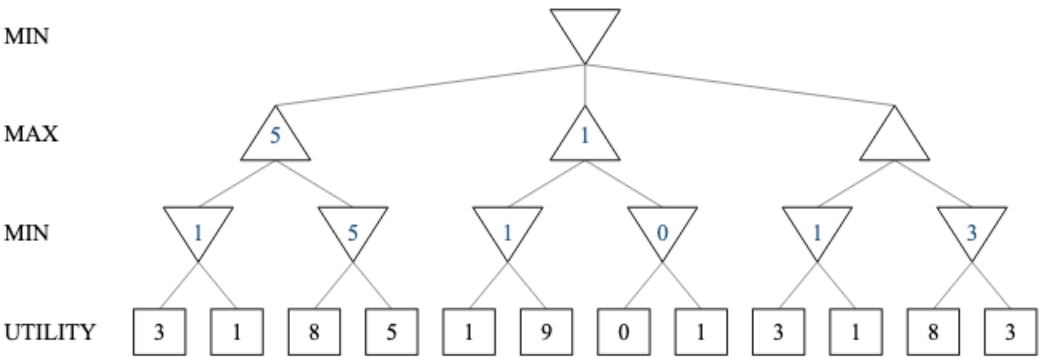
Slide 139 of 193 · Reveal 7

Speaker notes

Right-left pair: MIN chooses 1 from 3 and 1.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Minimax Practice

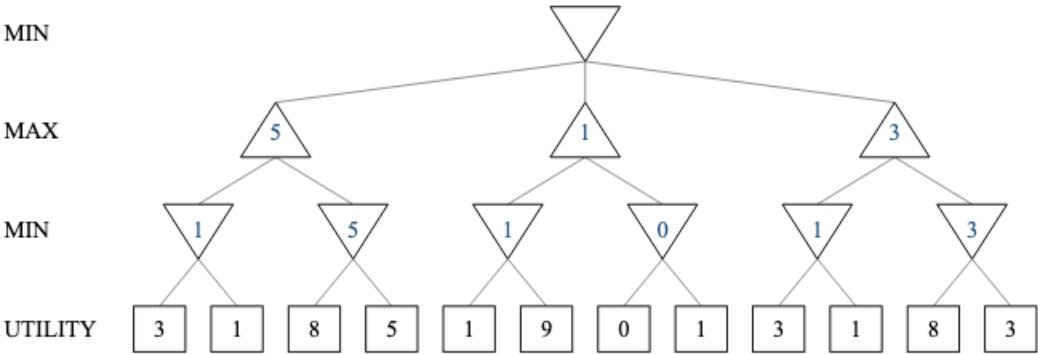
Slide 139 of 193 · Reveal 8

Speaker notes

Right-right pair: MIN chooses 3 from 8 and 3.

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Speaker notes

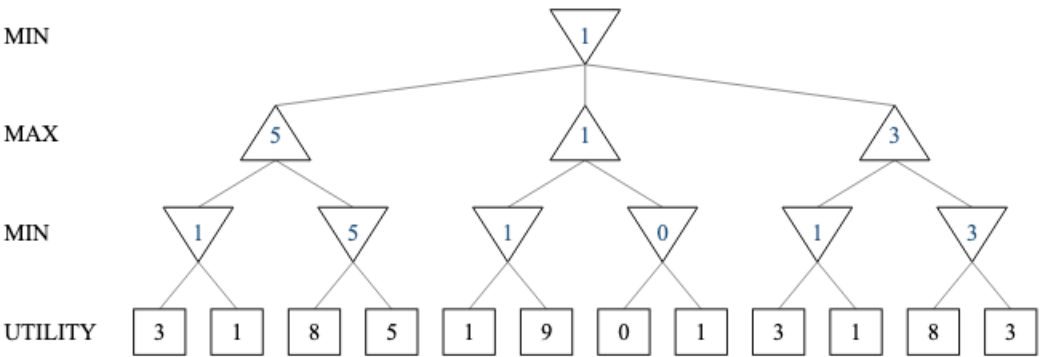
Right MAX chooses 3 from 1 and 3.

Minimax Practice

Slide 139 of 193 · Reveal 9

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



139

Speaker notes

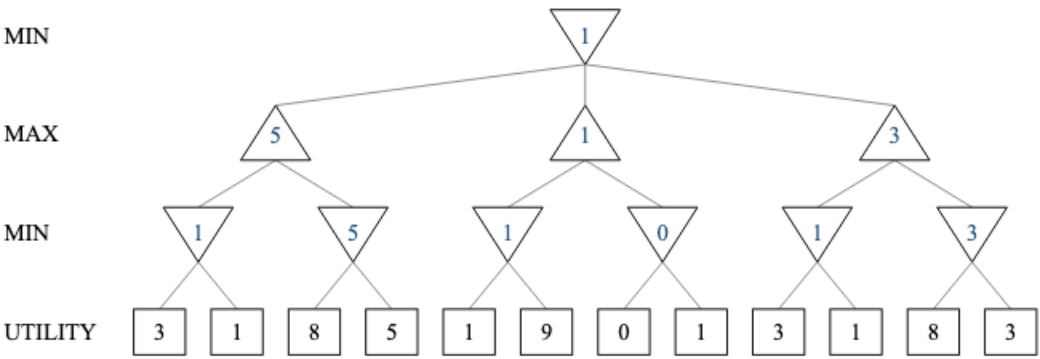
Root MIN compares 5, 1 and 3 and chooses 1.

Minimax Practice

Slide 139 of 193 · Reveal 10

Minimax Practice

MIN Moves First: What Is the Root Value? Which Move?



Value 1 · MIN Chooses the Middle Branch

139

Speaker notes

MIN chooses the middle branch, with value 1.

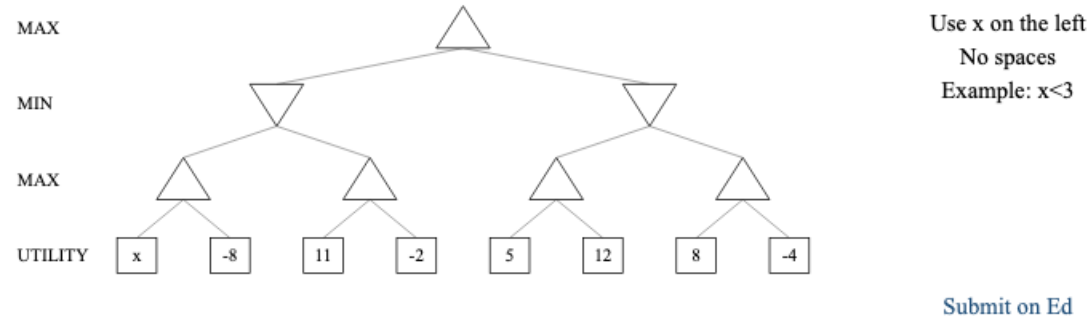
Minimax Practice

Slide 139 of 193 · Reveal 11

Task: Minimax

TASK

Find the Values of x That Make Left Strictly Better



140

Task: Minimax

Slide 140 of 193 · Initial

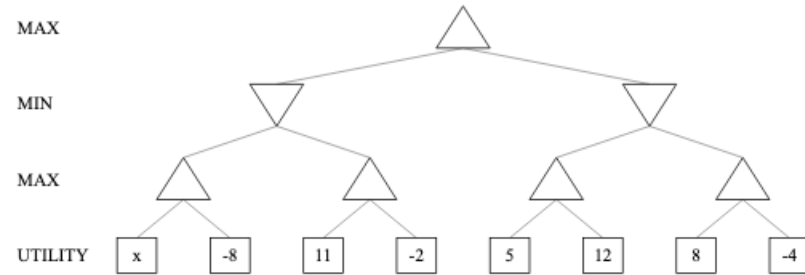
Speaker notes

Do not assume how ties are broken. A guaranteed left choice needs a strictly larger value.

Solution: Minimax

SOLUTION

Find the Values of x That Make Left Strictly Better



$$\text{Left} = \min(\max(x, -8), 11) \cdot \text{Right} = 8$$

140

Speaker notes

The right branch is $\min(12, 8) = 8$. Express the left branch in terms of x .

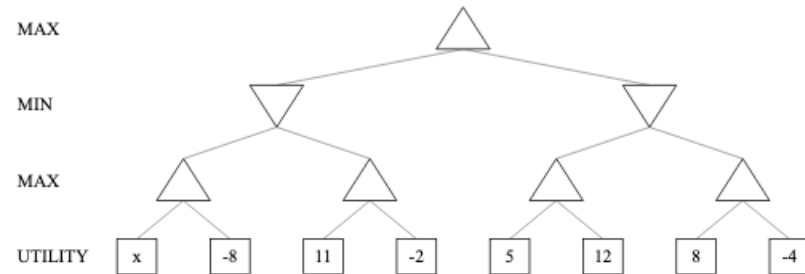
Task: Minimax

Slide 140 of 193 · Reveal 1

Solution: Minimax

SOLUTION

Find the Values of x That Make Left Strictly Better



$$\text{Left} = \min(\max(x, -8), 11) \cdot \text{Right} = 8$$

$x > 8$: Left Wins · $x = 8$: Tie

140

Speaker notes

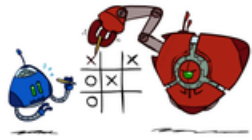
Left is strictly better exactly when x exceeds 8. At equality, either action is minimax-optimal.

Task: Minimax

Slide 140 of 193 · Reveal 2

What Does Minimax Guarantee?

An Optimal Strategy Against Optimal Opposition



141

What Does Minimax Guarantee?

Slide 141 of 193 · Initial

Speaker notes

The original slide's "No" was misleading. In a finite game with exact terminal utilities, minimax does give an optimal strategy.

What Does Minimax Guarantee?

An Optimal Strategy Against Optimal Opposition



It Maximizes the Worst-Case Payoff

141

Speaker notes

Against any legal opponent play, following the strategy guarantees at least the minimax value.

What Does Minimax Guarantee?

Slide 141 of 193 · Reveal 1

What Does Minimax Guarantee?

An Optimal Strategy Against Optimal Opposition



It Maximizes the Worst-Case Payoff

It Does Not Predict Every Opponent's Actual Move

141

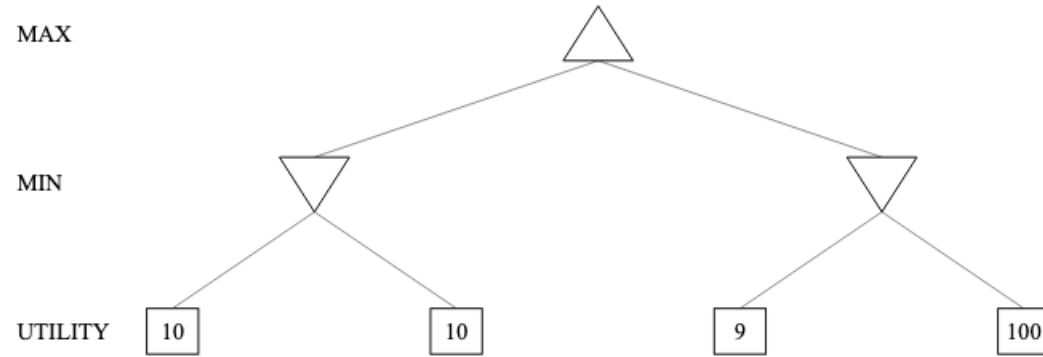
Speaker notes

An opponent can make mistakes. Exploiting a particular opponent requires assumptions beyond worst-case play.

What Does Minimax Guarantee?

Slide 141 of 193 · Reveal 2

What If MIN Makes Mistakes?



142

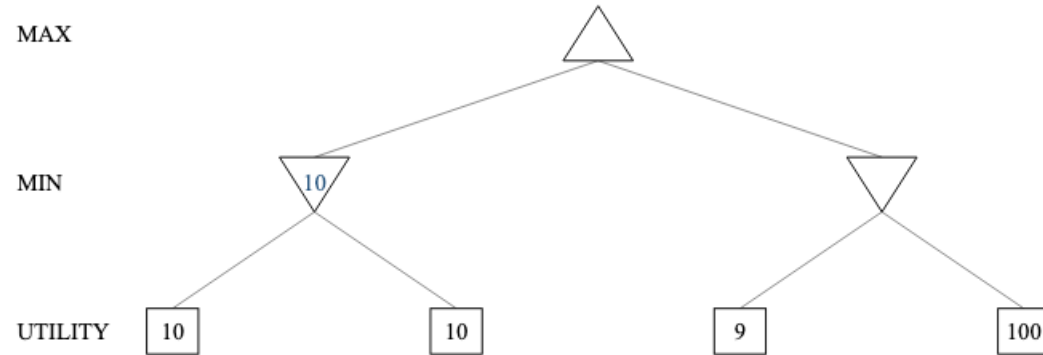
What If MIN Makes Mistakes?

Slide 142 of 193 · Initial

Speaker notes

Which branch is safe, and which branch offers a chance of a much higher score?

What If MIN Makes Mistakes?



142

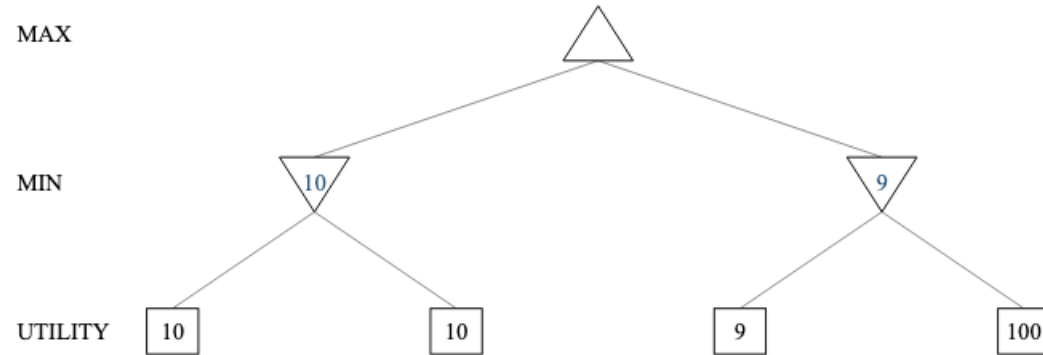
Speaker notes

MIN can hold the left branch to 10.

What If MIN Makes Mistakes?

Slide 142 of 193 · Reveal 1

What If MIN Makes Mistakes?



142

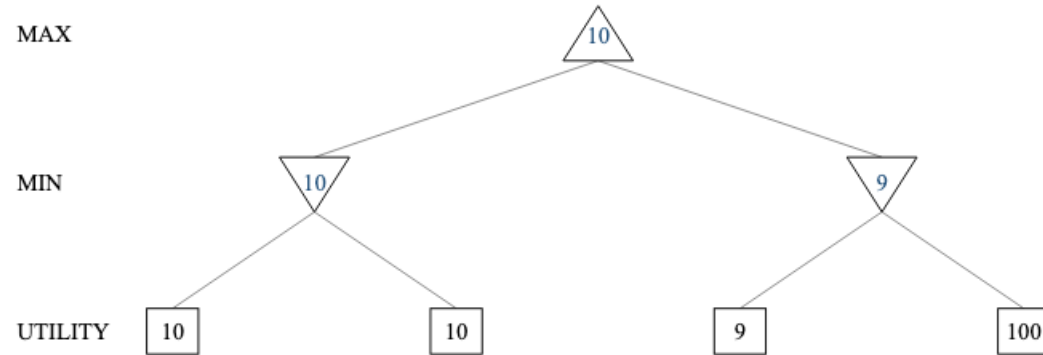
Speaker notes

MIN can hold the right branch to 9.

What If MIN Makes Mistakes?

Slide 142 of 193 · Reveal 2

What If MIN Makes Mistakes?



142

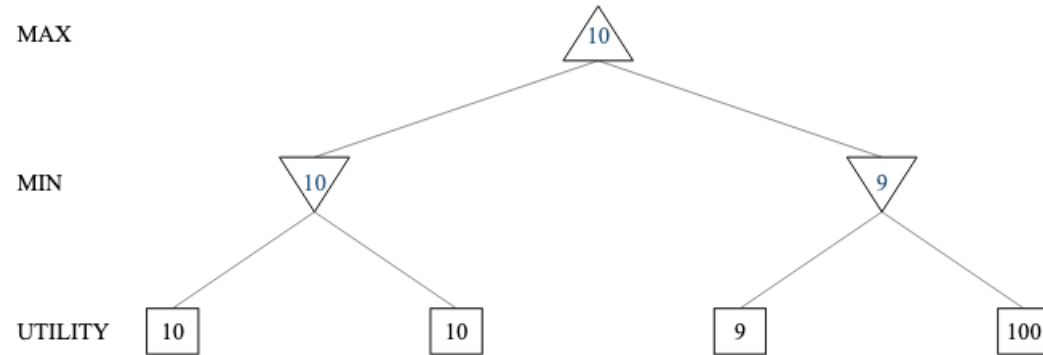
What If MIN Makes Mistakes?

Slide 142 of 193 · Reveal 3

Speaker notes

MAX takes the guarantee of 10.

What If MIN Makes Mistakes?



Left Guarantees 10 · Right Guarantees 9

142

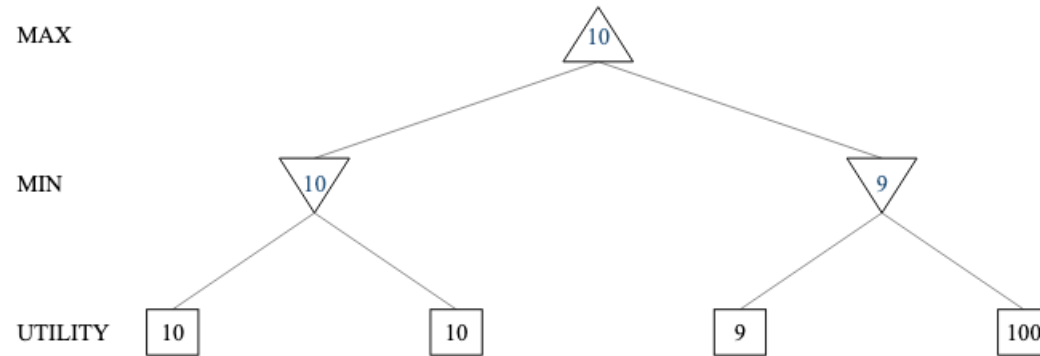
Speaker notes

Minimax protects against the worst reply; it does not maximize the payoff against every imperfect opponent.

What If MIN Makes Mistakes?

Slide 142 of 193 · Reveal 4

What If MIN Makes Mistakes?



Left Guarantees 10 · Right Guarantees 9

If MIN Chooses Randomly: Right Averages 54.5

142

Speaker notes

The average 54.5 assumes the two right replies are equally likely. Without an opponent model, that average is not justified.

What If MIN Makes Mistakes?

Slide 142 of 193 · Reveal 5

Minimax Properties

Finite Game Tree · Branching Factor b · Maximum Depth m

Search All Replies, Depth First

143

Speaker notes

A move and its reply are two separate levels. Minimax explores the full tree.

Minimax Properties

Slide 143 of 193 · Initial

Minimax Properties

Finite Game Tree · Branching Factor b · Maximum Depth m

Complete?

Yes, for a finite game tree

Search All Replies, Depth First

143

Speaker notes

Finite matters: cycles or unbounded play need additional handling.

Minimax Properties

Slide 143 of 193 · Reveal 1

Minimax Properties

Finite Game Tree · Branching Factor b · Maximum Depth m

Complete?	Yes, for a finite game tree
Optimal?	Yes, against optimal opposition

Search All Replies, Depth First

143

Speaker notes

Exact utilities and the stated game assumptions are essential.

Minimax Properties

Slide 143 of 193 · Reveal 2

Minimax Properties

Finite Game Tree · Branching Factor b · Maximum Depth m

Complete?	Yes, for a finite game tree
Optimal?	Yes, against optimal opposition
Time	$O(b^m)$

Search All Replies, Depth First

143

Speaker notes

Every branch is considered, so the number of nodes is exponential in depth.

Minimax Properties

Slide 143 of 193 · Reveal 3

Minimax Properties

Finite Game Tree · Branching Factor b · Maximum Depth m

Complete?	Yes, for a finite game tree
Optimal?	Yes, against optimal opposition
Time	$O(b^m)$
Space	$O(bm)$, with DFS and stored successor lists
	Search All Replies, Depth First

143

Speaker notes

With successor lists along a DFS path, space is $O(bm)$. This excludes storage of the given tree; lazy generation can reduce traversal overhead.

Minimax Properties

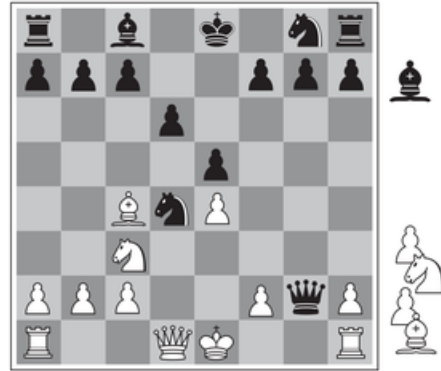
Slide 143 of 193 · Reveal 4

Full Search Is Too Large

Illustrative Chess Estimate

$$b \approx 35 \cdot m \approx 100$$

$$35^{100} \approx 2.6 \times 10^{154}$$



144

Speaker notes

These are rough branching and depth assumptions, not an exact count of legal chess games. The scale is the point.

Full Search Is Too Large

Slide 144 of 193 · Initial

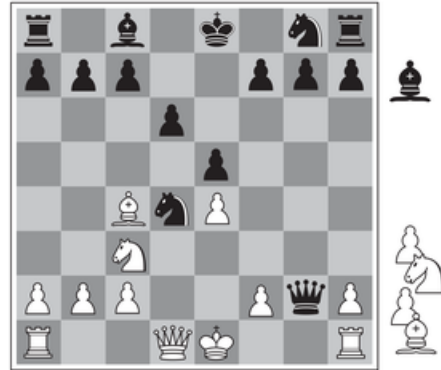
Full Search Is Too Large

Illustrative Chess Estimate

$$b \approx 35 \cdot m \approx 100$$

$$35^{100} \approx 2.6 \times 10^{154}$$

We Cannot Search to Every Terminal
Position



144

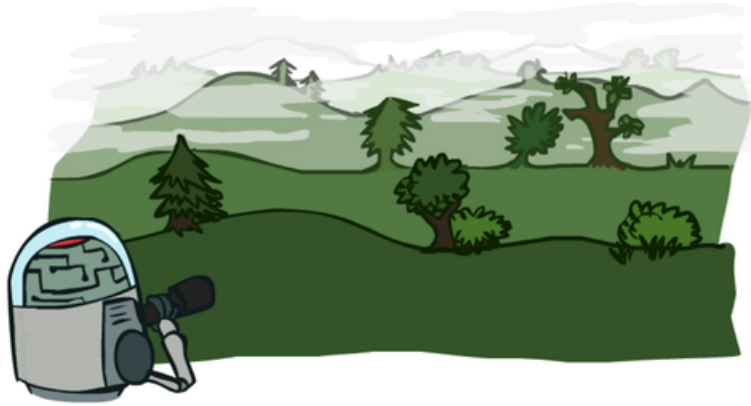
Speaker notes

We need to make a decision before reaching all terminal states.

Full Search Is Too Large

Slide 144 of 193 · Reveal 1

Resource Limits



145

Resource Limits

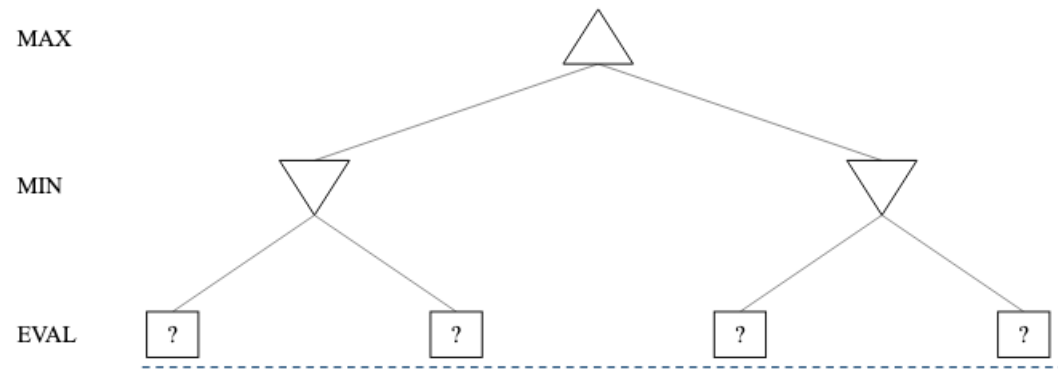
Slide 145 of 193 · Initial

Speaker notes

We cannot see the whole tree. How can we compare the positions at the edge of our search?

Stop at a Depth Limit

Depth Limit $\ell = 2$ · MAX and MIN Still Alternate



146

Speaker notes

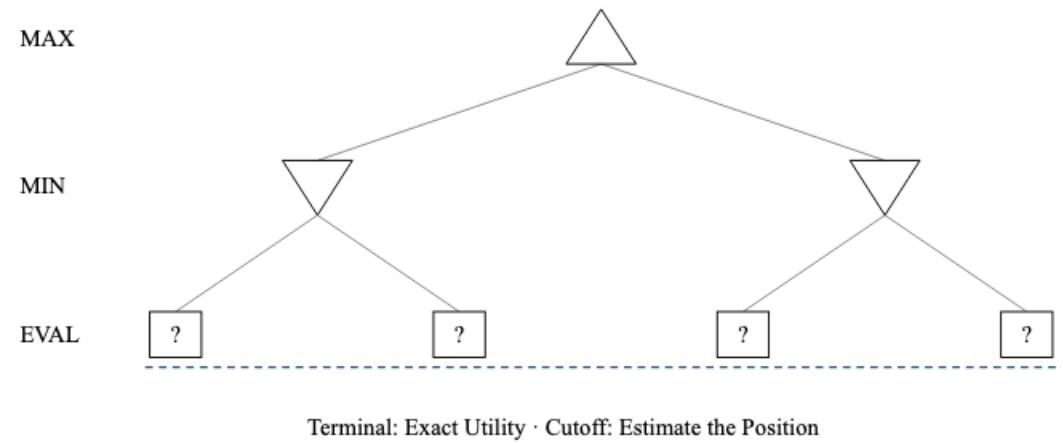
This is depth-limited minimax, not ordinary single-agent DFS. A depth level counts one move by one player.

Stop at a Depth Limit

Slide 146 of 193 · Initial

Stop at a Depth Limit

Depth Limit $\ell = 2$ · MAX and MIN Still Alternate



146

Speaker notes

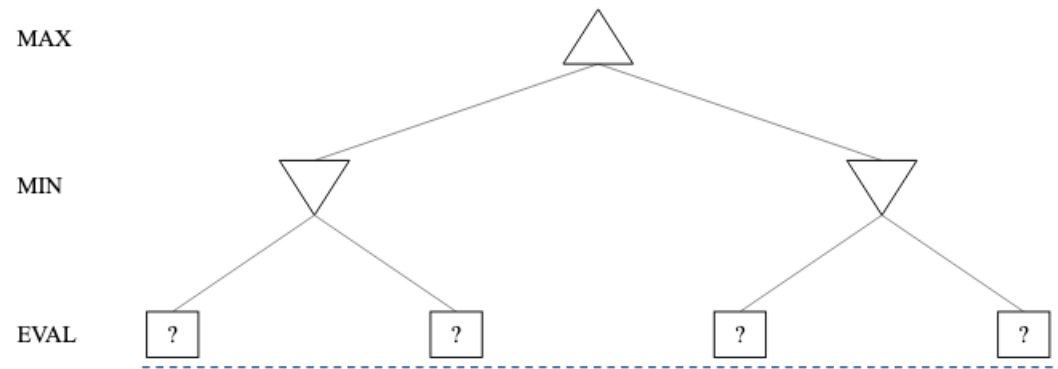
Check terminal states first. Only non-terminal cutoff states use an evaluation function.

Stop at a Depth Limit

Slide 146 of 193 · Reveal 1

Stop at a Depth Limit

Depth Limit $\ell = 2$ · MAX and MIN Still Alternate



Terminal: Exact Utility · Cutoff: Estimate the Position

Time: $O(b^\ell)$ · Space: $O(b\ell)$

146

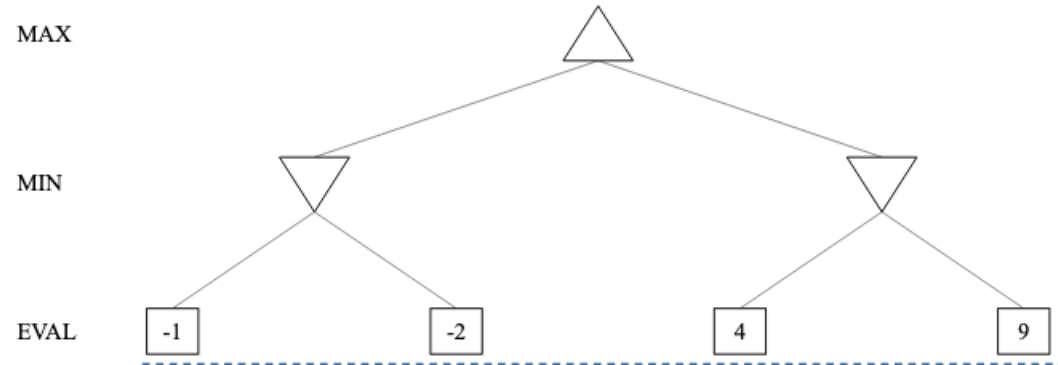
Speaker notes

Finite branching and a finite cutoff make the search terminate, but it need not reach a terminal outcome. The DFS bounds use stored successor lists.

Stop at a Depth Limit

Slide 146 of 193 · Reveal 2

Back Up the Cutoff Estimates



147

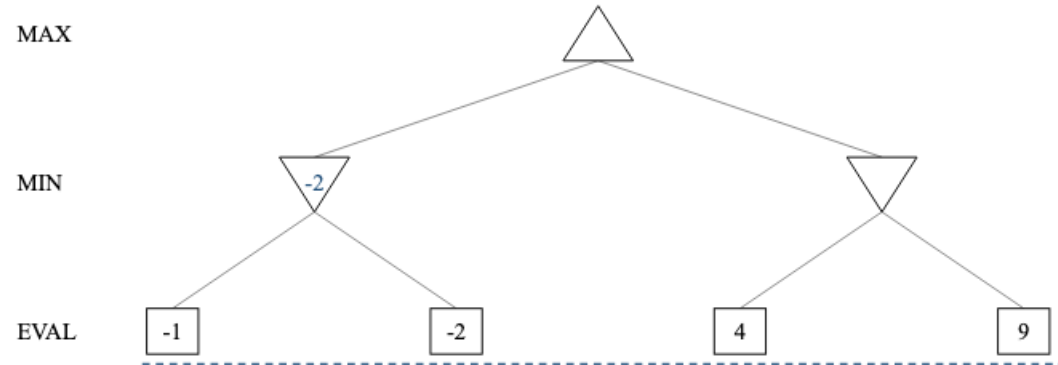
Speaker notes

The four numbers are evaluation estimates, not known terminal outcomes.

Back Up the Cutoff Estimates

Slide 147 of 193 · Initial

Back Up the Cutoff Estimates



147

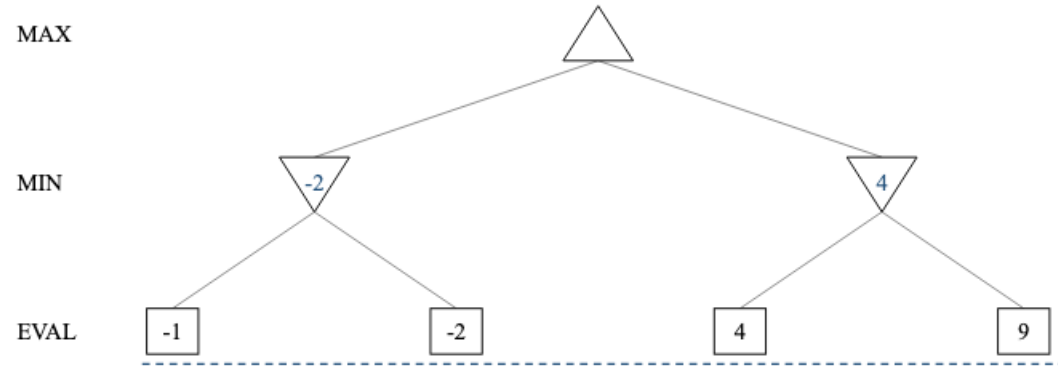
Speaker notes

Left MIN returns -2.

Back Up the Cutoff Estimates

Slide 147 of 193 · Reveal 1

Back Up the Cutoff Estimates



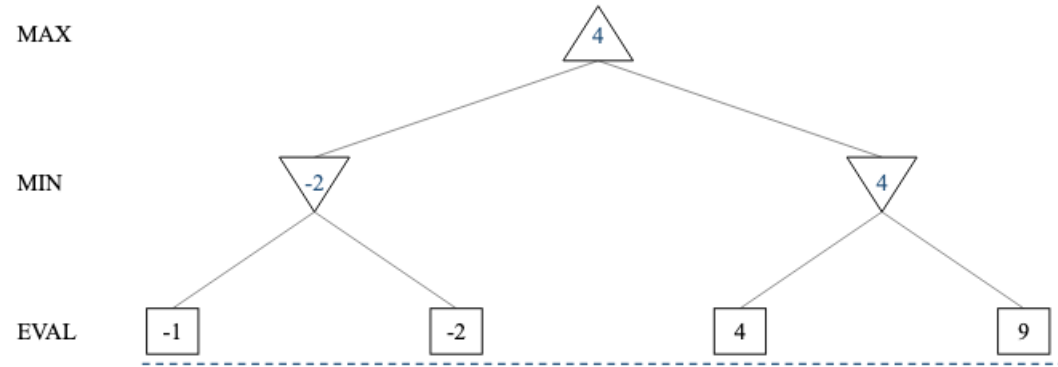
Speaker notes

Right MIN returns 4.

Back Up the Cutoff Estimates

Slide 147 of 193 · Reveal 2

Back Up the Cutoff Estimates



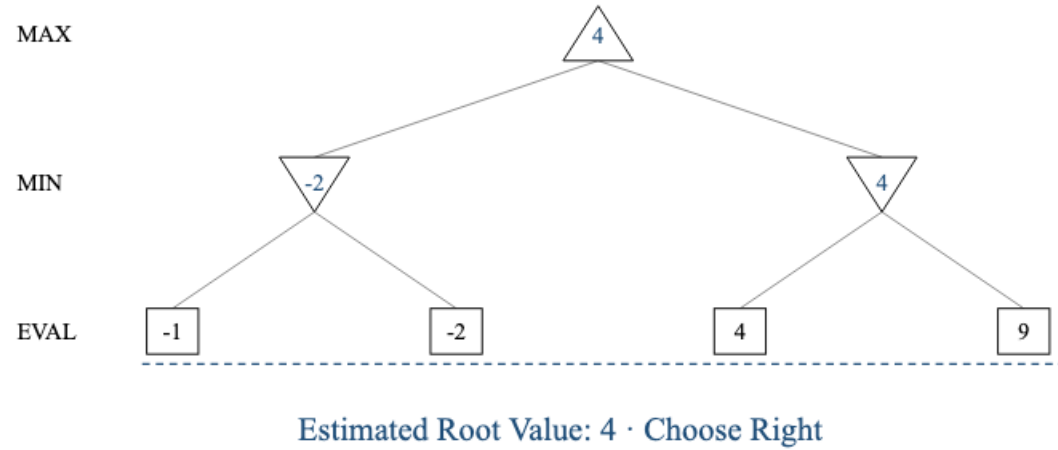
Speaker notes

MAX returns 4.

Back Up the Cutoff Estimates

Slide 147 of 193 · Reveal 3

Back Up the Cutoff Estimates



147

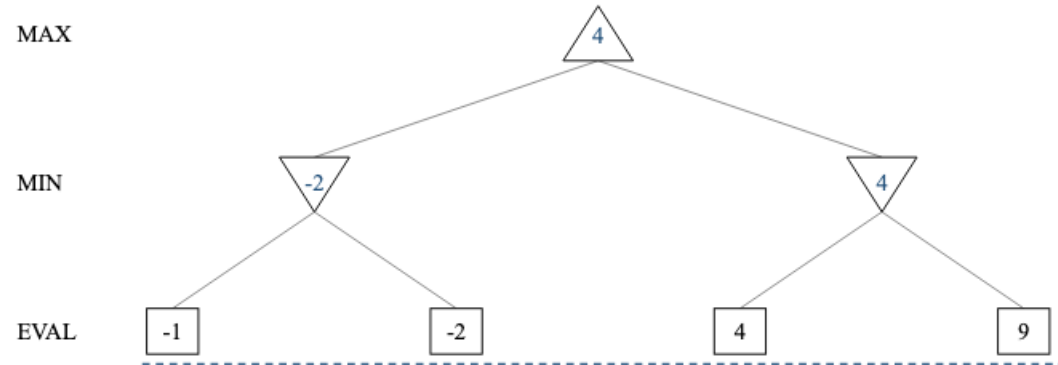
Speaker notes

Choose right according to these estimates.

Back Up the Cutoff Estimates

Slide 147 of 193 · Reveal 4

Back Up the Cutoff Estimates



Estimated Root Value: 4 · Choose Right

The Guarantee Depends on the Estimates

147

Speaker notes

Approximate leaf evaluations remove the full minimax optimality guarantee.

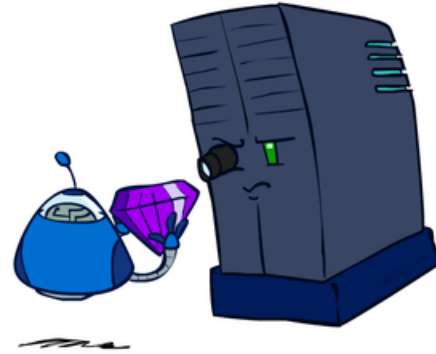
Back Up the Cutoff Estimates

Slide 147 of 193 · Reveal 5

Evaluate a Non-Terminal Position

Eval(s): Estimate the Position's Value
for MAX

Ideal: The True Minimax Value



148

Speaker notes

Use the same score perspective at every depth. The estimate is not necessarily a probability or expected value.

Evaluate a Non-Terminal Position

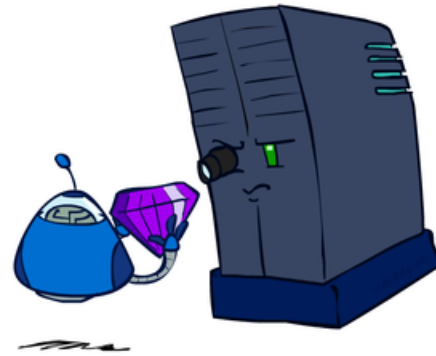
Slide 148 of 193 · Initial

Evaluate a Non-Terminal Position

Eval(s): Estimate the Position's Value
for MAX

Ideal: The True Minimax Value

Useful: Fast Enough to Evaluate Many
Positions



148

Speaker notes

An evaluation function must balance quality against the search time it consumes.

Evaluate a Non-Terminal Position

Slide 148 of 193 · Reveal 1

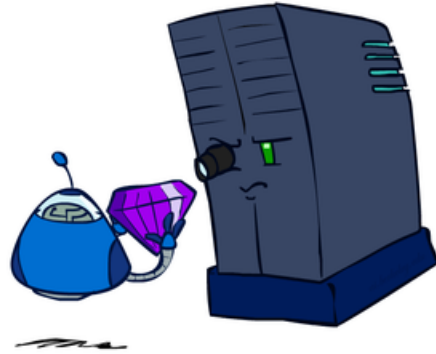
Evaluate a Non-Terminal Position

Eval(s): Estimate the Position's Value
for MAX

Ideal: The True Minimax Value

Useful: Fast Enough to Evaluate Many
Positions

A Larger Number Should Mean a Better
Position for MAX



148

Speaker notes

A poor estimate can make deeper search choose the wrong move.

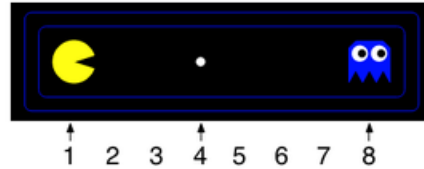
Evaluate a Non-Terminal Position

Slide 148 of 193 · Reveal 2

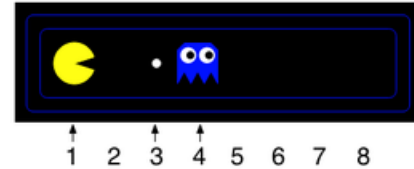
TASK

Task: Evaluation Functions

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left Position?

f = distance to food · g = distance to ghost

149

Speaker notes

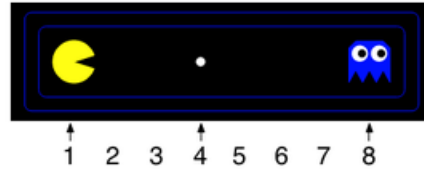
Use the marked corridor coordinates. Which evaluation functions prefer the left position?

Task: Evaluation Functions

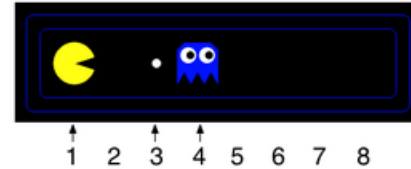
Slide 149 of 193 · Initial

TASK**Task: Evaluation Functions**

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left
Position?

A. $\frac{1}{f}$

f = distance to food · g = distance to ghost

149

Speaker notes

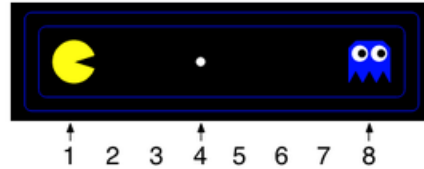
Choice A rewards proximity to food. Since $1/2$ is larger than $1/3$, A prefers right.

Task: Evaluation Functions

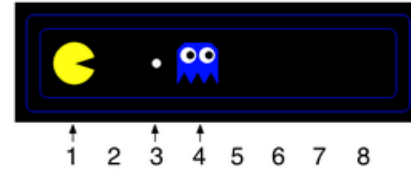
Slide 149 of 193 · Reveal 1

TASK**Task: Evaluation Functions**

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left Position?

A. $\frac{1}{f}$

B. g

f = distance to food · g = distance to ghost

149

Speaker notes

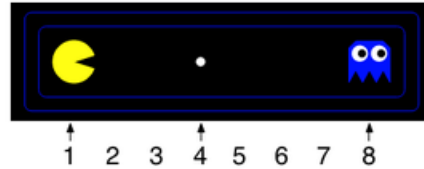
Choice B rewards distance from the ghost. Since 7 is larger than 3, B prefers left.

Task: Evaluation Functions

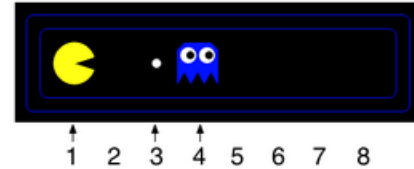
Slide 149 of 193 · Reveal 2

TASK**Task: Evaluation Functions**

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left Position?

f = distance to food · g = distance to ghost

A. $\frac{1}{f}$

B. g

C. $g + \frac{1}{f}$

149

Speaker notes

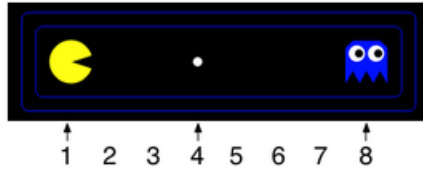
Choice C combines both features: left is $7 + 1/3$, while right is $3 + 1/2$, so C prefers left.

Task: Evaluation Functions

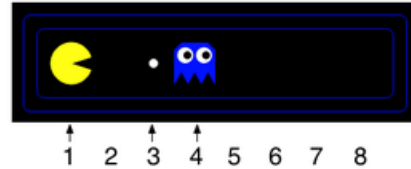
Slide 149 of 193 · Reveal 3

TASK**Task: Evaluation Functions**

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left Position?

f = distance to food · g = distance to ghost

A. $\frac{1}{f}$

B. g

C. $g + \frac{1}{f}$

D. $g + \frac{1000}{f}$

149

Speaker notes

Choice D weights food distance heavily: left is $7 + 1000/3$, while right is $3 + 1000/2$, so D prefers right.

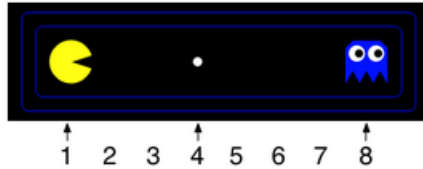
Task: Evaluation Functions

Slide 149 of 193 · Reveal 4

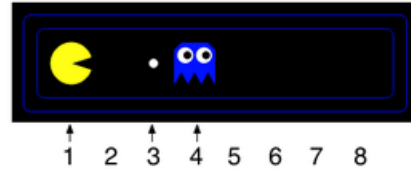
SOLUTION

Solution: Evaluation Functions

LEFT POSITION



RIGHT POSITION



Which Functions Prefer the Left Position?

f = distance to food · g = distance to ghost

- A. $\frac{1}{f}$ B. g C. $g + \frac{1}{f}$ D. $g + \frac{1000}{f}$

Left: $f = 3, g = 7$ · Right: $f = 2, g = 3$

149

Speaker notes

Now reveal the measured feature values for both positions.

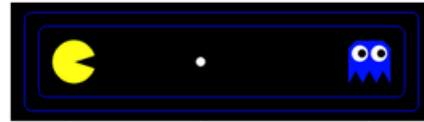
Task: Evaluation Functions

Slide 149 of 193 · Reveal 5

SOLUTION

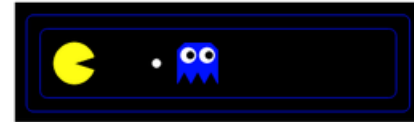
Solution: Evaluation Functions

LEFT POSITION



↑ 1 2 3 4 5 6 7 8 ↑

RIGHT POSITION



↑ 1 2 3 4 5 6 7 8 ↑

Which Functions Prefer the Left Position?

f = distance to food · g = distance to ghost

- A. $\frac{1}{f}$ B. g C. $g + \frac{1}{f}$ D. $g + \frac{1000}{f}$

B and C Prefer Left · A and D Prefer Right

149

Speaker notes

Therefore B and C prefer left, while A and D prefer right.

Task: Evaluation Functions

Slide 149 of 193 · Reveal 6

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \cdots + w_n f_n(s)$$

150

Combine Features into an Evaluation

Slide 150 of 193 · Initial

Speaker notes

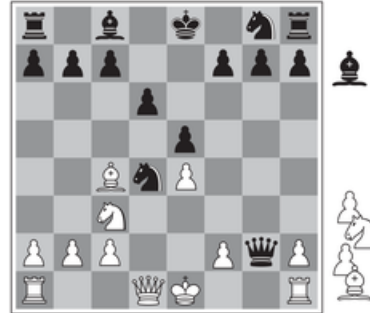
An evaluation function combines features using weights. Chess material gives a concrete example.

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn

1



150

Speaker notes

A pawn is the reference unit, with illustrative weight 1. Reveal the position we want to evaluate.

Combine Features into an Evaluation

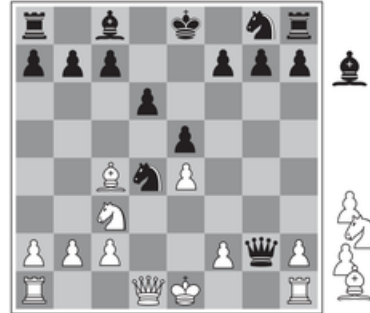
Slide 150 of 193 · Reveal 1

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn 1

Knight / Bishop 3



150

Combine Features into an Evaluation

Slide 150 of 193 · Reveal 2

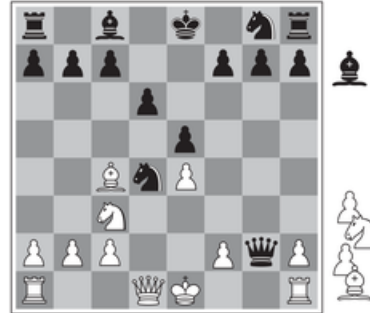
Speaker notes

Knights and bishops are often assigned an approximate material weight of 3.

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn	1
Knight / Bishop	3
Rook	5



150

Speaker notes

A rook is stronger and receives approximate weight 5.

Combine Features into an Evaluation

Slide 150 of 193 · Reveal 3

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn	1
Knight / Bishop	3
Rook	5
Queen	9



150

Speaker notes

A queen receives approximate weight 9. These are illustrative weights, not exact laws.

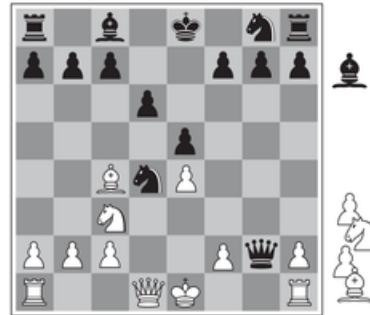
Combine Features into an Evaluation

Slide 150 of 193 · Reveal 4

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn	1
Knight / Bishop	3
Rook	5
Queen	9



Feature: MAX's Material – MIN's Material

150

Speaker notes

For each piece type, subtract MIN's count from MAX's count, multiply by its weight, and add the results.

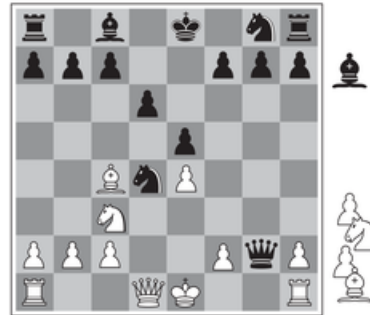
Combine Features into an Evaluation

Slide 150 of 193 · Reveal 5

Combine Features into an Evaluation

$$\text{Eval}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$$

Pawn	1
Knight / Bishop	3
Rook	5
Queen	9



Feature: MAX's Material – MIN's Material

Also Consider King Safety and Pawn Structure

150

Speaker notes

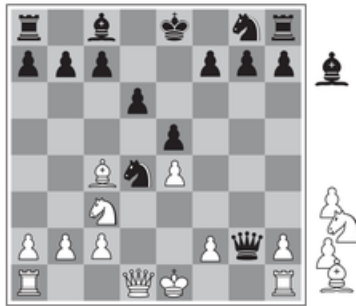
Material alone misses threats and position. A useful evaluation also considers features such as king safety and pawn structure.

Combine Features into an Evaluation

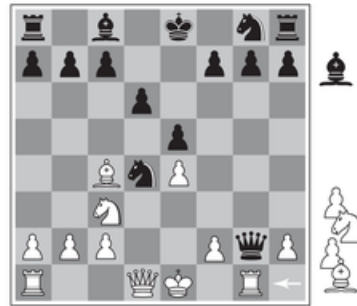
Slide 150 of 193 · Reveal 6

Material Is Not Enough

Who Has the Advantage in Each Position?



(a) White to move



(b) White to move

151

Speaker notes

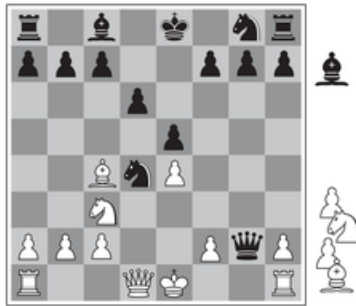
Ask students to compare material first, then look for an immediate tactical threat.

Material Is Not Enough

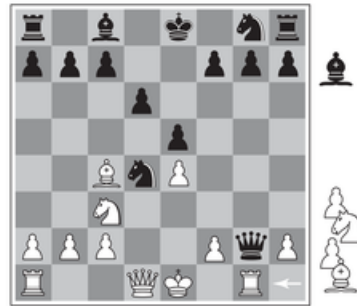
Slide 151 of 193 · Initial

Material Is Not Enough

Who Has the Advantage in Each Position?



(a) White to move



(b) White to move

(a) Material Count: Black Has More Material

151

Speaker notes

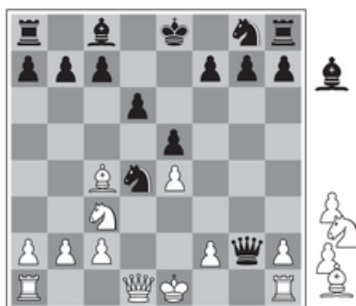
In position (a), counting the pieces gives Black a material advantage.

Material Is Not Enough

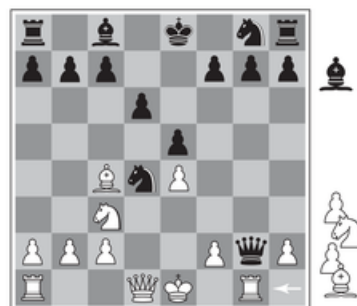
Slide 151 of 193 · Reveal 1

Material Is Not Enough

Who Has the Advantage in Each Position?



(a) White to move



(b) White to move

(b) Tactical Check: White Can Capture the Queen

151

Speaker notes

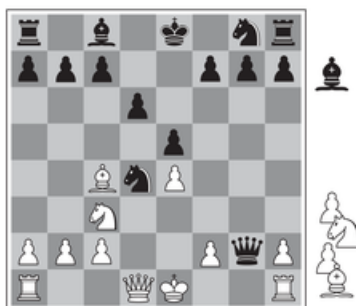
In position (b), White can capture the queen immediately. Looking one move ahead reverses the practical judgment.

Material Is Not Enough

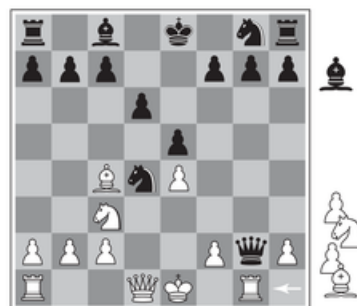
Slide 151 of 193 · Reveal 2

Material Is Not Enough

Who Has the Advantage in Each Position?



(a) White to move



(b) White to move

Material Alone Can Miss an Immediate Tactic

151

Speaker notes

The takeaway is that material alone can miss a decisive tactical feature.

Material Is Not Enough

Slide 151 of 193 · Reveal 3

Depth and Evaluation Error

At the Cutoff, We Estimate a Non-Terminal State



152

Speaker notes

A cutoff evaluation is an approximation.

Depth and Evaluation Error

Slide 152 of 193 · Initial

Depth and Evaluation Error

At the Cutoff, We Estimate a Non-Terminal State

A Deeper Cutoff Is Often Closer to the Real Outcome



152

Speaker notes

With more search, the approximation is applied closer to the real outcome, so local evaluation errors usually matter less.

Depth and Evaluation Error

Slide 152 of 193 · Reveal 1

Depth and Evaluation Error

At the Cutoff, We Estimate a Non-Terminal State

A Deeper Cutoff Is Often Closer to the Real Outcome

Tradeoff: Better Features or More Search?



152

Speaker notes

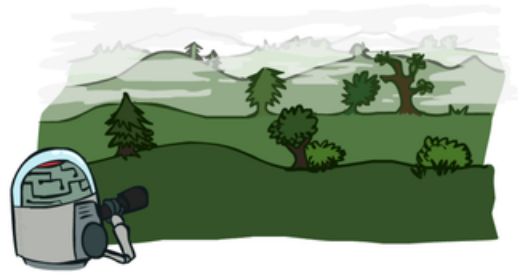
Computational effort can go into richer features or deeper search. Neither dominates in every domain.

Depth and Evaluation Error

Slide 152 of 193 · Reveal 2

The Horizon Effect

A Bad Consequence Lies Just Beyond
the Cutoff



153

The Horizon Effect

Slide 153 of 193 · Initial

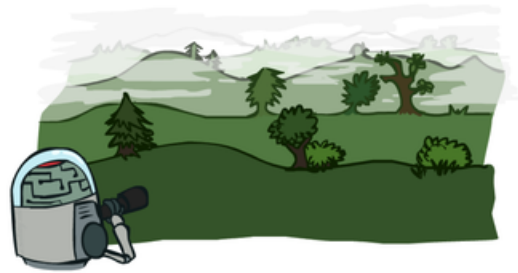
Speaker notes

The horizon effect occurs when a shallow search postpones a bad event past its cutoff.

The Horizon Effect

A Bad Consequence Lies Just Beyond
the Cutoff

A Delaying Move Pushes It Out of
Sight



153

Speaker notes

The evaluation then rewards delay even when the loss is unavoidable.

The Horizon Effect

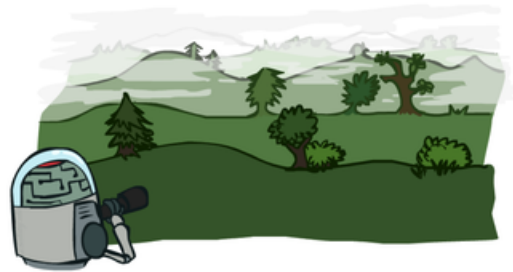
Slide 153 of 193 · Reveal 1

The Horizon Effect

A Bad Consequence Lies Just Beyond
the Cutoff

A Delaying Move Pushes It Out of
Sight

The Delay May Make the Final Outcome Worse



153

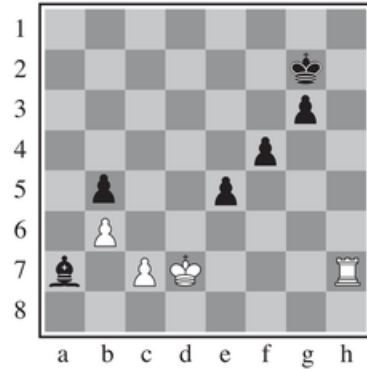
Speaker notes

Quiescence search is one practical response: continue through tactically unstable positions.

The Horizon Effect

Slide 153 of 193 · Reveal 2

Can Black Save the Bishop?



Black to Move

Can a Delaying Tactic Avoid the Loss?

154

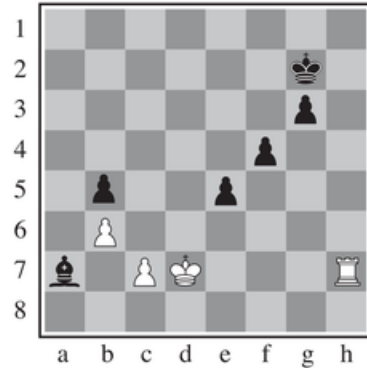
Speaker notes

Let students inspect the position before revealing the conclusion.

Can Black Save the Bishop?

Slide 154 of 193 · Initial

Can Black Save the Bishop?



Black to Move

Can a Delaying Tactic Avoid the Loss?

No · It Only Pushes the Loss Beyond a Shallow Horizon

154

Speaker notes

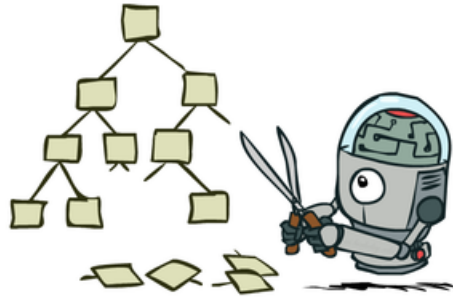
The bishop cannot be saved. A shallow search may still prefer moves that postpone the capture because the loss falls beyond its horizon.

Can Black Save the Bishop?

Slide 154 of 193 · Reveal 1

Can We Skip Part of the Tree?

Can We Compute the Exact Minimax Choice Without Visiting Every Node?



155

Speaker notes

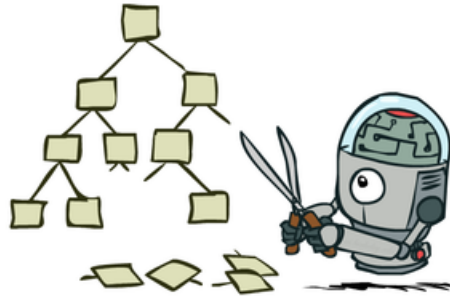
Pruning is safe only when the skipped branch cannot change the root decision.

Can We Skip Part of the Tree?

Slide 155 of 193 · Initial

Can We Skip Part of the Tree?

Can We Compute the Exact Minimax Choice Without Visiting Every Node?



Yes · When a Branch Cannot Affect the Decision

155

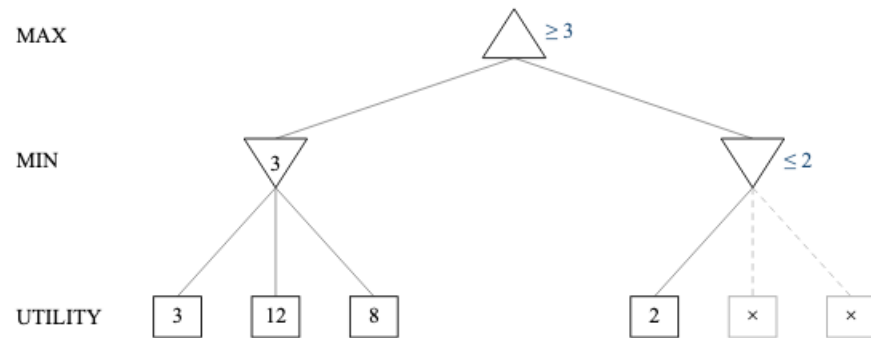
Speaker notes

Alpha-beta pruning returns the same root minimax value and an optimal action.

Can We Skip Part of the Tree?

Slide 155 of 193 · Reveal 1

A Branch Cannot Beat the Current Best



156

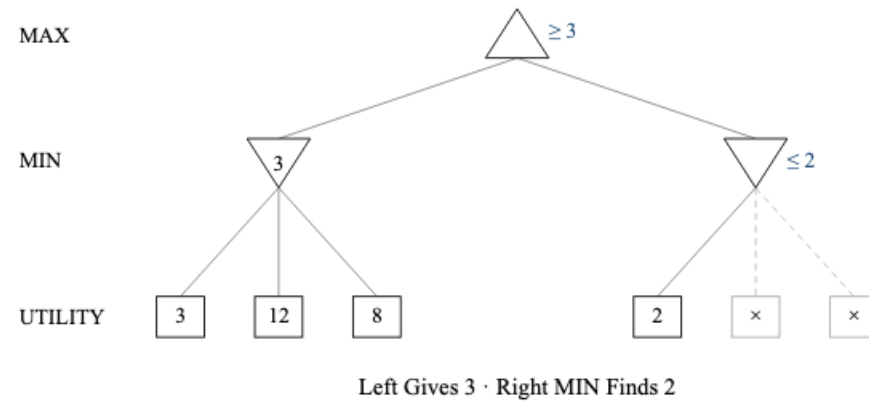
Speaker notes

Evaluate the left MIN node first: it returns 3, so MAX can already guarantee 3.

A Branch Cannot Beat the Current Best

Slide 156 of 193 · Initial

A Branch Cannot Beat the Current Best



156

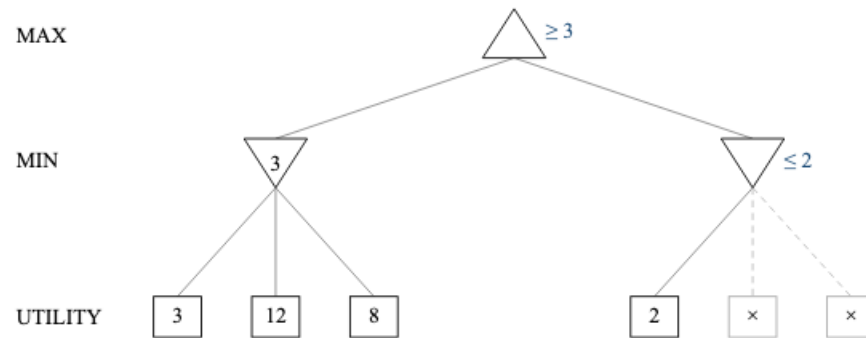
Speaker notes

At the right MIN node, the first child is 2. MIN can force at most 2 there.

A Branch Cannot Beat the Current Best

Slide 156 of 193 · Reveal 1

A Branch Cannot Beat the Current Best



Left Gives 3 · Right MIN Finds 2

The Other Right Leaves Cannot Change MAX's Choice

156

Speaker notes

MAX will never choose that branch over the known value 3, so the remaining children may be pruned.

A Branch Cannot Beat the Current Best

Slide 156 of 193 · Reveal 2

Alpha–Beta Pruning

Skip Branches That Cannot Change the Root Decision

157

Alpha–Beta Pruning

Slide 157 of 193 · Initial

Speaker notes

The previous slide showed one branch that could not beat MAX's current best. Alpha-beta generalizes that reasoning.

Alpha–Beta Pruning

Skip Branches That Cannot Change the Root Decision

Result

Same minimax value and optimal root action

157

Alpha–Beta Pruning

Slide 157 of 193 · Reveal 1

Speaker notes

The returned minimax value and an optimal root action stay unchanged.

Alpha–Beta Pruning

Skip Branches That Cannot Change the Root Decision

Result	Same minimax value and optimal root action
Benefit	Often visits far fewer nodes

157

Alpha–Beta Pruning

Slide 157 of 193 · Reveal 2

Speaker notes

The benefit is computational: branches that cannot affect the decision are not visited.

Alpha–Beta Pruning

Skip Branches That Cannot Change the Root Decision

Result	Same minimax value and optimal root action
Benefit	Often visits far fewer nodes
Requirement	Track bounds while searching left to right

157

Alpha–Beta Pruning

Slide 157 of 193 · Reveal 3

Speaker notes

To recognize those branches, carry bounds through a left-to-right depth-first search.

Alpha–Beta Pruning

Skip Branches That Cannot Change the Root Decision

Result	Same minimax value and optimal root action
Benefit	Often visits far fewer nodes
Requirement	Track bounds while searching left to right

Pruning Changes Work, Not the Answer

157

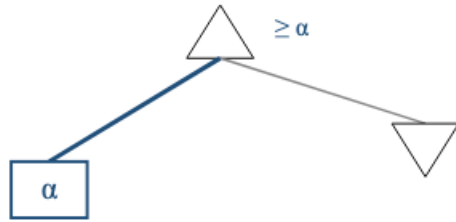
Speaker notes

Alpha-beta changes the amount of work, not the decision rule.

Alpha–Beta Pruning

Slide 157 of 193 · Reveal 4

MIN Pruning



158

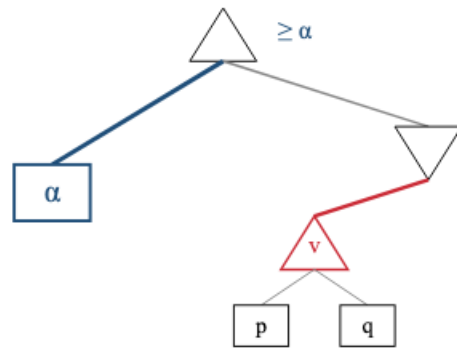
Speaker notes

Alpha is the best value MAX can already guarantee from another completed branch.

MIN Pruning

Slide 158 of 193 · Initial

MIN Pruning



158

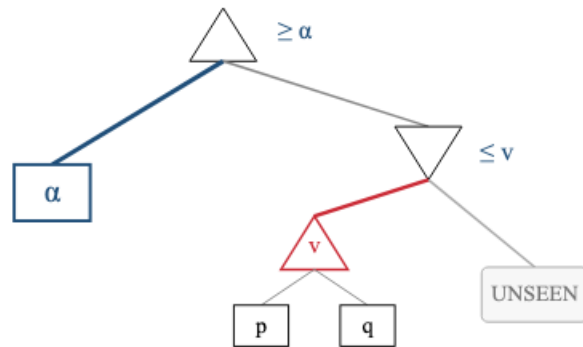
Speaker notes

The competing MIN node completes one child. That child may be a deeper subtree returning v .

MIN Pruning

Slide 158 of 193 · Reveal 1

MIN Pruning



158

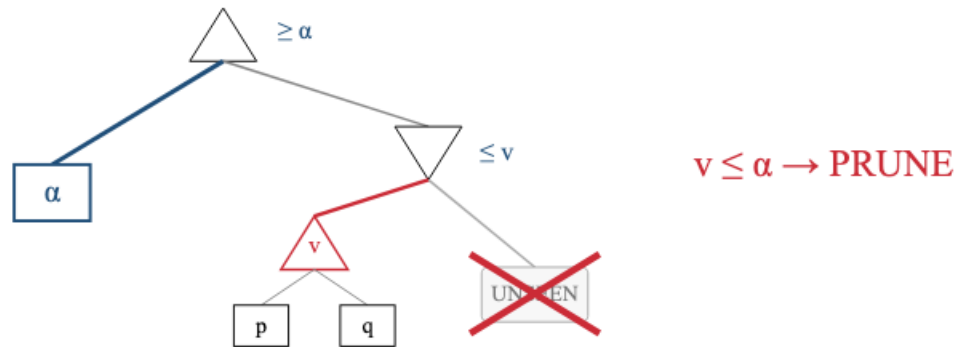
Speaker notes

The MIN node still has another sibling that has not been searched.

MIN Pruning

Slide 158 of 193 · Reveal 2

MIN Pruning



158

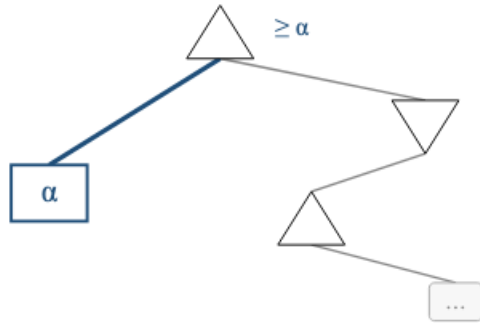
Speaker notes

When v is no greater than α , MIN can keep this branch at v or push it lower. The unseen branch cannot change MAX's choice, so prune it.

MIN Pruning

Slide 158 of 193 · Reveal 3

MIN Pruning at Greater Depth



159

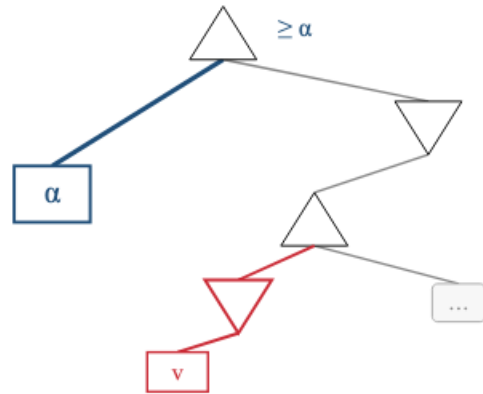
MIN Pruning at Greater Depth

Slide 159 of 193 · Initial

Speaker notes

The root's completed left branch gives MAX an alternative worth alpha.

MIN Pruning at Greater Depth



- α : MAX's best explored option on the path from the current node to the root

159

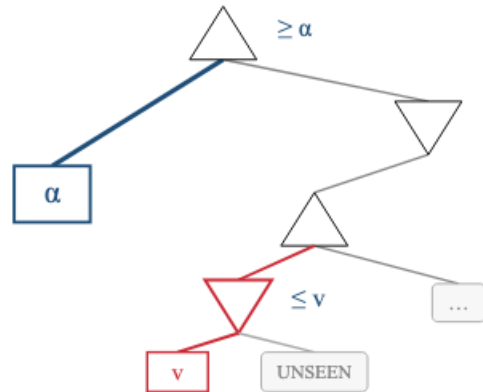
Speaker notes

Search one child of the lower MAX node. It reaches another MIN node, whose first child returns v . Alpha is MAX's best explored option on the path from this MIN node back to the root.

MIN Pruning at Greater Depth

Slide 159 of 193 · Reveal 1

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

159

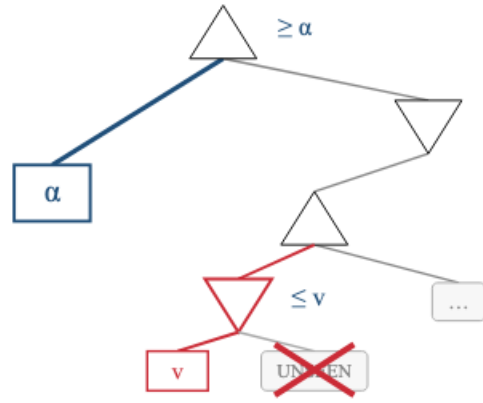
Speaker notes

The deeper MIN node has an unseen sibling, so its final value can be at most v .

MIN Pruning at Greater Depth

Slide 159 of 193 · Reveal 2

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

$v \leq \alpha \rightarrow \text{PRUNE}$

159

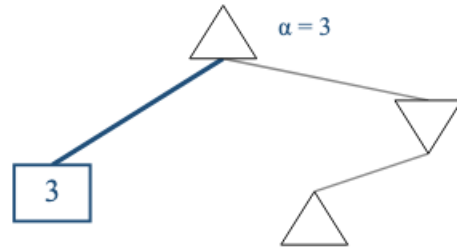
Speaker notes

If v is no greater than α , skip that sibling. The same cutoff works at this depth.

MIN Pruning at Greater Depth

Slide 159 of 193 · Reveal 3

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

160

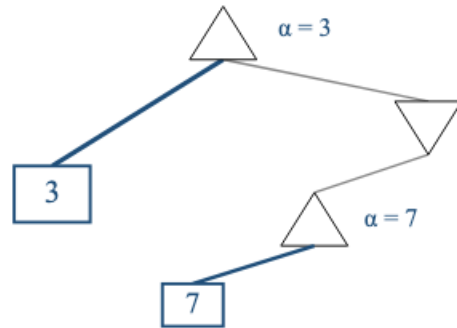
Speaker notes

The root has a completed option worth 3, so this path starts with alpha 3.

MIN Pruning at Greater Depth

Slide 160 of 193 · Initial

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

160

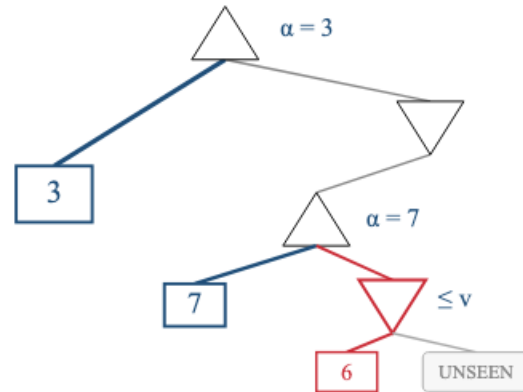
Speaker notes

The lower MAX node completes its own option worth 7. It raises alpha from 3 to 7.

MIN Pruning at Greater Depth

Slide 160 of 193 · Reveal 1

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

160

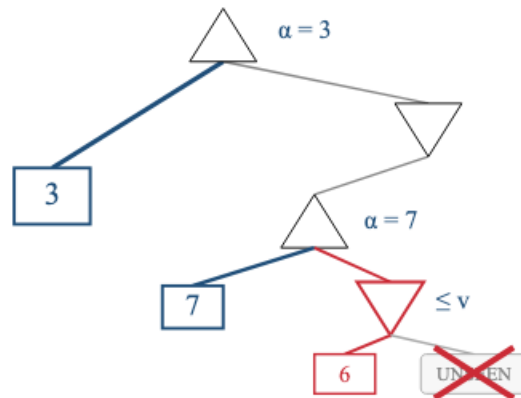
Speaker notes

The next child is a MIN node. Its completed child returns $v = 6$, so the MIN node can finish at no more than v . Its alpha is the best explored MAX option on the path back to the root.

MIN Pruning at Greater Depth

Slide 160 of 193 · Reveal 2

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

$v \leq \alpha \rightarrow \text{PRUNE}$

160

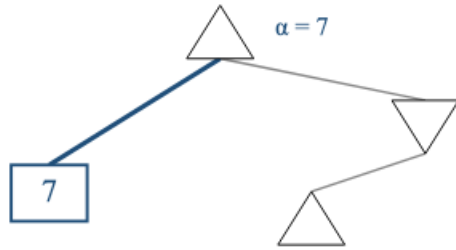
Speaker notes

Here $v = 6$ and $\alpha = 7$. The lower MAX node already has 7, so the MIN node's unseen sibling cannot affect its choice. Prune it.

MIN Pruning at Greater Depth

Slide 160 of 193 · Reveal 3

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

161

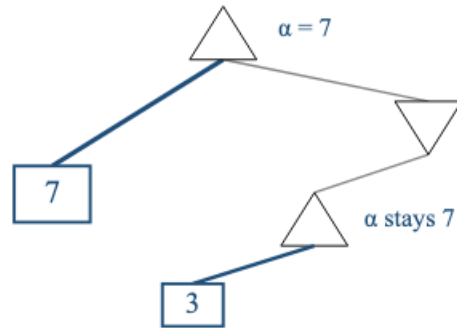
Speaker notes

Now swap the two completed options. The root already has 7, so alpha starts at 7.

MIN Pruning at Greater Depth

Slide 161 of 193 · Initial

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

161

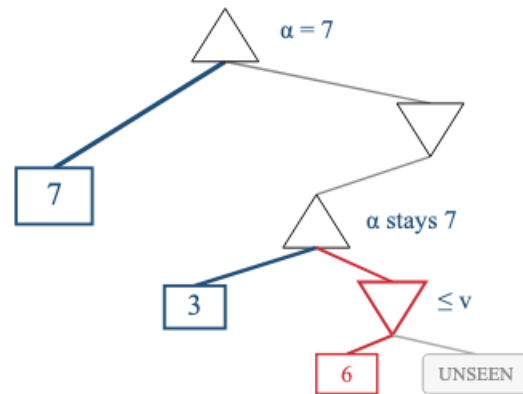
Speaker notes

The lower MAX node finds 3. That does not improve the inherited alpha, which stays at 7.

MIN Pruning at Greater Depth

Slide 161 of 193 · Reveal 1

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

161

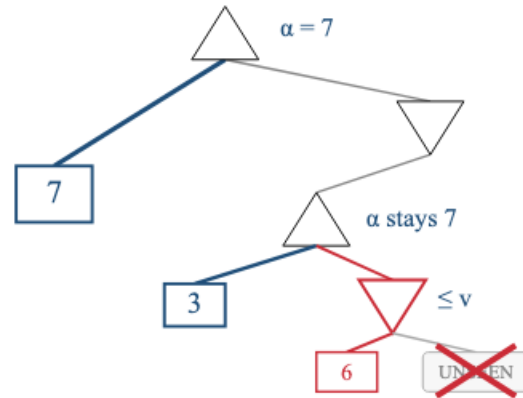
Speaker notes

The next MIN node sees $v = 6$ and has an unseen sibling. Its value can only stay at 6 or fall.

MIN Pruning at Greater Depth

Slide 161 of 193 · Reveal 2

MIN Pruning at Greater Depth



α : MAX's best explored option
on the path from the current node to the root

$v \leq \alpha \rightarrow \text{PRUNE}$

161

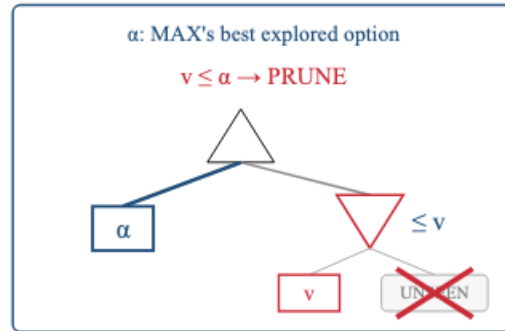
Speaker notes

Because v is no greater than α , the root's option worth 7 makes the remaining search irrelevant. Prune it.

MIN Pruning at Greater Depth

Slide 161 of 193 · Reveal 3

Pruning Conditions



162

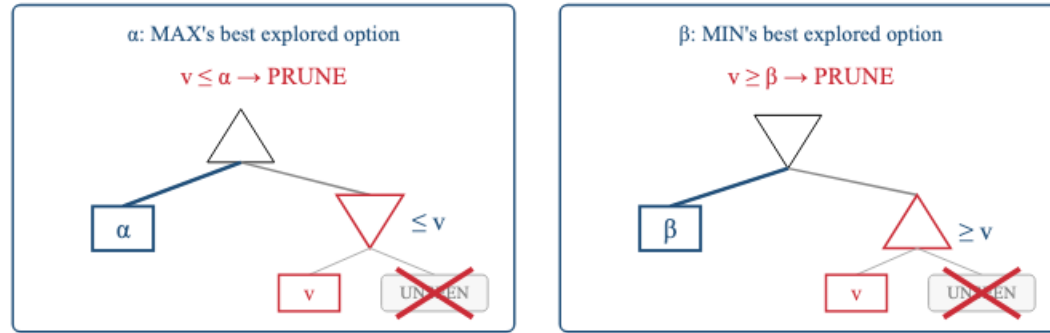
Pruning Conditions

Slide 162 of 193 · Initial

Speaker notes

The left diagram repeats the previous slide: alpha is the best alternative MAX already has. At a MIN node, v no greater than alpha means the unseen child cannot rescue this branch.

Pruning Conditions



162

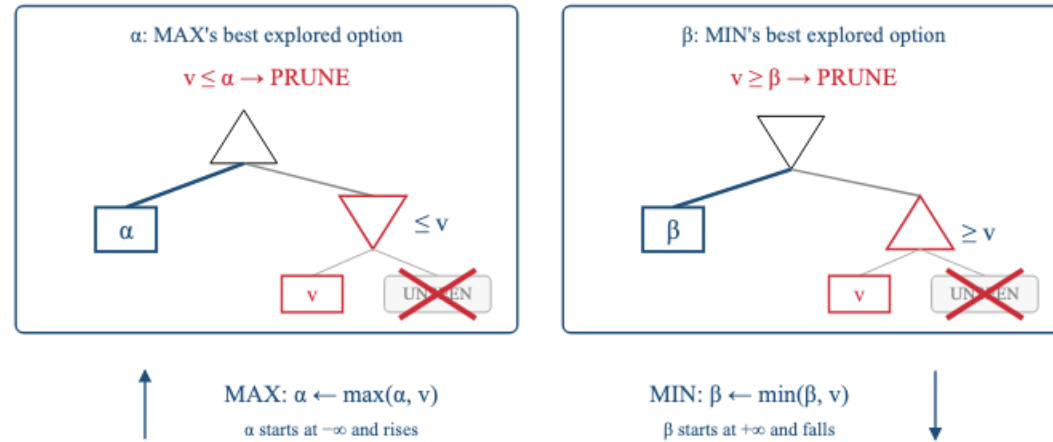
Speaker notes

Beta is the mirror image. If MIN already has an alternative worth beta, a MAX node reaching v at least beta cannot make MIN choose this branch, so its unseen child can be pruned.

Pruning Conditions

Slide 162 of 193 · Reveal 1

Pruning Conditions



162

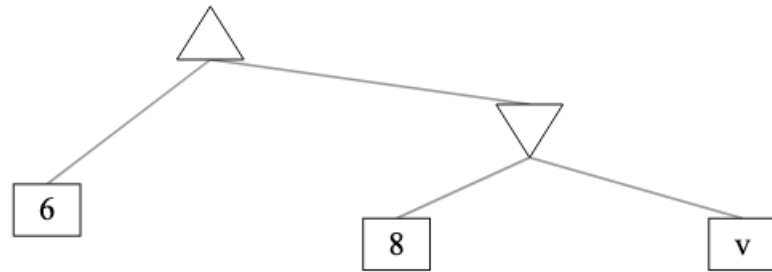
Speaker notes

After each child returns value v , MAX updates alpha to $\max(\alpha, v)$, while MIN updates beta to $\min(\beta, v)$. Alpha starts at negative infinity and can only rise; beta starts at positive infinity and can only fall. Next we will see how ancestors supply the two bounds along one path.

Pruning Conditions

Slide 162 of 193 · Reveal 2

Which Values Can Change the Choice?



163

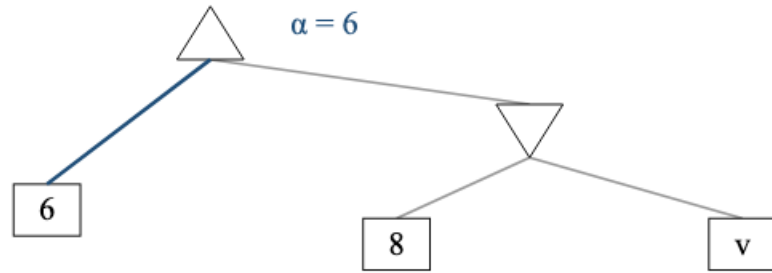
Which Values Can Change the Choice?

Slide 163 of 193 · Initial

Speaker notes

The two completed sibling branches set the alternatives along this search path.

Which Values Can Change the Choice?



163

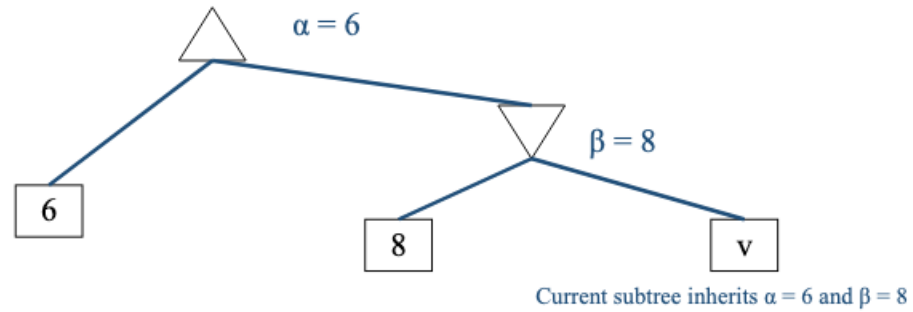
Speaker notes

MAX can already choose 6, so the current path inherits alpha 6.

Which Values Can Change the Choice?

Slide 163 of 193 · Reveal 1

Which Values Can Change the Choice?



163

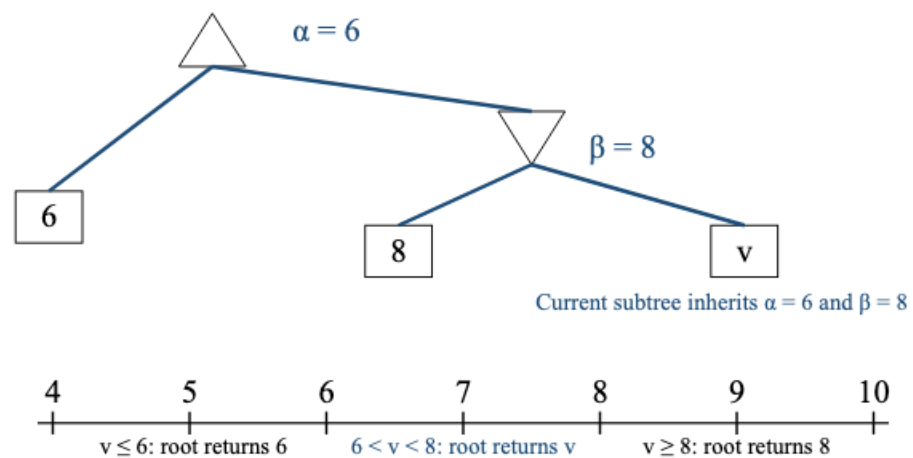
Speaker notes

MIN can already choose 8, so the deeper subtree inherits beta 8 as well.

Which Values Can Change the Choice?

Slide 163 of 193 · Reveal 2

Which Values Can Change the Choice?



163

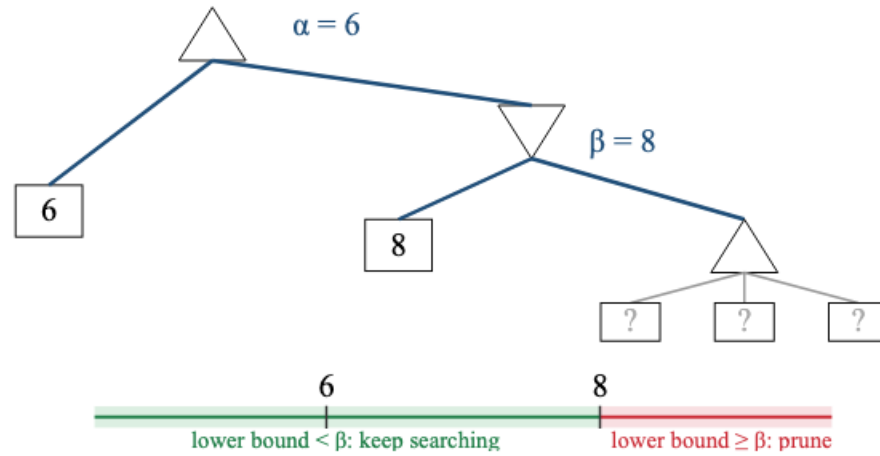
Speaker notes

Hover along the axis to vary v ; move away to reset. MIN returns $\min(8, v)$, updating beta; MAX returns $\max(6, \text{beta})$, updating alpha. Equal values retain the earlier alternative.

Which Values Can Change the Choice?

Slide 163 of 193 · Reveal 3

MIN Pruning



164

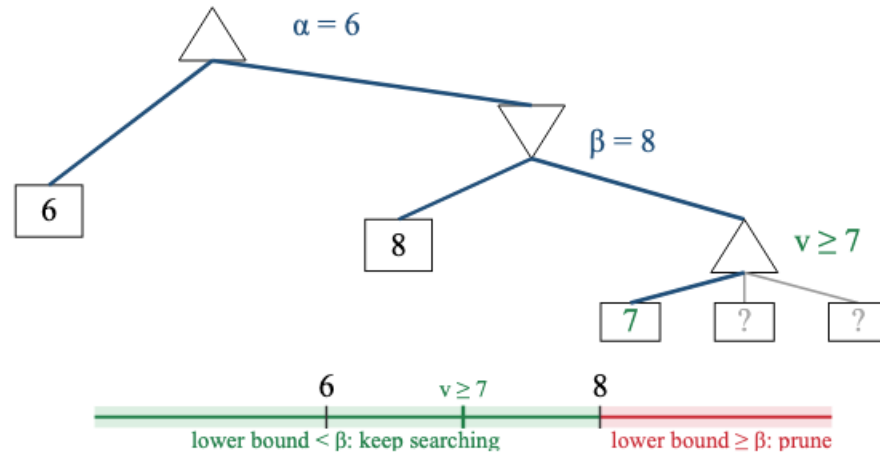
Speaker notes

The root's alpha 6 and MIN's beta 8 come from earlier alternatives. The value v of this deeper subtree is still unknown.

MIN Pruning

Slide 164 of 193 · Initial

MIN Pruning



164

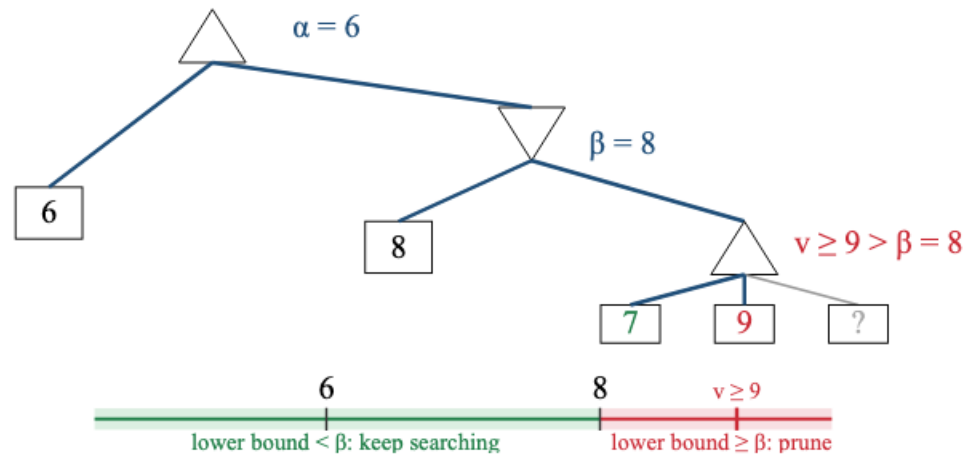
Speaker notes

The first child gives 7, so v is at least 7. The bound is still in the green search region, below beta 8.

MIN Pruning

Slide 164 of 193 · Reveal 1

MIN Pruning



164

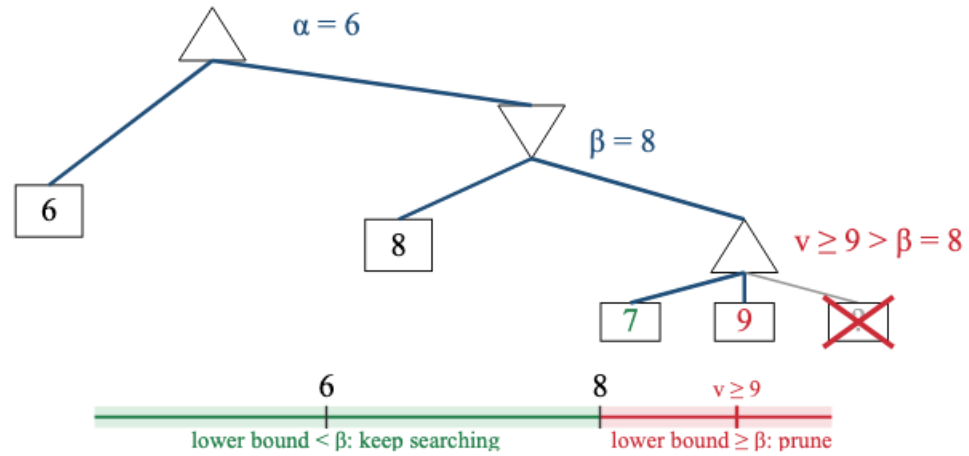
Speaker notes

The next child gives 9, so v is at least 9. Its lower bound reaches the red cutoff region, and MIN will keep its alternative 8.

MIN Pruning

Slide 164 of 193 · Reveal 2

MIN Pruning



164

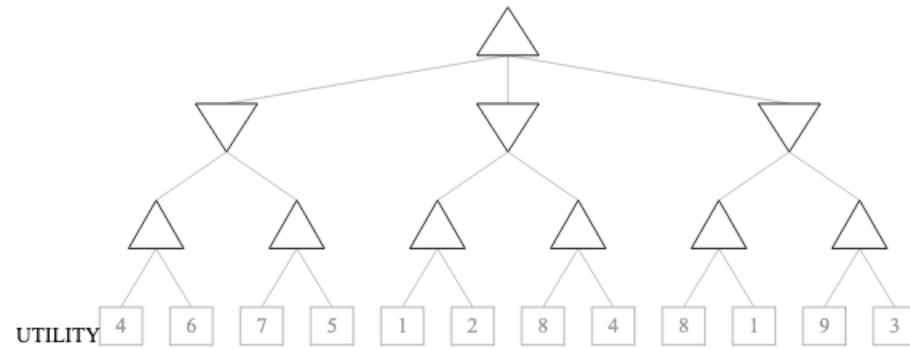
Speaker notes

Skip the last child. v could exceed 9; its lower bound is enough to justify this cutoff.

MIN Pruning

Slide 164 of 193 · Reveal 3

Initial Bounds



Root $\alpha = -\infty$ · Current $\beta = +\infty$

165

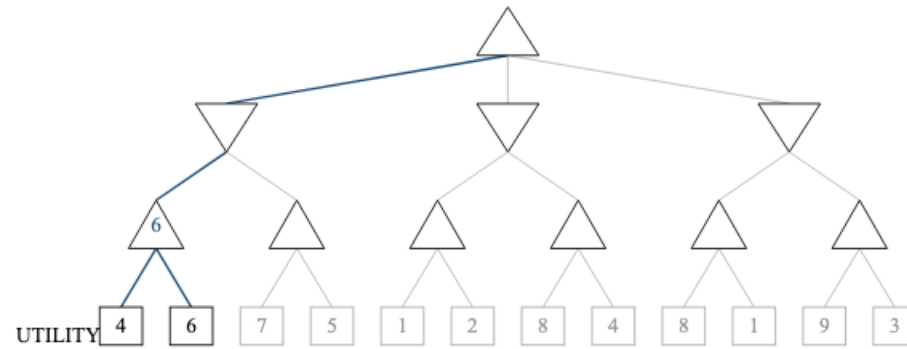
Speaker notes

Search proceeds depth first from left to right.

Initial Bounds

Slide 165 of 193 · Initial

First MAX Returns 6



Left MIN updates β to 6

166

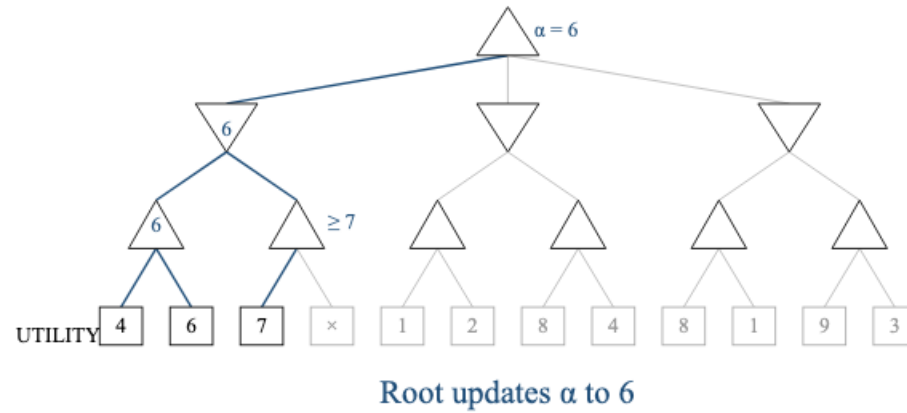
Speaker notes

The first MAX node compares 4 and 6.

First MAX Returns 6

Slide 166 of 193 · Initial

Left MIN Sets Alpha to 6



167

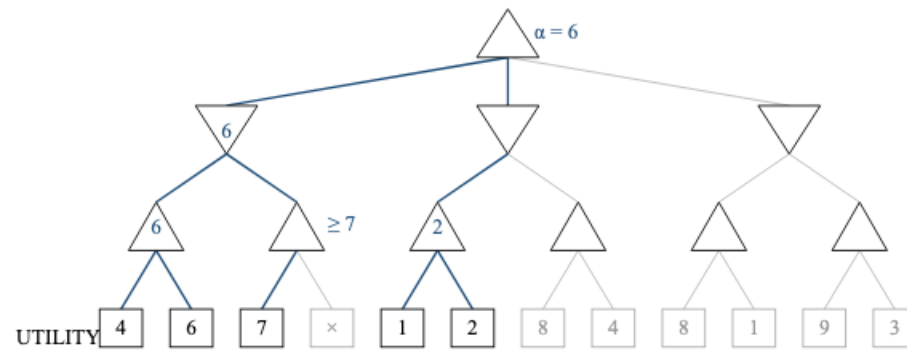
Speaker notes

The second MAX reaches $7 \geq \text{beta } 6$, so it prunes leaf 5. Left MIN returns 6, setting root alpha to 6.

Left MIN Sets Alpha to 6

Slide 167 of 193 · Initial

Middle MAX Returns 2



168

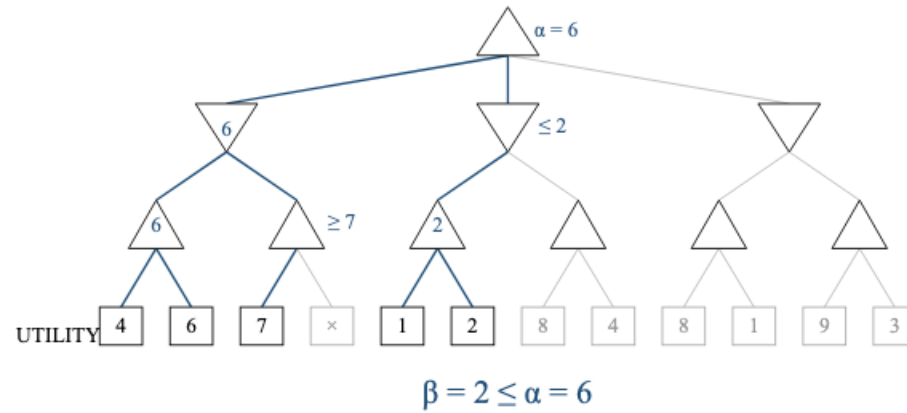
Speaker notes

The middle branch first produces $\max(1, 2) = 2$.

Middle MAX Returns 2

Slide 168 of 193 · Initial

Middle MIN Reaches the Cutoff



169

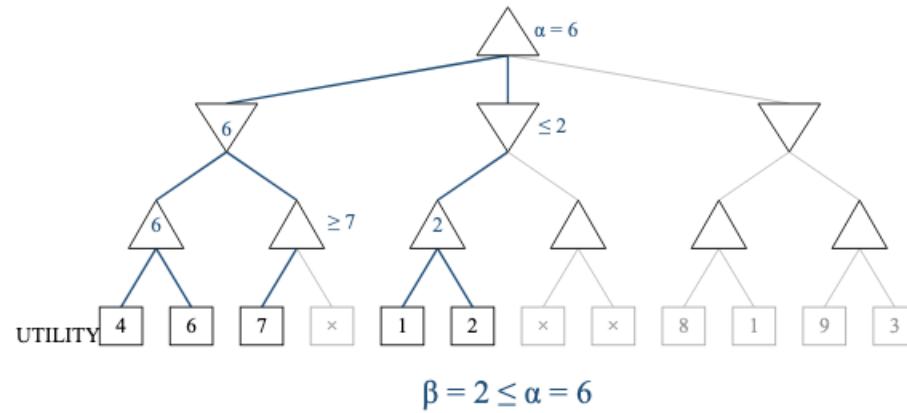
Speaker notes

At this MIN node, $\beta = 2 \leq \alpha = 6$.

Middle MIN Reaches the Cutoff

Slide 169 of 193 · Initial

Middle Branch Pruned



170

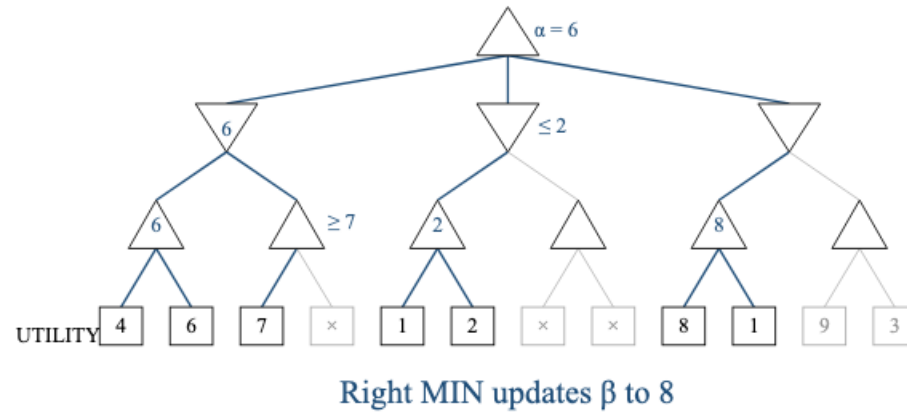
Speaker notes

Nothing else below this MIN node can make MAX prefer it.

Middle Branch Pruned

Slide 170 of 193 · Initial

Right MAX Returns 8



171

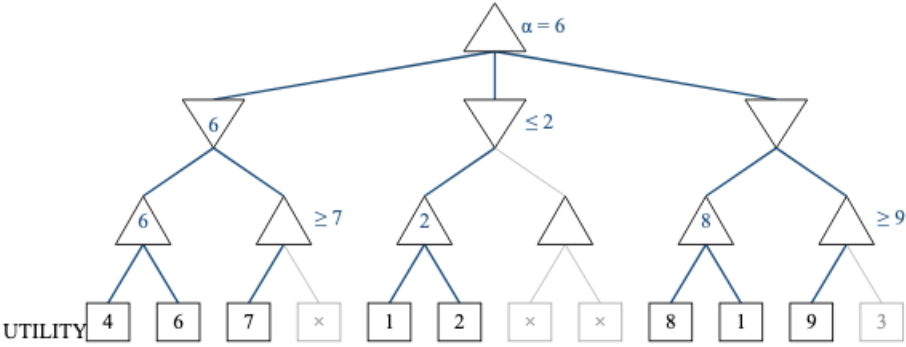
Right MAX Returns 8

Slide 171 of 193 · Initial

Speaker notes

The first MAX child gives the right MIN node $\beta = 8$.

Next MAX Sees 9



Current $v = 9$ · Right MIN $\beta = 8$

172

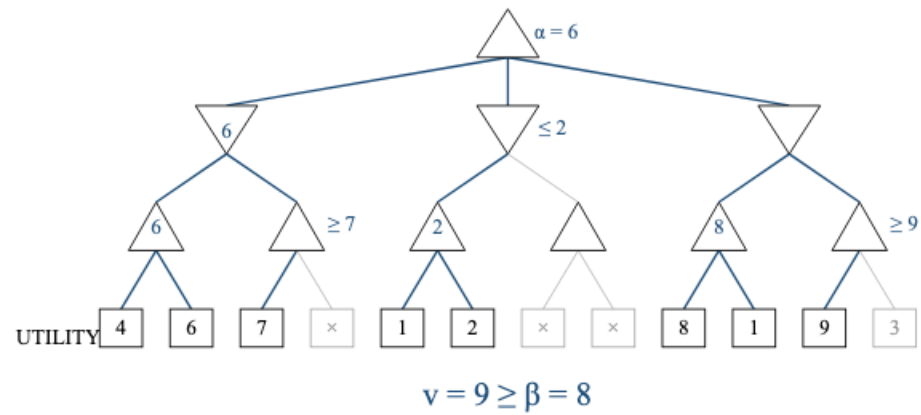
Speaker notes

This MAX node now has $v = 9$.

Next MAX Sees 9

Slide 172 of 193 · Initial

Right MAX Reaches the Cutoff



173

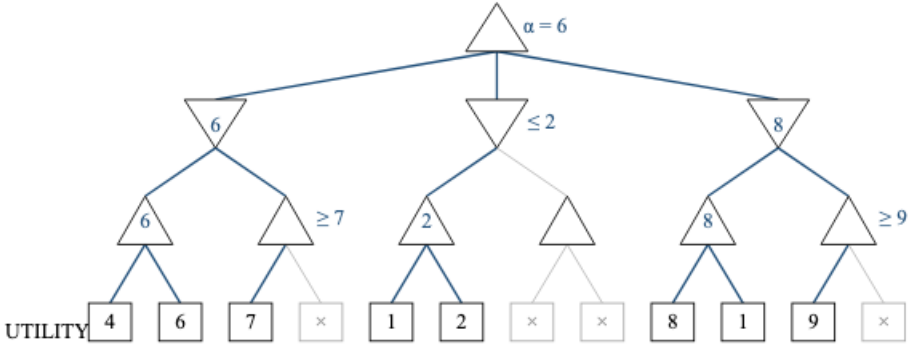
Right MAX Reaches the Cutoff

Slide 173 of 193 · Initial

Speaker notes

Because $9 \geq \beta = 8$, any remaining child can be pruned.

Right MIN Returns 8



Root $\alpha = 6$ · Right value = 8

174

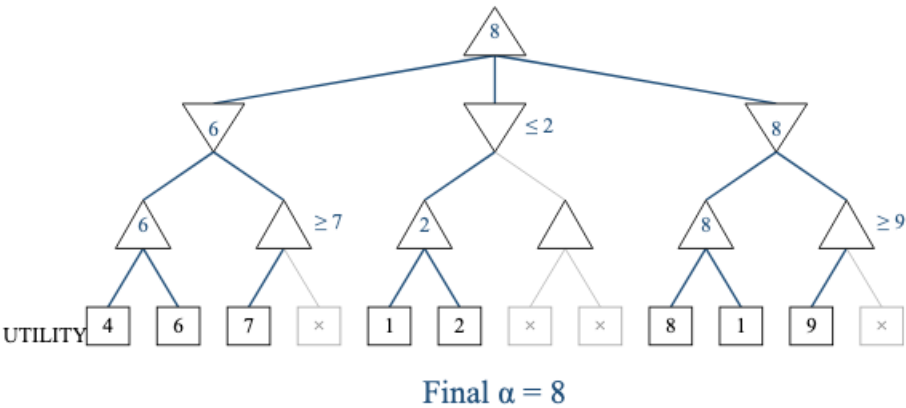
Speaker notes

The pruned leaf cannot reduce the first child's value of 8.

Right MIN Returns 8

Slide 174 of 193 · Initial

Root Chooses 8



175

Speaker notes

Alpha-beta returns the same minimax value: 8.

Root Chooses 8

Slide 175 of 193 · Initial

Alpha–Beta Search in Python

```
main.py
1 def alphabeta(tree, alpha, beta, max_turn):
2     if not isinstance(tree, list): return tree
3     value = -float("inf") if max_turn else float("inf")
4     for child in tree:
5         child_value = alphabeta(child, alpha, beta, not max_turn)
6         if max_turn:
7             value = max(value, child_value)
8             alpha = max(alpha, value)
9         else:
10            value = min(value, child_value)
11            beta = min(beta, value)
12            if alpha >= beta: break
13    return value
14 tree = [[[4, 6], [7, 5]], [[1, 2], [8, 4]], [[8, 1], [9, 3]]]
15 print(alphabeta(tree, -float("inf"), float("inf"), True)) # 8
16
```

176

Alpha–Beta Search in Python

Slide 176 of 193 · Initial

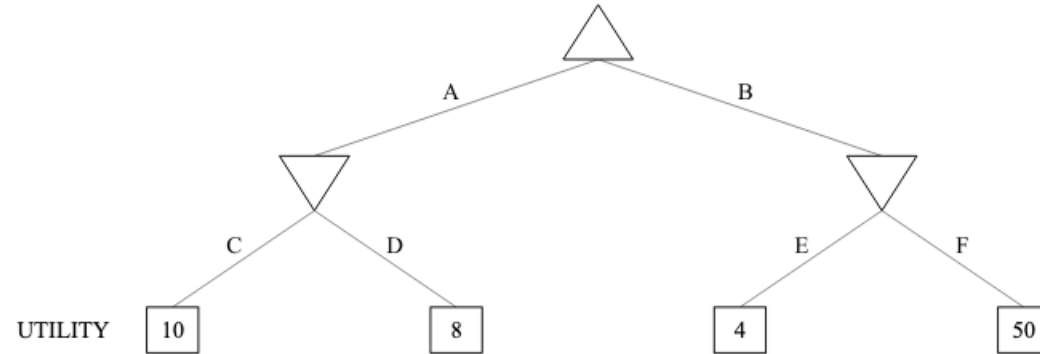
Speaker notes

Run the same tree from the trace. Its root value is 8. MAX updates alpha; MIN updates beta. Stop when the bounds meet.

Task: Alpha–Beta Pruning

TASK

Search Left to Right · Which Branch Is Pruned?



177

Task: Alpha–Beta Pruning

Slide 177 of 193 · Initial

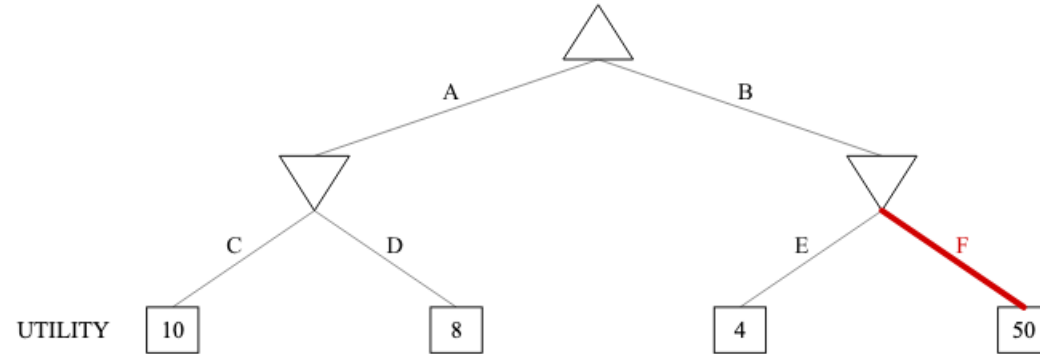
Speaker notes

Search left to right. The left branch returns 8, setting alpha to 8.

Solution: Alpha–Beta Pruning

SOLUTION

Search Left to Right · Which Branch Is Pruned?



Pruned Branch: F · Left MIN Gives 8, Then Right MIN Finds $4 \leq \alpha$

177

Speaker notes

On the right, MIN sees 4. Prune branch F, leading to 50, because 4 is no greater than alpha 8.

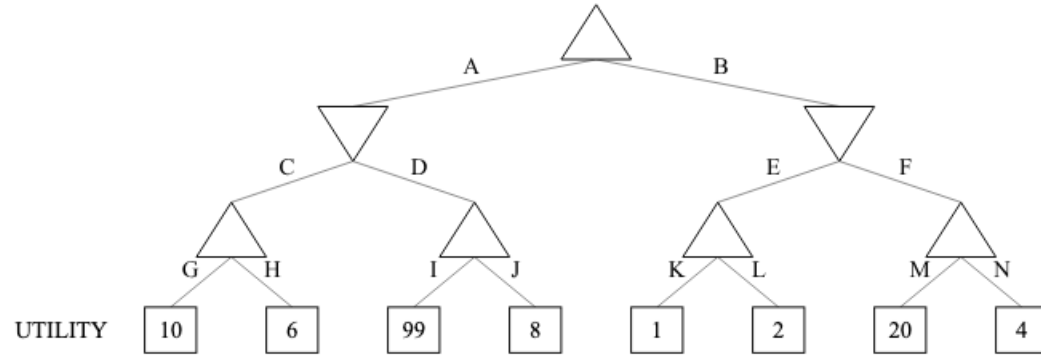
Task: Alpha–Beta Pruning

Slide 177 of 193 · Reveal 1

Task: Deeper Alpha–Beta Pruning

TASK

Search Left to Right · Which Branches Are Pruned?



178

Task: Deeper Alpha–Beta Pruning

Slide 178 of 193 · Initial

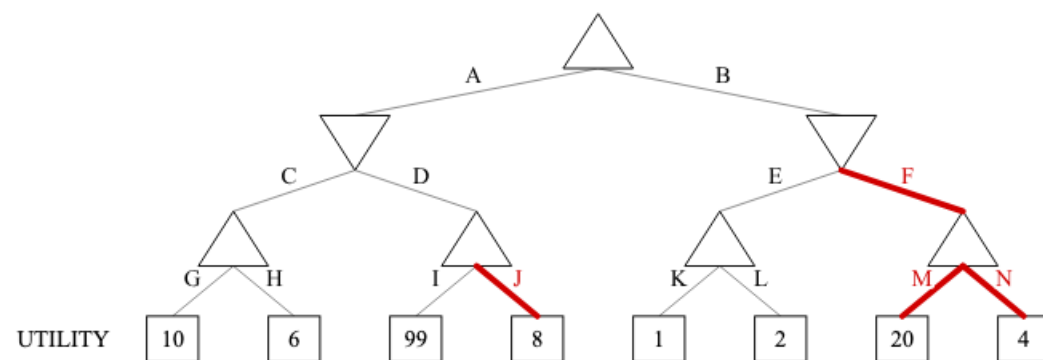
Speaker notes

Search left to right. Select every labelled branch that is not explored.

Solution: Deeper Alpha–Beta Pruning

SOLUTION

Search Left to Right · Which Branches Are Pruned?



Pruned Branches: F, J, M, N

178

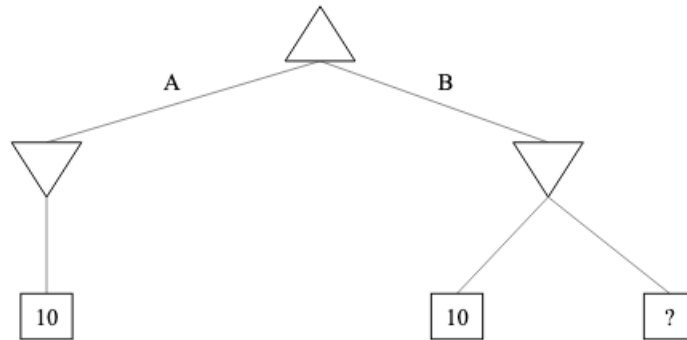
Speaker notes

J is pruned when $99 \geq \beta = 10$. Left MIN returns 10, setting root α to 10. On the right, $\beta = 2 \leq \alpha = 10$, so F and its descendant branches M and N are not explored.

Task: Deeper Alpha–Beta Pruning

Slide 178 of 193 · Reveal 1

A Bound Is Not an Exact Value



Search Left to Right · Which Move Should MAX Keep?

179

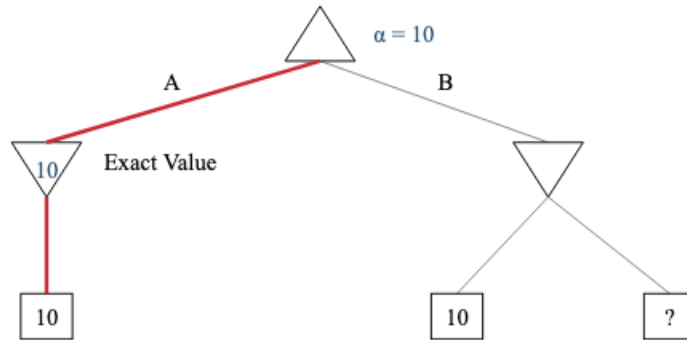
Speaker notes

This is the original example: A has one leaf, 10. B has 10 and one leaf we will not inspect.

A Bound Is Not an Exact Value

Slide 179 of 193 · Initial

A Bound Is Not an Exact Value



A Returns 10 · Keep A and Set $\alpha = 10$

179

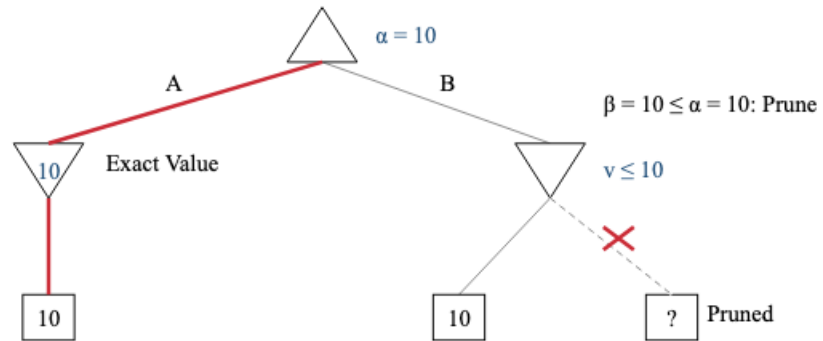
Speaker notes

A returns an exact 10. Keep A as the best move and pass alpha 10 into B.

A Bound Is Not an Exact Value

Slide 179 of 193 · Reveal 1

A Bound Is Not an Exact Value



B Returns 10 as a Bound · This Does Not Prove a Tie

179

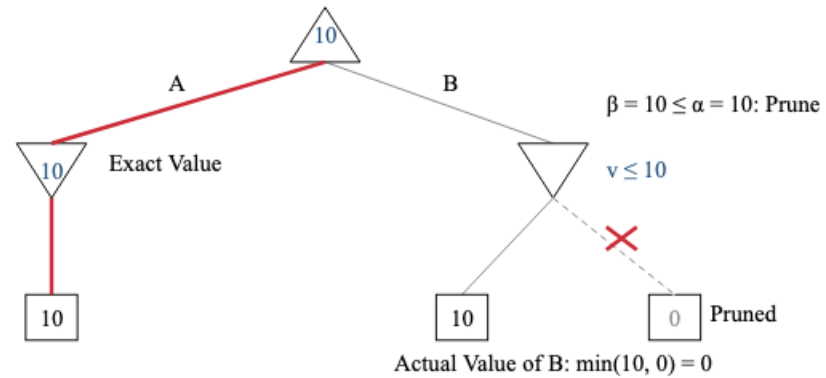
Speaker notes

B sees 10, so beta equals alpha and the other leaf is pruned. The code returns 10, but this means only that B is at most 10.

A Bound Is Not an Exact Value

Slide 179 of 193 · Reveal 2

A Bound Is Not an Exact Value



Keep A · Replace the Best Move Only on a Strict Improvement

179

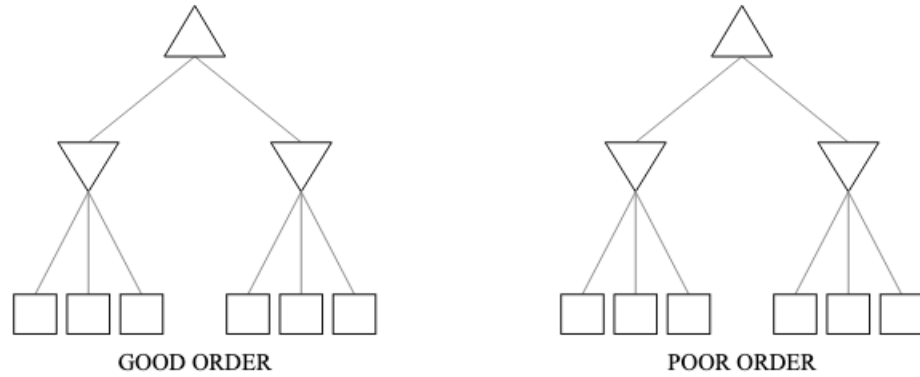
Speaker notes

The unvisited leaf is actually 0, so B is worth 0. The root value is still exactly 10. Keep A on equality; replacing it with B would mistake a bound for an exact tie.

A Bound Is Not an Exact Value

Slide 179 of 193 · Reveal 3

Move Ordering Controls Pruning



180

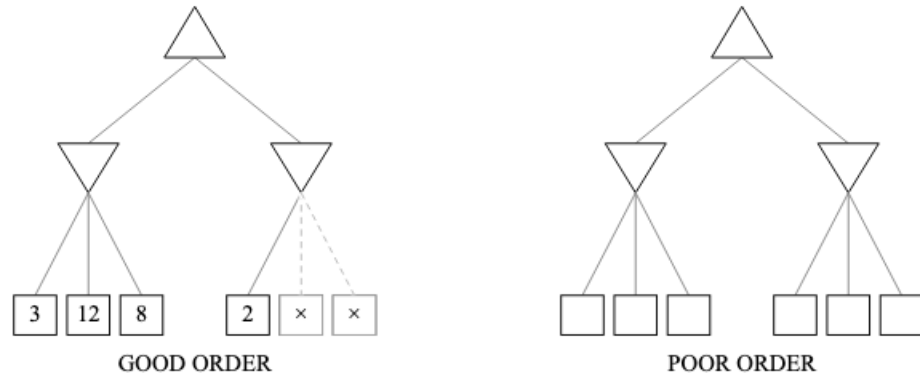
Speaker notes

Both diagrams have the same game tree but explore the moves in different orders.

Move Ordering Controls Pruning

Slide 180 of 193 · Initial

Move Ordering Controls Pruning



180

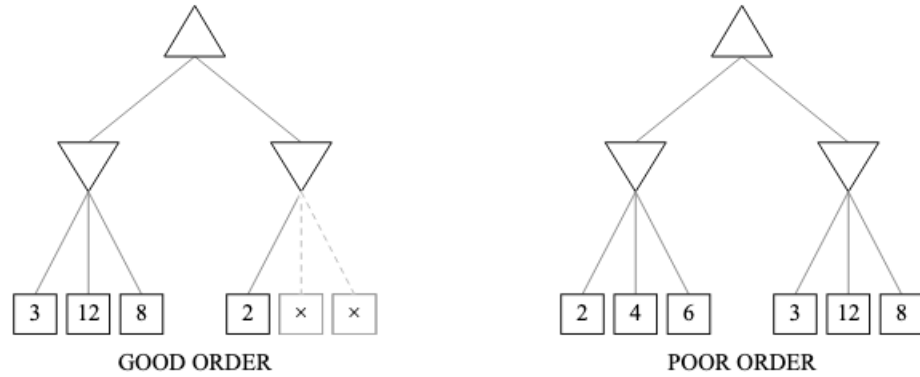
Speaker notes

Good ordering finds value 3 on the left; the right MIN node sees 2, so its other two leaves are pruned.

Move Ordering Controls Pruning

Slide 180 of 193 · Reveal 1

Move Ordering Controls Pruning



180

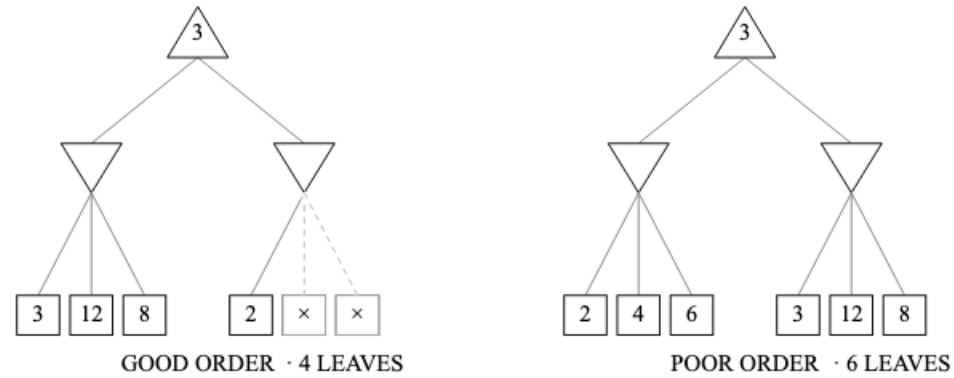
Move Ordering Controls Pruning

Slide 180 of 193 · Reveal 2

Speaker notes

Poor ordering begins with the weaker branch and must examine all six leaves.

Move Ordering Controls Pruning



Same Minimax Value: 3 · Different Amount of Search

180

Speaker notes

Both root values are 3. Ordering changes the amount of search, not the minimax decision.

Move Ordering Controls Pruning

Slide 180 of 193 · Reveal 3

Good Ordering Searches Much Less

Minimax: $O(b^m)$

Knuth & Moore (1975), An Analysis of Alpha-Beta Pruning

181

Speaker notes

With ideal move ordering, alpha-beta can reduce the effective exponent by about half.

Good Ordering Searches Much Less

Slide 181 of 193 · Initial

Good Ordering Searches Much Less

Minimax: $O(b^m)$

Ideal Alpha-Beta Ordering: $O(b^{m/2})$

Knuth & Moore (1975), An Analysis of Alpha-Beta Pruning

181

Speaker notes

That means the same node budget can search approximately twice as deep.

Good Ordering Searches Much Less

Slide 181 of 193 · Reveal 1

Good Ordering Searches Much Less

Minimax: $O(b^m)$

Ideal Alpha–Beta Ordering: $O(b^{m/2})$

Roughly Twice the Search Depth for the Same Node Budget

Knuth & Moore (1975), An Analysis of Alpha-Beta Pruning

181

Speaker notes

Poor ordering can remove most of the benefit; the worst-case asymptotic time is unchanged.

Good Ordering Searches Much Less

Slide 181 of 193 · Reveal 2

Good Ordering Searches Much Less

Minimax: $O(b^m)$

Ideal Alpha–Beta Ordering: $O(b^{m/2})$

Roughly Twice the Search Depth for the Same Node Budget

Example: $b = 10$ and $m = 6$ Gives 10^6 Leaves vs About 10^3

Knuth & Moore (1975), An Analysis of Alpha-Beta Pruning

181

Speaker notes

Classic analysis: Knuth and Moore, 1975.

Good Ordering Searches Much Less

Slide 181 of 193 · Reveal 3

Good Ordering Searches Much Less

Minimax: $O(b^m)$

Ideal Alpha-Beta Ordering: $O(b^{m/2})$

Roughly Twice the Search Depth for the Same Node Budget

Example: $b = 10$ and $m = 6$ Gives 10^6 Leaves vs About 10^3

Worst Case Remains $O(b^m)$

Knuth & Moore (1975), An Analysis of Alpha-Beta Pruning

181

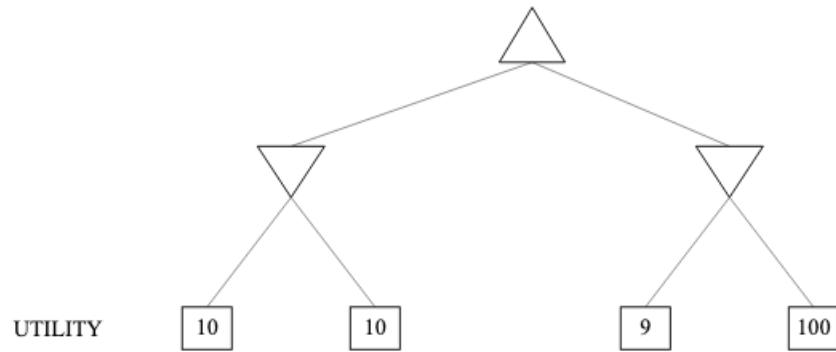
Speaker notes

No speaker notes for this stage.

Good Ordering Searches Much Less

Slide 181 of 193 · Reveal 4

Worst Case Uses MIN's Choice



182

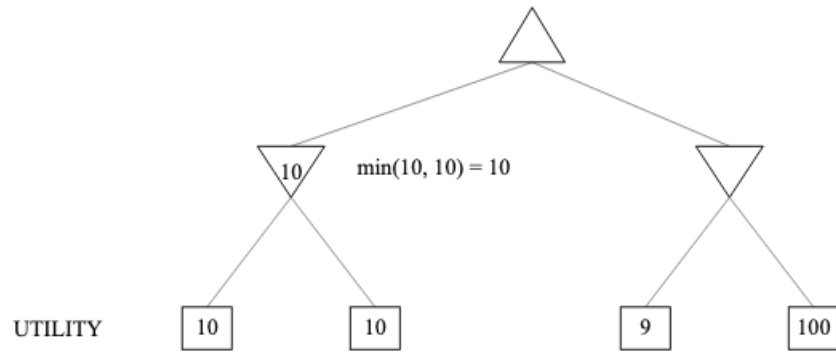
Worst Case Uses MIN's Choice

Slide 182 of 193 · Initial

Speaker notes

This reuses minimax before introducing chance: MIN controls each reply.

Worst Case Uses MIN's Choice



182

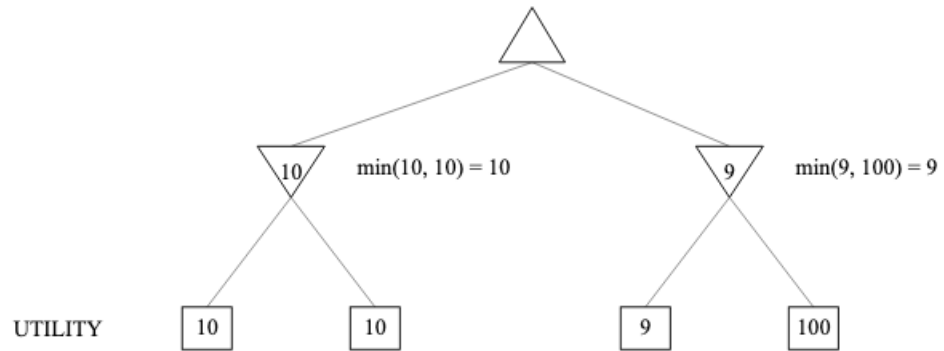
Speaker notes

The left MIN node backs up $\min(10, 10) = 10$.

Worst Case Uses MIN's Choice

Slide 182 of 193 · Reveal 1

Worst Case Uses MIN's Choice



182

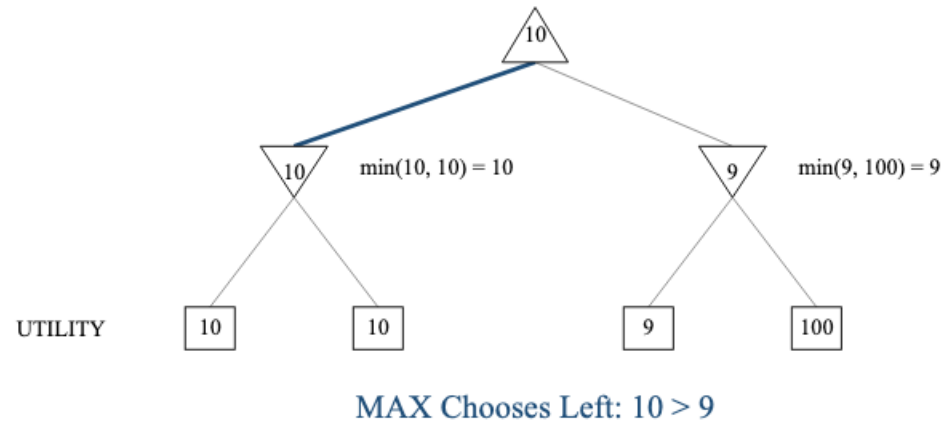
Speaker notes

The right MIN node backs up $\min(9, 100) = 9$; MIN can choose 9 despite the 100.

Worst Case Uses MIN's Choice

Slide 182 of 193 · Reveal 2

Worst Case Uses MIN's Choice



182

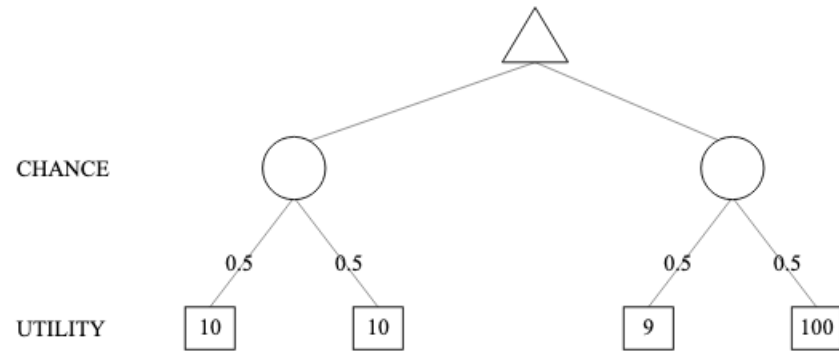
Speaker notes

MAX therefore chooses left and backs up 10 at the root.

Worst Case Uses MIN's Choice

Slide 182 of 193 · Reveal 3

Chance Uses the Expected Value



183

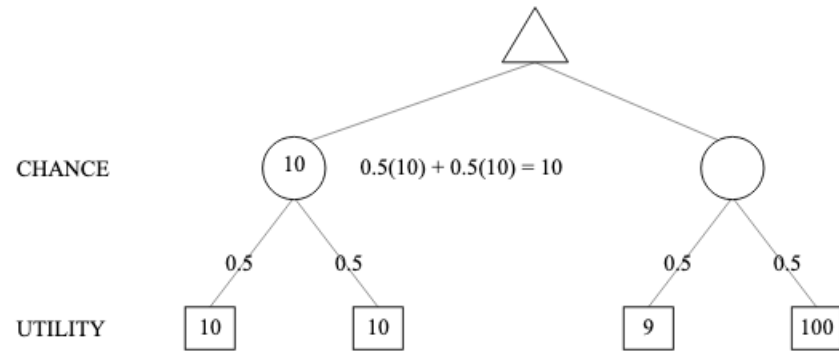
Chance Uses the Expected Value

Slide 183 of 193 · Initial

Speaker notes

Keep the same outcomes, but replace adversarial MIN by equal-probability chance nodes.

Chance Uses the Expected Value



183

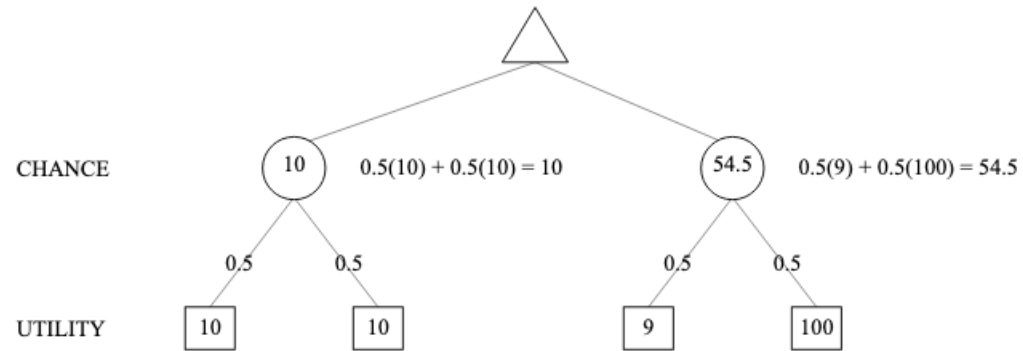
Speaker notes

The left chance node backs up $0.5(10) + 0.5(10) = 10$.

Chance Uses the Expected Value

Slide 183 of 193 · Reveal 1

Chance Uses the Expected Value



183

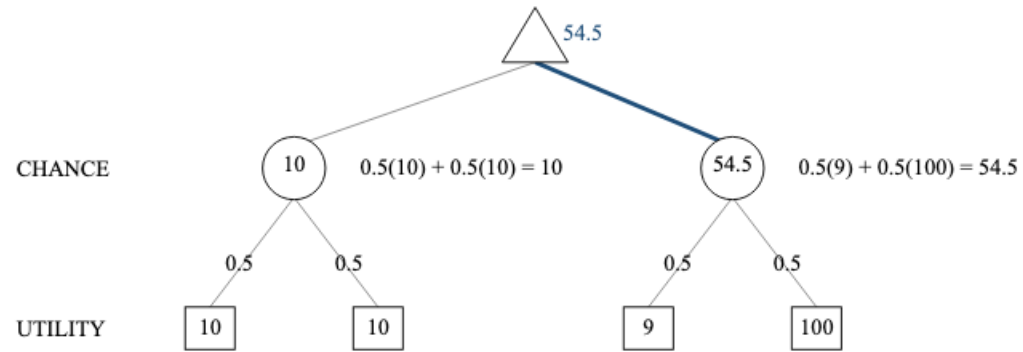
Speaker notes

The right chance node backs up $0.5(9) + 0.5(100) = 54.5$. Here the 100 affects the average.

Chance Uses the Expected Value

Slide 183 of 193 · Reveal 2

Chance Uses the Expected Value



MAX Chooses Right: $54.5 > 10$

183

Speaker notes

With this chance model, MAX chooses right and backs up 54.5.

Chance Uses the Expected Value

Slide 183 of 193 · Reveal 3

Why Chance Nodes Appear

Explicit randomness	Dice, shuffled cards, noisy outcomes
Agent model	An opponent policy represented by probabilities
Action failure	The intended action may not occur

184

Why Chance Nodes Appear

Slide 184 of 193 · Initial

Speaker notes

Chance nodes model uncertainty over successor states.

Why Chance Nodes Appear

- Explicit randomness Dice, shuffled cards, noisy outcomes
- Agent model An opponent policy represented by probabilities
- Action failure The intended action may not occur

A Chance Node Needs a Probability Model

184

Speaker notes

Probabilities may come from game rules, data, or an explicit model of another agent.

Why Chance Nodes Appear

Slide 184 of 193 · Reveal 1

Why Chance Nodes Appear

Explicit randomness	Dice, shuffled cards, noisy outcomes
Agent model	An opponent policy represented by probabilities
Action failure	The intended action may not occur

A Chance Node Needs a Probability Model

Expected Value Is Only as Good as That Model

184

Speaker notes

Without justified probabilities, an expected value is not justified either.

Why Chance Nodes Appear

Slide 184 of 193 · Reveal 2

Expectimax Search



185

Expectimax Search

Slide 185 of 193 · Initial

Speaker notes

The previous slides justified probability models. Expectimax now specifies how to search with one.

Expectimax Search

MAX Node

Take the Largest Child Value



185

Speaker notes

At MAX nodes, keep the same rule as minimax: choose the largest child value.

Expectimax Search

Slide 185 of 193 · Reveal 1

Expectimax Search

MAX Node

Take the Largest Child Value

CHANCE Node

Take the Probability-Weighted Average



185

Speaker notes

At chance nodes, use the probability-weighted average instead of a minimum.

Expectimax Search

Slide 185 of 193 · Reveal 2

Expectimax Search

MAX Node

Take the Largest Child Value

CHANCE Node

Take the Probability-Weighted Average

Terminal Node

Return Utility



185

Speaker notes

At terminal nodes, return utility exactly as before.

Expectimax Search

Slide 185 of 193 · Reveal 3

Expectimax Search

MAX Node

Take the Largest Child Value

CHANCE Node

Take the Probability-Weighted Average

Terminal Node

Return Utility



$$\text{CHANCE: } V(s) = \sum p_i V(s_i)$$

185

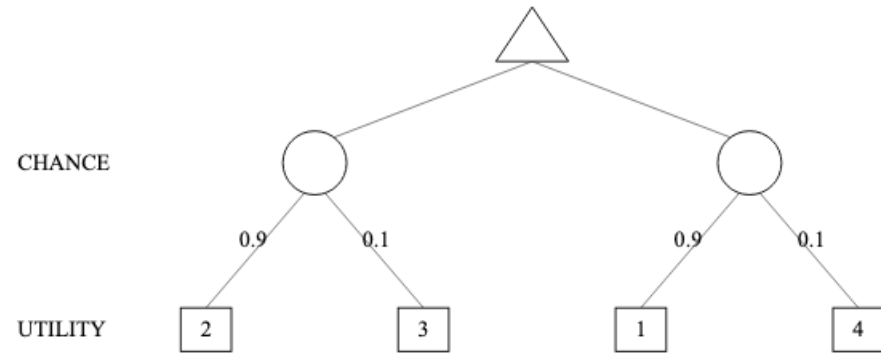
Speaker notes

The result is optimal only relative to the stated chance model.

Expectimax Search

Slide 185 of 193 · Reveal 4

Expectimax Example



186

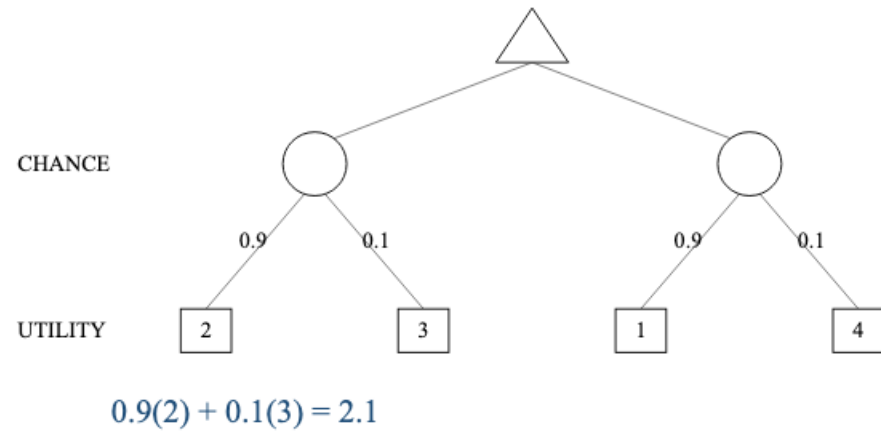
Expectimax Example

Slide 186 of 193 · Initial

Speaker notes

Apply the expectimax rules from the previous slide to this concrete tree.

Expectimax Example



186

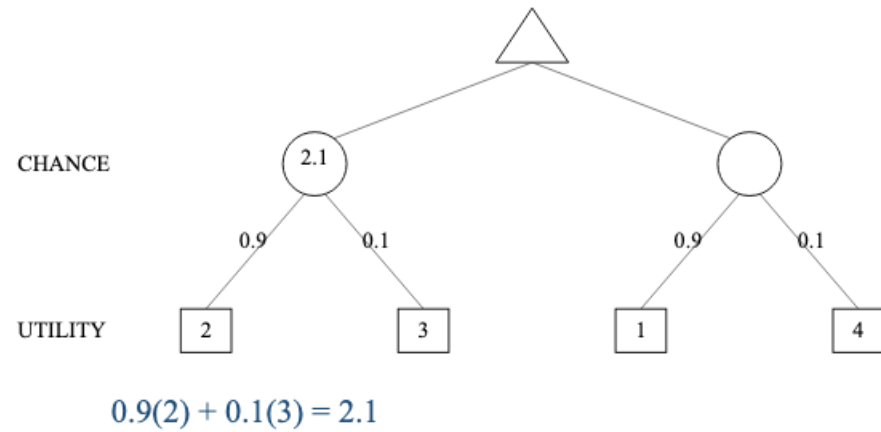
Speaker notes

Calculate the left expected value: $0.9(2) + 0.1(3) = 2.1$.

Expectimax Example

Slide 186 of 193 · Reveal 1

Expectimax Example



186

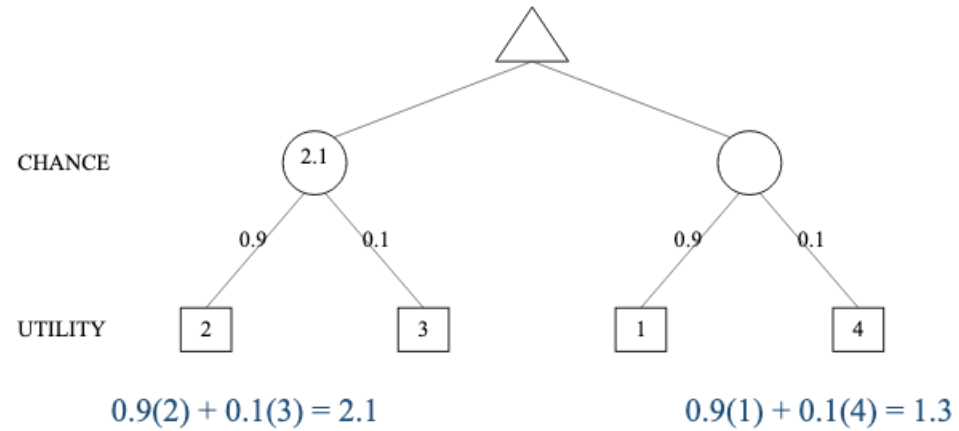
Speaker notes

Back up 2.1 at the left chance node.

Expectimax Example

Slide 186 of 193 · Reveal 2

Expectimax Example



186

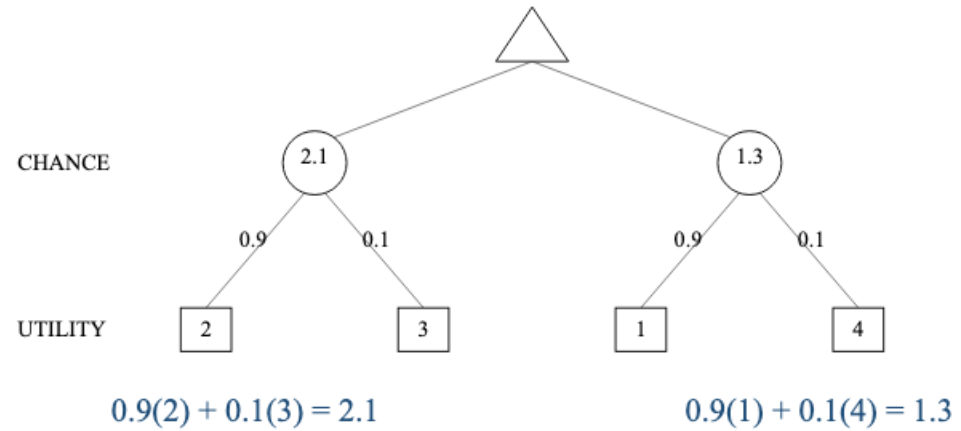
Speaker notes

Calculate the right expected value: $0.9(1) + 0.1(4) = 1.3$.

Expectimax Example

Slide 186 of 193 · Reveal 3

Expectimax Example



186

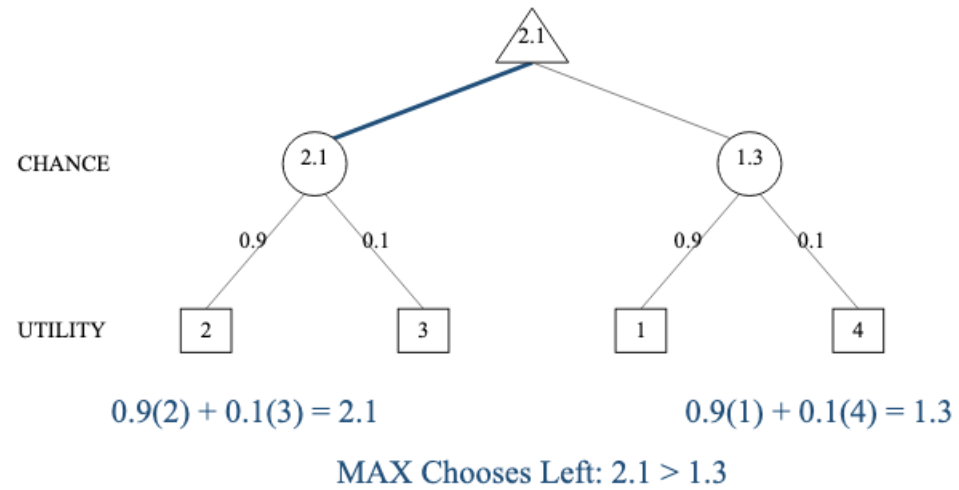
Expectimax Example

Slide 186 of 193 · Reveal 4

Speaker notes

Back up 1.3 at the right chance node.

Expectimax Example



186

Speaker notes

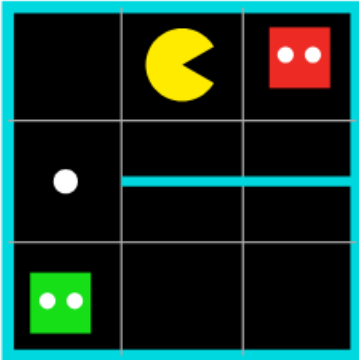
MAX compares the two expectations and chooses the left branch.

Expectimax Example

Slide 186 of 193 · Reveal 5

Task: Minimax or Expectimax?

TASK



Pacman First: WEST or EAST

After WEST and Both Ghosts, Pacman: EAST or SOUTH

Lower Ghost: NORTH or EAST · Upper Ghost: WEST only

Minimax: Worst Legal Ghost Move

Expectimax: Lower Ghost N/E with Probability 0.5 Each

Only Food Eaten: Win · Ghost Collision: Loss

Score: -1 per Pacman action · $+10$ food · $+500$ win · -500 loss

Collision on Food: Loss, No Food Points

187

Speaker notes

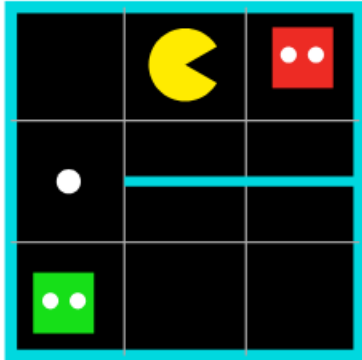
Explain the legal moves and scoring. A collision on the food is a loss with no food reward.

Task: Minimax or Expectimax?

Slide 187 of 193 · Initial

Task: Minimax or Expectimax?

TASK



Pacman First: WEST or EAST

After WEST and Both Ghosts, Pacman: EAST or SOUTH

Lower Ghost: NORTH or EAST · Upper Ghost: WEST only

Minimax: Worst Legal Ghost Move

Expectimax: Lower Ghost N/E with Probability 0.5 Each

Only Food Eaten: Win · Ghost Collision: Loss

Score: -1 per Pacman action · $+10$ food · $+500$ win · -500 loss

Collision on Food: Loss, No Food Points

1. Under minimax, what is Pacman's optimal first action?
2. What value does minimax assign to the root?
3. Under expectimax, what is Pacman's optimal first action?
4. What value does expectimax assign to the root?

187

Speaker notes

Ask for each model's first action and backed-up root value.

Task: Minimax or Expectimax?

Slide 187 of 193 · Reveal 1

Task: Minimax or Expectimax?

TASK



Pacman First: WEST or EAST

After WEST and Both Ghosts, Pacman: EAST or SOUTH

Lower Ghost: NORTH or EAST · Upper Ghost: WEST only

Minimax: Worst Legal Ghost Move

Expectimax: Lower Ghost N/E with Probability 0.5 Each

Only Food Eaten: Win · Ghost Collision: Loss

Score: -1 per Pacman action · $+10$ food · $+500$ win · -500 loss

Collision on Food: Loss, No Food Points

1. Under minimax, what is Pacman's optimal first action?
2. What value does minimax assign to the root?
3. Under expectimax, what is Pacman's optimal first action?
4. What value does expectimax assign to the root?

187

Speaker notes

Reveal the Ed QR code. Ed activity: <https://edstem.org/au/courses/41129/lessons/119797/slides/821339>

Task: Minimax or Expectimax?

Slide 187 of 193 · Reveal 2

Solution: Pacman Game Tree

SOLUTION

188

Solution: Pacman Game Tree

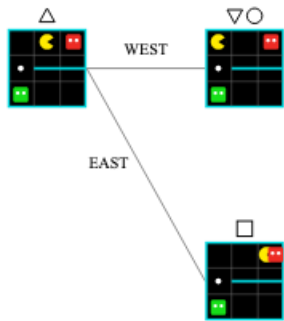
Slide 188 of 193 · Initial

Speaker notes

Show the complete tree. Reveal Pacman's WEST and EAST choices; EAST is an immediate collision and game over.

Solution: Pacman Game Tree

SOLUTION



188

Solution: Pacman Game Tree

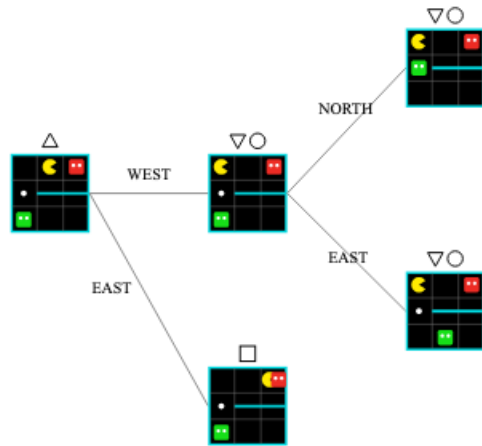
Slide 188 of 193 · Reveal 1

Speaker notes

After WEST, reveal both lower-ghost choices. In expectimax, they are equally likely.

Solution: Pacman Game Tree

SOLUTION



188

Speaker notes

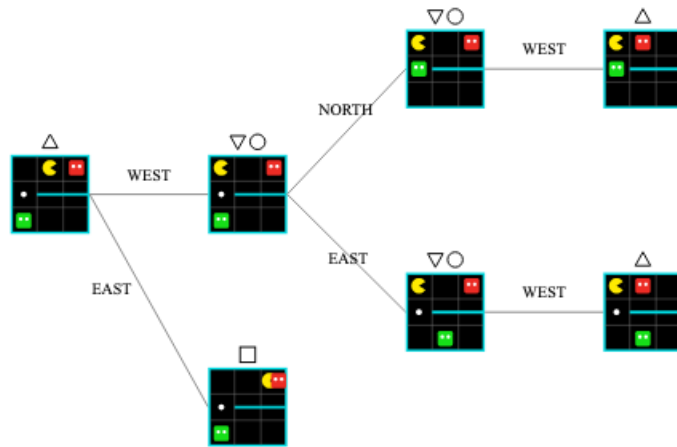
Reveal the upper ghost's forced WEST move in each branch.

Solution: Pacman Game Tree

Slide 188 of 193 · Reveal 2

Solution: Pacman Game Tree

SOLUTION



188

Solution: Pacman Game Tree

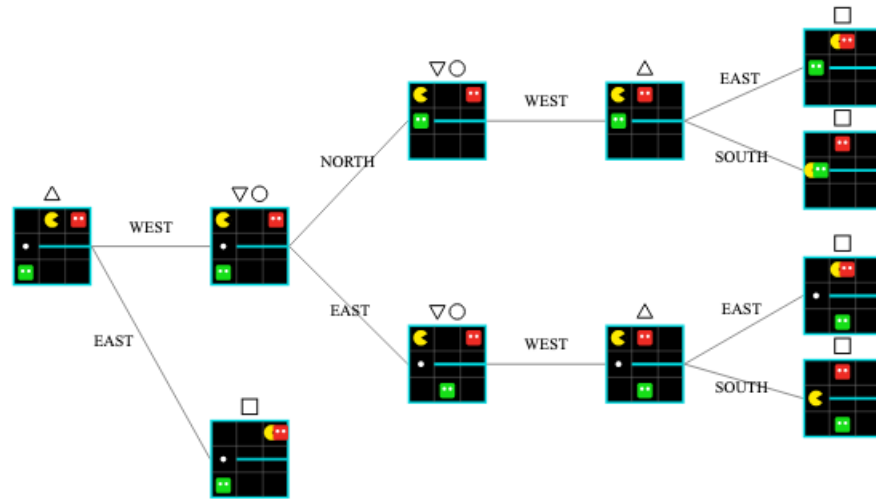
Slide 188 of 193 · Reveal 3

Speaker notes

Reveal both legal second Pacman moves after each ghost sequence. Each board is a state; five leaves end the game.

Solution: Pacman Game Tree

SOLUTION



188

Speaker notes

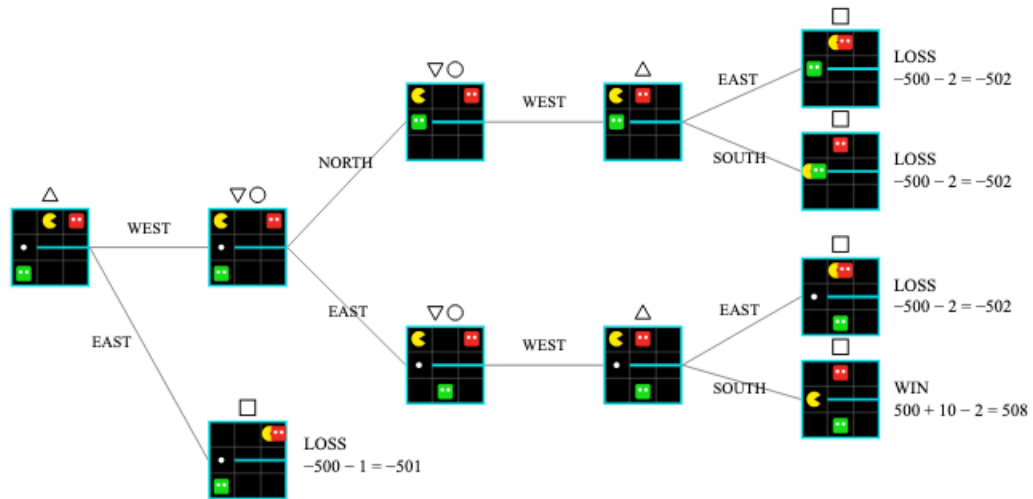
Reveal terminal arithmetic. Loss after two Pacman actions is -502 ; safe food after two actions is 508 ; immediate loss is -501 .

Solution: Pacman Game Tree

Slide 188 of 193 · Reveal 4

Solution: Pacman Game Tree

SOLUTION



188

Solution: Pacman Game Tree

Slide 188 of 193 · Reveal 5

Speaker notes

The next two slides back up the same terminal values under different ghost models.

SOLUTION



Slide 189 of 193 · Initial

The full tree stays in the same positions as on the task slide. First back up up the four final Pacman decisions: three losing leaves and one winning leaf.

SOLUTION

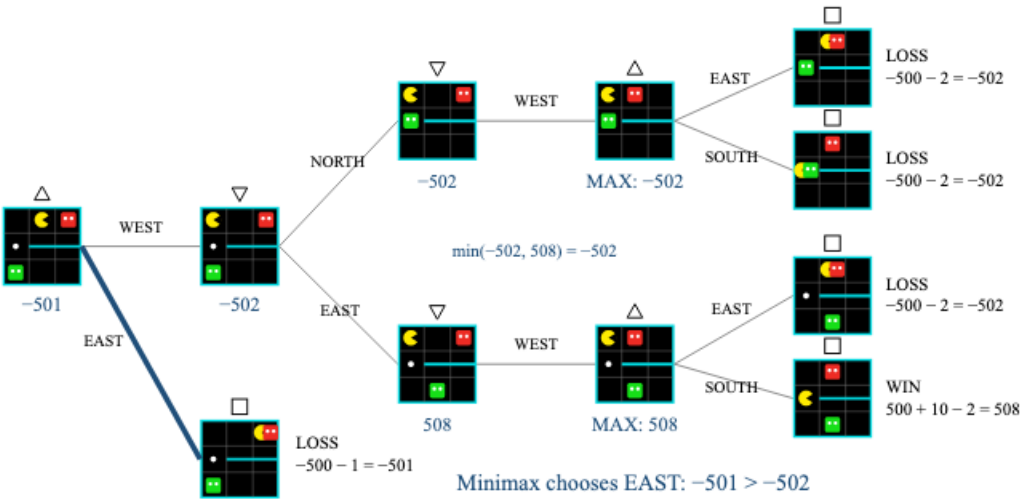


Slide 189 of 193 · Reveal 1

The upper ghost has only one action, so values pass unchanged. At the lower ghost, MIN selects the NORTH outcome: $\min(-502, 508) = -502$.

Solution: Minimax

SOLUTION



189

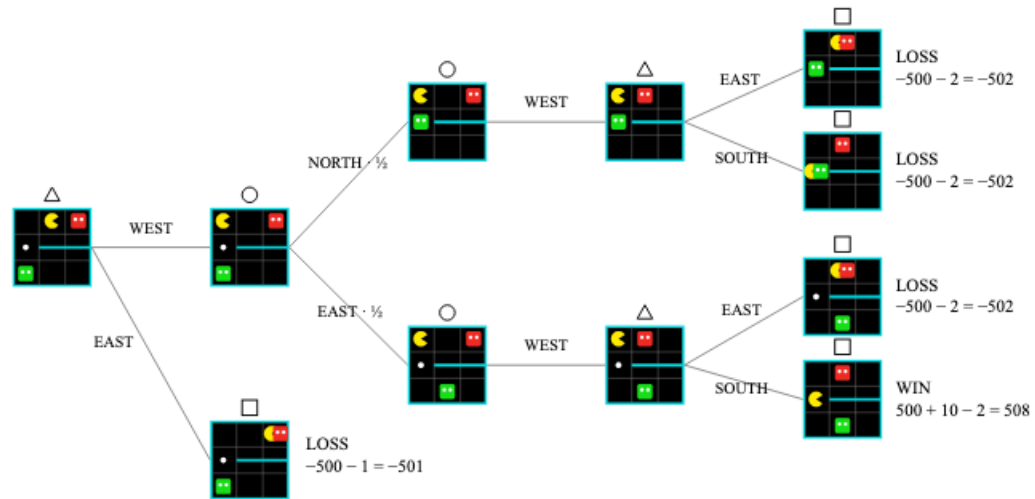
Speaker notes

No speaker notes for this stage.

Solution: Minimax

Slide 189 of 193 · Reveal 3

Solution: Expectimax

SOLUTION

190

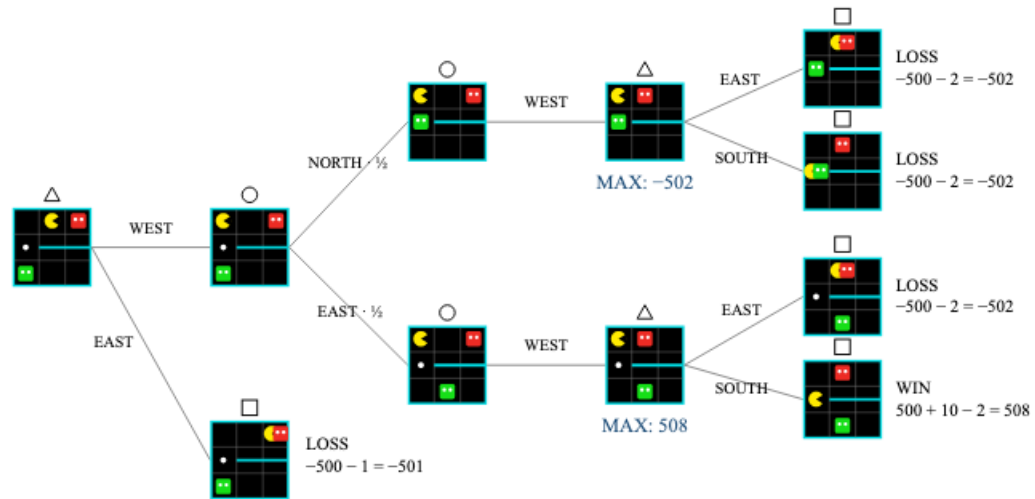
Speaker notes

The states and terminal scores have not changed. First back up Pacman's final choices in the two ghost branches.

Solution: Expectimax

Slide 190 of 193 · Initial

Solution: Expectimax

SOLUTION

190

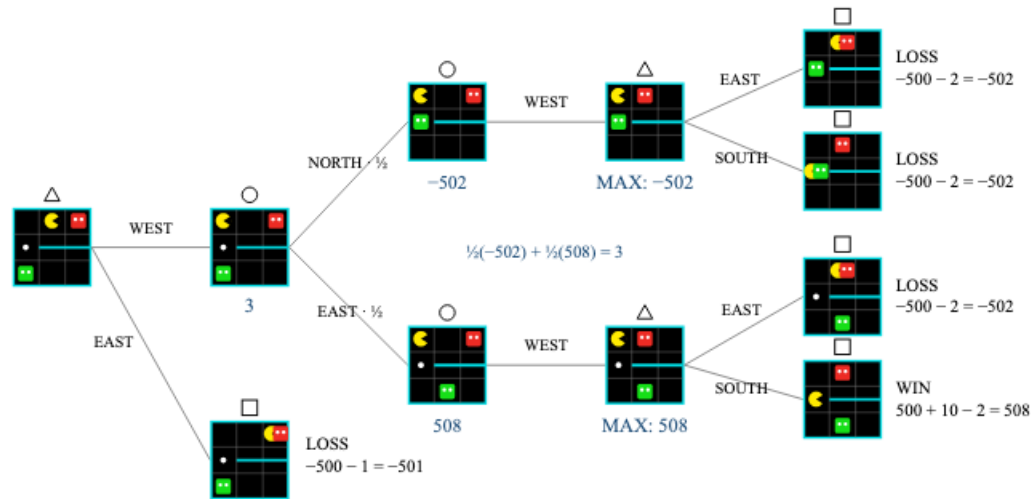
Solution: Expectimax

Slide 190 of 193 · Reveal 1

Speaker notes

The lower ghost is now a chance node: $0.5(-502) + 0.5(508) = 3$ for WEST.

Solution: Expectimax

SOLUTION

190

Speaker notes

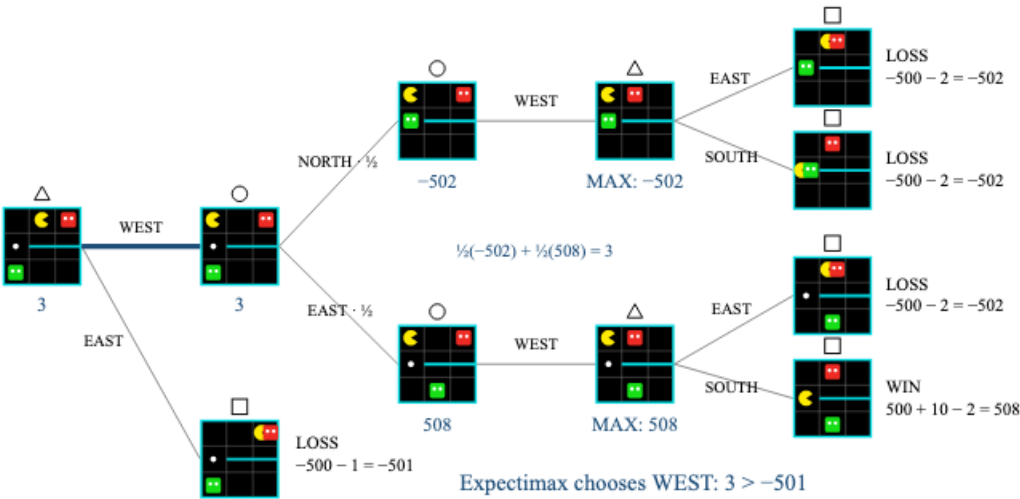
At the root, MAX compares WEST = 3 with EAST = -501 and chooses WEST. The outcome model, not the possible states, changed.

Solution: Expectimax

Slide 190 of 193 · Reveal 2

Solution: Expectimax

SOLUTION



190

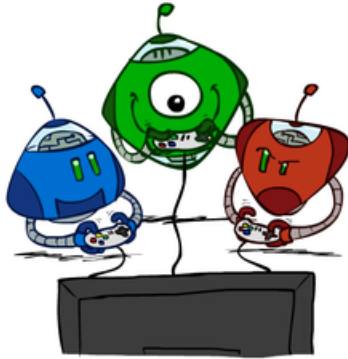
Speaker notes

No speaker notes for this stage.

Solution: Expectimax

Slide 190 of 193 · Reveal 3

More Than Two Agents



191

More Than Two Agents

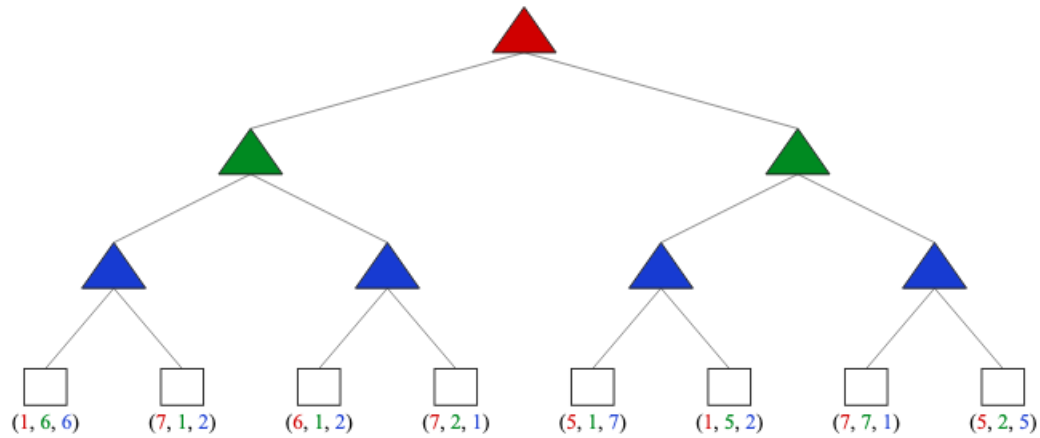
Slide 191 of 193 · Initial

Speaker notes

Two-player zero-sum games use a single scalar utility naturally. With more agents, each outcome can affect players differently.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

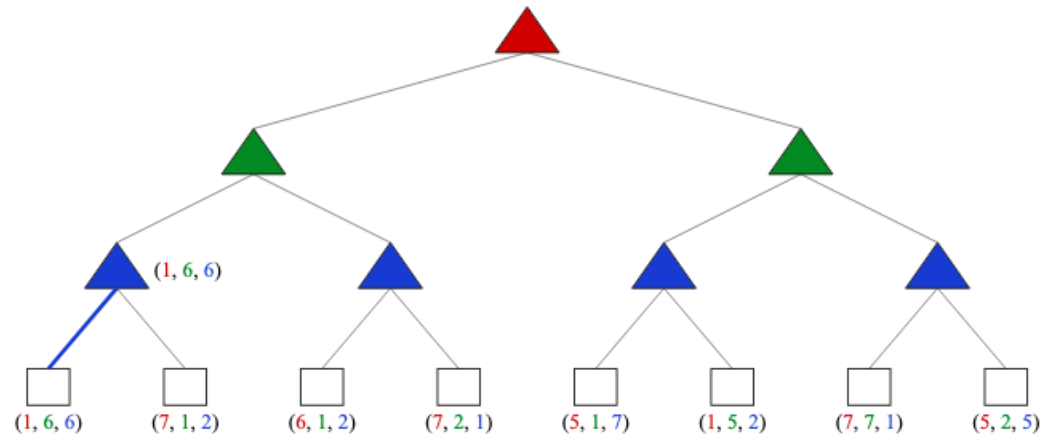
Slide 192 of 193 · Initial

Speaker notes

Each terminal has a utility vector. The triangle colour tells us which component that player maximises.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Speaker notes

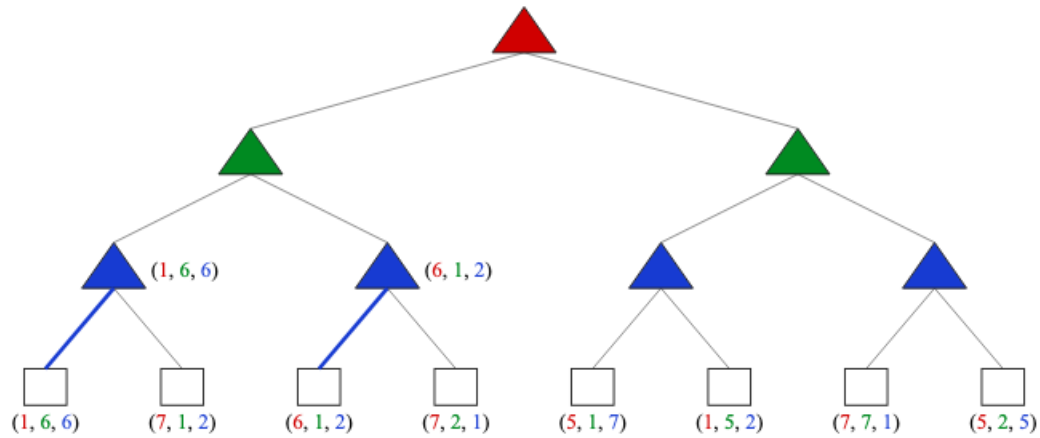
Blue chooses (1, 6, 6): the third value 6 beats 2.

Utility Vectors

Slide 192 of 193 · Reveal 1

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

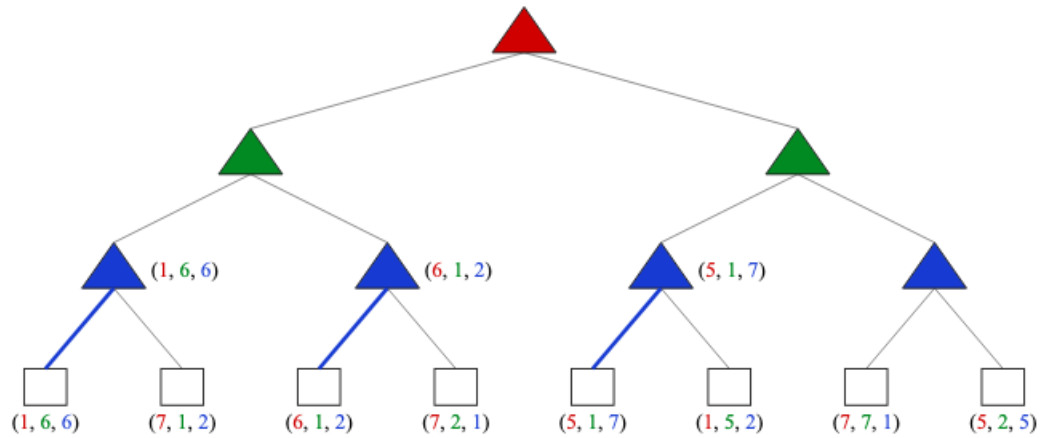
Slide 192 of 193 · Reveal 2

Speaker notes

Blue chooses (6, 1, 2): the third value 2 beats 1.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

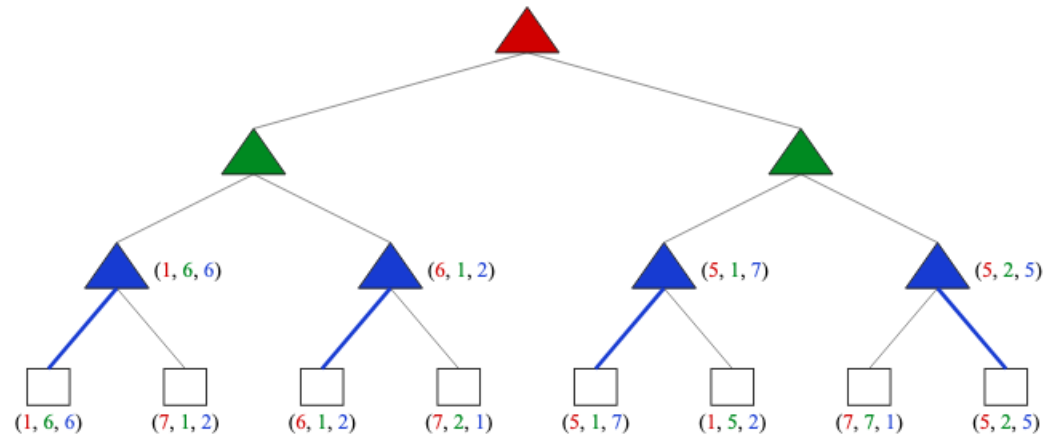
Slide 192 of 193 · Reveal 3

Speaker notes

Blue chooses $(5, 1, 7)$: the third value 7 beats 2.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

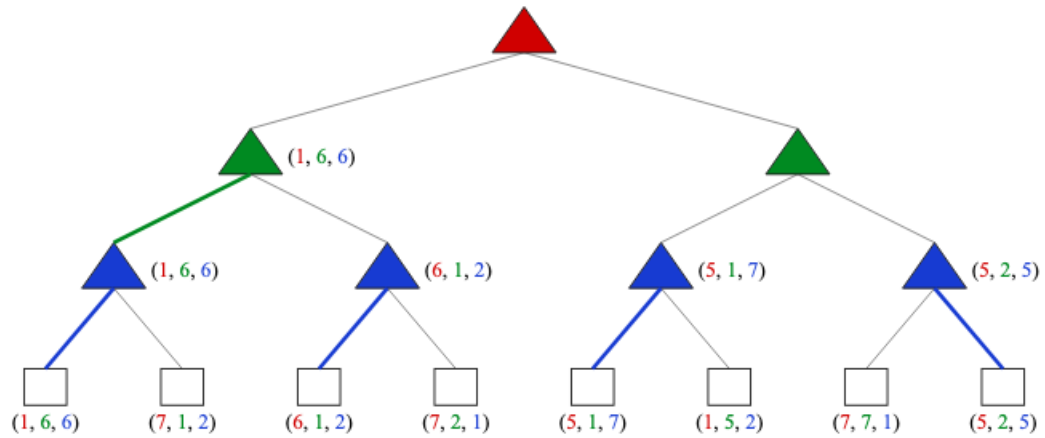
Slide 192 of 193 · Reveal 4

Speaker notes

Blue chooses $(5, 2, 5)$: the third value 5 beats 1.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

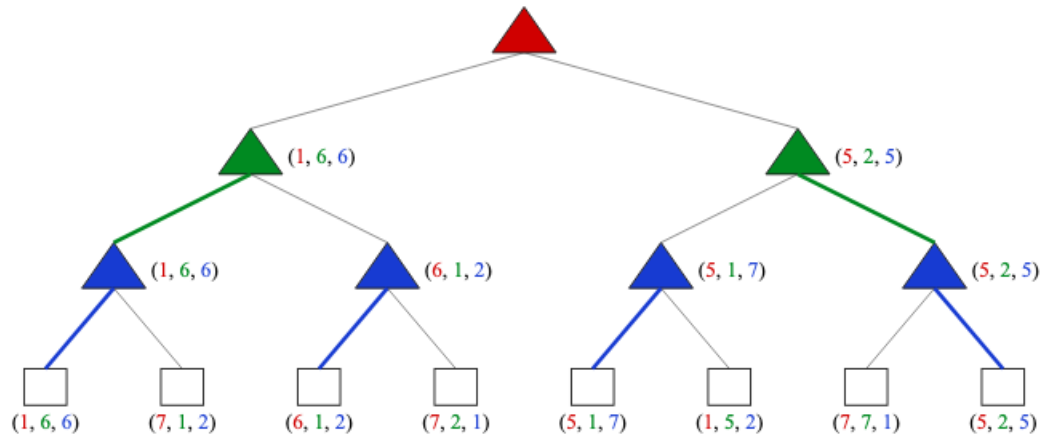
Slide 192 of 193 · Reveal 5

Speaker notes

Green chooses (1, 6, 6): the second value 6 beats 1.

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Speaker notes

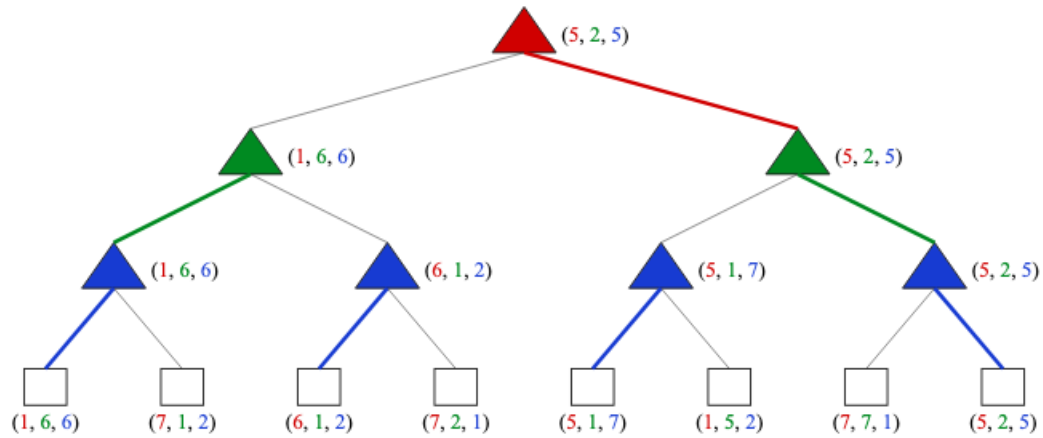
Green chooses $(5, 2, 5)$: the second value 2 beats 1.

Utility Vectors

Slide 192 of 193 · Reveal 6

Utility Vectors

Red chooses the first value · Green the second · Blue the third



192

Utility Vectors

Slide 192 of 193 · Reveal 7

Speaker notes

Red chooses the right branch, (5, 2, 5): the first value 5 beats 1.

Reading

AIMA Chapter 3

Search · 88 pages

https://comp3270.dirk.hk/AIMA_Chapter_3.pdf

AIMA Chapter 4

Search in Complex Environments · 62 pages

https://comp3270.dirk.hk/AIMA_Chapter_4.pdf

AIMA Chapter 5

Adversarial Search · 63 pages

https://comp3270.dirk.hk/AIMA_Chapter_5.pdf

AIMA Chapter 6

Constraint Satisfaction · 50 pages

https://comp3270.dirk.hk/AIMA_Chapter_6.pdf

Password: Dirk

193

Speaker notes

These are the companion AIMA readings for the search block. Chapter 5 covers minimax, alpha-beta pruning and stochastic games.

Reading

Slide 193 of 193 · Initial