KR-IST - Lecture 2b: Problem Solving

Chris Thornton

October 30, 2014

Introduction

The two basic search strategies are depth-first search (DFS) and breadth-first search (BFS).

DFS always expands a node at the deepest level of the tree.

BFS expands all nodes at one level before proceeding to the next.

DFS is cheap on memory

Because it always expands the deepest node, DFS is guaranteed to always explore any given path to its furthest extent.

This means it only has to store one path at any given time.

The space complexity is therefore proportional to the average length of paths.

BFS has to store all paths under consideration *simultaneously*.

DFS can use recursion

Depth-first search has the attraction that it is easy to implement simply by exploiting *recursion*.

A single method is used to carry out the search.

Initially, it is applied to the start node.

If the node turns out to be a goal node, a solution has been obtained.

Otherwise, the method generates all the node's successors and calls itself recursively on each one.

DFS is better when there are many solution nodes -

If the space contains many equally good solutions, DFS search is likely to succeed quite quickly.

If the space contains just one solution, quite close to the start node, DFS may perform a huge amount of unnecessary work, exploring ‘the depths’ of the tree.

Breadth-first search

The breadth-first strategy always expands all the nodes at one level of the tree, before expanding any of their children.

This strategy is guaranteed to find the shortest solution path first.

In most circumstances, the shortest path is also the *best* solution available.

Memory usage in BFS

BFS must represent all paths simultaneously.

The memory cost thus increases exponentially with the exploration depth and can be calculated in the usual way, using the branching factor raised to the relevant depth.

If there are few solutions, however, the strategy may be more effective than depth-first search.

A depth-first search may waste time exploring deep into the tree.

BFS is guaranteed not to do this.

Depth-limited search

Depth-limited search (DLS) is a compromise offering some of the benefits of breadth-first search without the memory costs.

The idea is to perform a depth-first search to a limited depth in the tree.

If we suspect that there is a solution at a depth of four, we might arrange for a depth-first search to give up searching any path containing more than four nodes.

This strategy offers the low memory costs of depth-first search but does no more work than a breadth-first strategy.

Unfortunately, we rarely know how deep the (best) solution is likely to be.

Iterative-deepening search

Iterative-deepening search (IDS) provides a way around the difficulty of identifying the right limit for depth-limited searching.

With iterative-deepening search, we start off by applying a depth-limited search using a minimal depth limit (e.g., one level).

If this doesn't succeed we increase the depth limit by one and repeat the search, continuing on until we find a solution.

IDS is the best all-rounder

IDS is arguably the best, general-purpose search strategy since it offers the low memory costs of depth-first search together with the optimality and completeness of breadth-first search.

Intuition suggest that IDS will do a lot of unnecessay work since it will *repeatedly* explore the upper levels of the search tree.

However, in general, ‘most’ of the nodes in a search space are in the lower levels.

By avoiding unnecessary exploration of these levels, the iterative-deepening strategy manages to achieve a respectable time complexity.

Completeness and optimality

A search strategy is said to be

Completeness and optimality

A search strategy is said to be

- *complete* if it is guaranteed to find a solution when there is one, and

Completeness and optimality

A search strategy is said to be

- *complete* if it is guaranteed to find a solution when there is one, and
- *optimal* if it is guaranteed to find the highest-quality (e.g., shortest) solution.

Completeness and optimality

A search strategy is said to be

- *complete* if it is guaranteed to find a solution when there is one, and
- *optimal* if it is guaranteed to find the highest-quality (e.g., shortest) solution.

Search as problem solving

We've seen how search can be applied to route-finding problems.

In fact, it can be applied to *any* problem involving sequencing of transitions between 'situations'.

The situations don't have to be physical locations.

They can be intermediate states of affairs which are achieved by relevant actions.

This application of search is known as **problem solving**.

Nodes are **states**

The start node is the **initial state** or **root state**.

8-puzzle example

The 8-puzzle is a problem readily solved by search.

A small plastic tray is divided into a 3x3 grid.

This is covered with eight tiles numbered 1 to 8. One of the positions on the grid is empty and into this space we can slide a tile from left, right, above or below, depending on where the space is.

To solve the puzzle we have to slide tiles around so as to get them into numeric order reading left-to-right and top-to-bottom, i.e, we have to produce a tile pattern which looks something like this.

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

The 8-puzzle as a search problem

To solve the 8-puzzle, we need to find a sequence of transitions (i.e., tile movements) which achieves a particular goal state (all tiles in numeric order) starting from some given starting state (the initial tile configuration).

Ideally, the search should identify the shortest solution path.

Implementing the search

To solve any problem using search—whether we do it by hand or using a computer program—we have to decide two things.

Implementing the search

To solve any problem using search—whether we do it by hand or using a computer program—we have to decide two things.

(1) How are we going to represent states?

Implementing the search

To solve any problem using search—whether we do it by hand or using a computer program—we have to decide two things.

(1) How are we going to represent states?
(2) How are we going to generate successors?

Implementing the search

To solve any problem using search—whether we do it by hand or using a computer program—we have to decide two things.

(1) How are we going to represent states?
(2) How are we going to generate successors?

All other aspects of the process stay the same in all problems.

State representation

If we are working by hand, a convenient representation for states is just a 3x3 grid of numbers, e.g.,

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

If we are programming, a convenient representation is likely to be a 1-dimensional or 2-dimensional array of integers.

Successor generation

Any given state in this problem potentially has *four* successor states:

Successor generation

Any given state in this problem potentially has *four* successor states:

- one resulting from moving a tile *down* into the hole,

Successor generation

Any given state in this problem potentially has *four* successor states:

- one resulting from moving a tile *down* into the hole,
- one resulting from moving a tile *up* into the hole,

Successor generation

Any given state in this problem potentially has *four* successor states:

- one resulting from moving a tile *down* into the hole,
- one resulting from moving a tile *up* into the hole,
- one resulting from moving a tile *left* into the hole and

Successor generation

Any given state in this problem potentially has *four* successor states:

- one resulting from moving a tile *down* into the hole,
- one resulting from moving a tile *up* into the hole,
- one resulting from moving a tile *left* into the hole and
- one resulting from moving a tile *right* into the hole.

Successor generation

Any given state in this problem potentially has *four* successor states:

- one resulting from moving a tile *down* into the hole,
- one resulting from moving a tile *up* into the hole,
- one resulting from moving a tile *left* into the hole and
- one resulting from moving a tile *right* into the hole.

Successors in practice

In fact, there will only be all four successors if the hole is right in the middle.

If it is on an edge, there will be only three.

If it is in a corner there will be only two.

To generate successors we take the parent node and, for each possible move, generate a copy of the parent with the relevant number moved to a new cell.

Any method which generates all the successors of a node is a **successor function**.

In effect, the successor function is a *virtual* representation of the entire search tree, since it enables any part of the tree to be brought into existence 'on demand'.

Program run

Initial state

```
1 2 5
3 4 8
6 0 7
```

Search tree using a linear representation of states

Tree generated by DFS applied to eight puzzle.

```
[1, 2, 5, 3, 4, 8, 6, 0, 7]
|-- [1, 2, 5, 3, 4, 8, 0, 6, 7]
|   | ...
|-- [1, 2, 5, 3, 4, 8, 6, 7, 0]
|   |-- [1, 2, 5, 3, 4, 0, 6, 7, 8]
|       |-- [1, 2, 0, 3, 4, 5, 6, 7, 8]
|       |   |-- [1, 0, 2, 3, 4, 5, 6, 7, 8]
|       |       |-- [1, 4, 2, 3, 0, 5, 6, 7, 8]
|       |       |-- [0, 1, 2, 3, 4, 5, 6, 7, 8]
|       |-- [1, 2, 5, 3, 0, 4, 6, 7, 8]
|           | ...
|-- [1, 2, 5, 3, 0, 8, 6, 4, 7]
    | ...
```

Solution path obtained (5 steps)

```
[1, 2, 5, 3, 4, 8, 6, 0, 7]
[1, 2, 5, 3, 4, 8, 6, 7, 0]
[1, 2, 5, 3, 4, 0, 6, 7, 8]
[1, 2, 0, 3, 4, 5, 6, 7, 8]
[1, 0, 2, 3, 4, 5, 6, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

.

Summary

Summary

- DFS is cheap on memory

Summary

- DFS is cheap on memory
- DFS can use recursion

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder
- Completeness and optimality

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder
- Completeness and optimality
- Search as problem solving

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder
- Completeness and optimality
- Search as problem solving
- Implementing an 8-puzzle search

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder
- Completeness and optimality
- Search as problem solving
- Implementing an 8-puzzle search
- State representation and successor generation

Summary

- DFS is cheap on memory
- DFS can use recursion
- DFS is better when there are many solution nodes
- Depth-limited search v. Iterative-deepening search
- IDS is the best all-rounder
- Completeness and optimality
- Search as problem solving
- Implementing an 8-puzzle search
- State representation and successor generation

Questions

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?
- How do we use an estimate of branching factor in working out the space complexity of BFS?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?
- How do we use an estimate of branching factor in working out the space complexity of BFS?
- Iterative deepening search offers many of the advantages of BFS. Why does it not incur the same memory costs?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?
- How do we use an estimate of branching factor in working out the space complexity of BFS?
- Iterative deepening search offers many of the advantages of BFS. Why does it not incur the same memory costs?
- As a rule, the depth limit in iterative deepening search is increased by 1 with each iteration of the search? In what circumstances might it make sense to increase the limit by a greater amount?

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?
- How do we use an estimate of branching factor in working out the space complexity of BFS?
- Iterative deepening search offers many of the advantages of BFS. Why does it not incur the same memory costs?
- As a rule, the depth limit in iterative deepening search is increased by 1 with each iteration of the search? In what circumstances might it make sense to increase the limit by a greater amount?
- What is the justification for saying that 'most' of the nodes in a search tree are in the bottom level.

Questions

- If DFS only stores the path it is currently exploring at any given time, how does it know where the other paths are?
- As a rule, DFS will do a lot of unnecessary searching where there is just one solution node close to the start node. What is the exception to this?
- Why is BFS guaranteed to find the shortest solution path?
- How do we use an estimate of branching factor in working out the space complexity of BFS?
- Iterative deepening search offers many of the advantages of BFS. Why does it not incur the same memory costs?
- As a rule, the depth limit in iterative deepening search is increased by 1 with each iteration of the search? In what circumstances might it make sense to increase the limit by a greater amount?
- What is the justification for saying that 'most' of the nodes in a search tree are in the bottom level.

More questions

More questions

- Imagine a breadth-first search which applies a depth-limit, i.e., it stops searching at a certain depth. Is the strategy formally *complete*? Is it *optimal*?

More questions

- Imagine a breadth-first search which applies a depth-limit, i.e., it stops searching at a certain depth. Is the strategy formally *complete*? Is it *optimal*?
- Describe a problem that could not be solved using search.

More questions

- Imagine a breadth-first search which applies a depth-limit, i.e., it stops searching at a certain depth. Is the strategy formally *complete*? Is it *optimal*?
- Describe a problem that could not be solved using search.
- What is the difference between a state and a node in a search tree?

More questions

- Imagine a breadth-first search which applies a depth-limit, i.e., it stops searching at a certain depth. Is the strategy formally *complete*? Is it *optimal*?
- Describe a problem that could not be solved using search.
- What is the difference between a state and a node in a search tree?
- How can a search process detect that it has reached a terminal node?

More questions

- Imagine a breadth-first search which applies a depth-limit, i.e., it stops searching at a certain depth. Is the strategy formally *complete*? Is it *optimal*?
- Describe a problem that could not be solved using search.
- What is the difference between a state and a node in a search tree?
- How can a search process detect that it has reached a terminal node?

Exercises

You have 99 pounds of debt accumulated on your credit card. The annual rate of interest on this card is 16% for debt under 110 pounds of debt and 25% otherwise. The problem is to decide whether it is worth transferring the debt to another card. Card A offers 8% for debt under 125 pounds and 35% for anything over that. Card B offers 18% for debt under 150 pounds and 23% for anything over that. You can only transfer your debt once per year and the rules are you have to transfer all of it in one go at a cost of 10 pounds. You never make any payments but (for some reason) no penalties or other charges are applied for this.

Exercises

You have 99 pounds of debt accumulated on your credit card. The annual rate of interest on this card is 16% for debt under 110 pounds of debt and 25% otherwise. The problem is to decide whether it is worth transferring the debt to another card. Card A offers 8% for debt under 125 pounds and 35% for anything over that. Card B offers 18% for debt under 150 pounds and 23% for anything over that. You can only transfer your debt once per year and the rules are you have to transfer all of it in one go at a cost of 10 pounds. You never make any payments but (for some reason) no penalties or other charges are applied for this.

- Specify a suitable representation for states in this problem.

Exercises

You have 99 pounds of debt accumulated on your credit card. The annual rate of interest on this card is 16% for debt under 110 pounds of debt and 25% otherwise. The problem is to decide whether it is worth transferring the debt to another card. Card A offers 8% for debt under 125 pounds and 35% for anything over that. Card B offers 18% for debt under 150 pounds and 23% for anything over that. You can only transfer your debt once per year and the rules are you have to transfer all of it in one go at a cost of 10 pounds. You never make any payments but (for some reason) no penalties or other charges are applied for this.

- Specify a suitable representation for states in this problem.
- Specify a suitable method for generating successors in this problem. (Note that a transition is the action you take at the end of each year either to transfer the debt or stick with your current card.)

Exercises

You have 99 pounds of debt accumulated on your credit card. The annual rate of interest on this card is 16% for debt under 110 pounds of debt and 25% otherwise. The problem is to decide whether it is worth transferring the debt to another card. Card A offers 8% for debt under 125 pounds and 35% for anything over that. Card B offers 18% for debt under 150 pounds and 23% for anything over that. You can only transfer your debt once per year and the rules are you have to transfer all of it in one go at a cost of 10 pounds. You never make any payments but (for some reason) no penalties or other charges are applied for this.

- Specify a suitable representation for states in this problem.
- Specify a suitable method for generating successors in this problem. (Note that a transition is the action you take at the end of each year either to transfer the debt or stick with your current card.)

Exercises cont.

Exercises cont.

- Draw out the full search tree for possible actions over a four year period.

Exercises cont.

- Draw out the full search tree for possible actions over a four year period.
- Assuming you stick with your current card in the first year, what sequence of subsequent actions will minimise the debt build-up over the five years.

Exercises cont.

- Draw out the full search tree for possible actions over a four year period.
- Assuming you stick with your current card in the first year, what sequence of subsequent actions will minimise the debt build-up over the five years.
- What will the accumulated debt be after five years if you switch to card B after the first year?

Exercises cont.

- Draw out the full search tree for possible actions over a four year period.
- Assuming you stick with your current card in the first year, what sequence of subsequent actions will minimise the debt build-up over the five years.
- What will the accumulated debt be after five years if you switch to card B after the first year?
- What would the accumulated debt be if no transfers were made?

Exercises cont.

- Draw out the full search tree for possible actions over a four year period.
- Assuming you stick with your current card in the first year, what sequence of subsequent actions will minimise the debt build-up over the five years.
- What will the accumulated debt be after five years if you switch to card B after the first year?
- What would the accumulated debt be if no transfers were made?

Exercises cont.

Exercises cont.

- In a depth-first search on this problem, we could always choose to expand the node associated with the lowest level of accumulated debt. Would this strategy be guaranteed to identify the least debt-accumulation?

Exercises cont.

- In a depth-first search on this problem, we could always choose to expand the node associated with the lowest level of accumulated debt. Would this strategy be guaranteed to identify the least debt-accumulation?
- Consider the case where there are just two cards A and B. A offers 10% for debt up to 100 pounds and 20% thereafter. B offers 8% up to 120 pounds and 20% thereafter. You start with 100 pounds of debt on card A and the rules for transfers remain the same, i.e., a maximum of one per year at a cost of 10 pounds. Draw the search tree down to four levels (years) and identify the sequence of actions which achieves the lowest accumulated debt.

Exercises cont.

- In a depth-first search on this problem, we could always choose to expand the node associated with the lowest level of accumulated debt. Would this strategy be guaranteed to identify the least debt-accumulation?
- Consider the case where there are just two cards A and B. A offers 10% for debt up to 100 pounds and 20% thereafter. B offers 8% up to 120 pounds and 20% thereafter. You start with 100 pounds of debt on card A and the rules for transfers remain the same, i.e., a maximum of one per year at a cost of 10 pounds. Draw the search tree down to four levels (years) and identify the sequence of actions which achieves the lowest accumulated debt.

Resources

Resources

- Russell and Norvig, p. 73 covers of the four main criteria of evaluation relevant to search.

Resources

- Russell and Norvig, p. 73 covers of the four main criteria of evaluation relevant to search.