KR-IST - Lecture 4a
Heuristic search with A*

Chris Thornton

November 16, 2011

The problem of the supersized search space

Search is a flexible tool which can be used, in principle, to obtain a solution to any problem.

In practice, there is a serious difficulty.

For *most* problems, the search tree is just too big to explore in a reasonable amount of time.

Even for a problem as simple as the 8-puzzle, there are more than 31 thousand, million states to be checked.

Checking states at the rate of one per millisecond, this would take nearly a year.

The need for knowledge

If the search process is left to blindly explore the entire search space there is the risk that it will take too long.

It is generally necessary to provide *knowlege* which will enable the search to move more directly towards a solution node.

Search processes with knowledge of this type are said to be **informed**.

Processes which carry out the search in a blind or exhaustive fashion are said to be **uninformed**.

Evaluation functions

Knowledge is provided to the search in the form of an **evaluation function** for search nodes.

This function returns a value which estimates the cost of a given node, i.e., how far it is from a goal node.

The search process can use the function to select the best node to expand at any point (i.e., to choose 'which way to go').

Evaluation functions are heuristic functions

Evaluation functions are often called **heuristic functions** on the grounds that they utilise rules-of-thumb.

Search using evaluation functions is therefore **heurstic search**.

But this is more than just terminology.

It would be inconsistent to use a *completely* accurate evaluation function for purposes of guiding a search.

Or at least it would be strange to describe the resulting process as 'search'.

With a completely accurate evaluation function, the right branch can be selected at every stage. Search is not required.

Best-first search

Let's say we have an evaluation function which estimates the cost of reaching a goal from any given node, i.e., its 'distance'.

Best-first search

Let's say we have an evaluation function which estimates the cost of reaching a goal from any given node, i.e., its 'distance'.

- f(n) = distance to the nearest goal from n

Best-first search

Let's say we have an evaluation function which estimates the cost of reaching a goal from any given node, i.e., its 'distance'.

- f(n) = distance to the nearest goal from n

We can use a cost function of this type to decide which of a set of nodes should be expanded next. We just expand the node with the lowest f value.

This procedure is known as **best-first** or **ordered** search.

Implementational issues

With best-first search, successors are not checked in a fixed sequence.

So it is necessary to maintain some sort of data structure to show which nodes remain unchecked.

This is normally done using two list structures:

Implementational issues

With best-first search, successors are not checked in a fixed sequence.

So it is necessary to maintain some sort of data structure to show which nodes remain unchecked.

This is normally done using two list structures:

- A list called OPEN containing nodes which have been generated but not expanded.

Implementational issues

With best-first search, successors are not checked in a fixed sequence.

So it is necessary to maintain some sort of data structure to show which nodes remain unchecked.

This is normally done using two list structures:

- A list called OPEN containing nodes which have been generated but not expanded.
- A list called CLOSED containing nodes which have already been expanded.

Implementational issues

With best-first search, successors are not checked in a fixed sequence.

So it is necessary to maintain some sort of data structure to show which nodes remain unchecked.

This is normally done using two list structures:

- A list called OPEN containing nodes which have been generated but not expanded.
- A list called CLOSED containing nodes which have already been expanded.

Processing loop

In each iteration, the algorithm selects the most promising node from OPEN, e.g., the node with lowest estimated cost.

If the node is a goal node, a solution has been obtained.

If the node is not a goal, it is then moved from OPEN to CLOSED.

Its successors are then examined and any that don't currently appear in OPEN or CLOSED are added to OPEN.

Best-first search algorithm

(1) Put the start node n on a list called OPEN, of unexpanded nodes and associate the f(n) value with it.

(2) If OPEN is empty, exit with failure; no solution exists.

(3) Select from OPEN a node n for which f(n) is a minimum. If several nodes qualify, choose a solution node if there is one, and otherwise choose among them arbitrarily.

(4) Remove node n from OPEN and place it on CLOSED.

(5) If n is a goal node, exit with success; a solution has been found.

(6) Expand node n, creating nodes for all its successors. For every successor n, if n is neither in OPEN nor in CLOSED, then add it to OPEN, with its f(n) value. Attach a pointer from n back to the predecessor node (to provide access to the path to the goal node.)

(7) Go to step (2).

A* search

A* search is essentially best-first search upgraded for use with path-oriented evaluation functions.

This is the kind of evaluation function that we will need to use whenever the solution is the *path* to the goal node rather than the node itself.

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

- g(n) = the cost of reaching node n from the start

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

- g(n) = the cost of reaching node n from the start

and

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

- g(n) = the cost of reaching node n from the start

and

- h(n) = the cost of reaching a goal from n

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

- g(n) = the cost of reaching node n from the start

and

- h(n) = the cost of reaching a goal from n

The final evaluation of cost is then the sum of g(n) and h(n):

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, $g(n)$ and $h(n)$, where

- $g(n) =$ the cost of reaching node n from the start

and

- $h(n) =$ the cost of reaching a goal from n

The final evaluation of cost is then the sum of $g(n)$ and $h(n)$:

- $f(n) = g(n) + h(n) =$ overall cost of node n

Evaluation with A*

In A* search, evaluation of the cost of a given node is assumed to be defined in terms of two components, g(n) and h(n), where

- g(n) = the cost of reaching node n from the start

and

- h(n) = the cost of reaching a goal from n

The final evaluation of cost is then the sum of g(n) and h(n):

- f(n) = g(n) + h(n) = overall cost of node n

(In some presentations these functions are written as f*, g* and h* with the unadorned letters being used to denote the *actual* costs.)

Node updates are required in A* search

The catch with A* is that, due to taking the known part of the solution into account when working out costs, it is actually possible for f(n) values to change.

This happens if the search uncovers a new, lower-cost path to a previously expanded state.

If this situation is detected, the algorithm must update the evaluations associated with the state, both in OPEN and CLOSED, and change the predecessor pointer so as to connect it to the new path.

If there is an improvement in the evaluation of a state in CLOSED, that state must be transferred back to OPEN.

(See step step 6.c in the Handbook of AI, vol 1, p. 61).

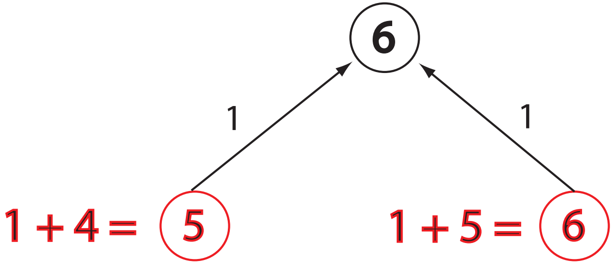
A heuristic which guarantees that f(n) values will never drop in this way is said to be *consistent*.

Evaluation change for open node, state 1

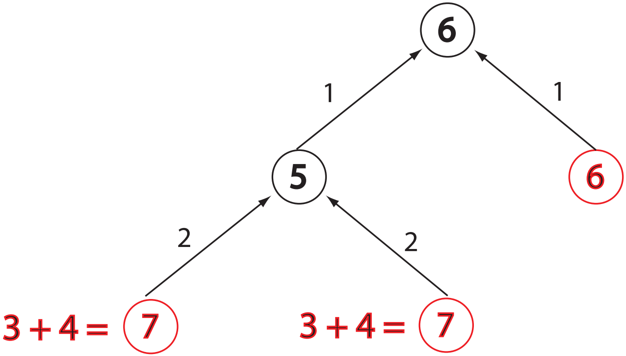
g	h	f
0 +	6 =	6

$$0 + 6 = 6$$

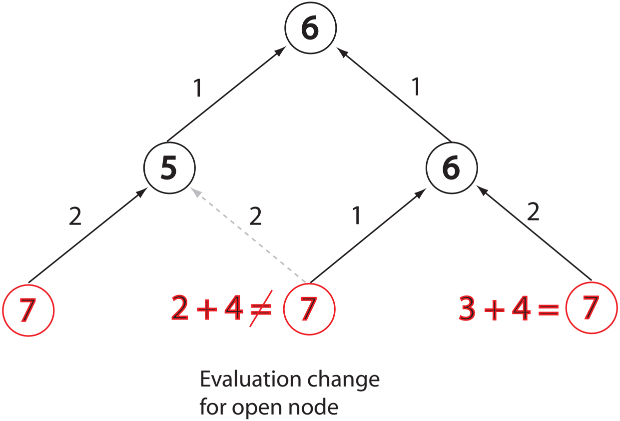
Evaluation change for open node, state 2



Evaluation change for open node, state 3



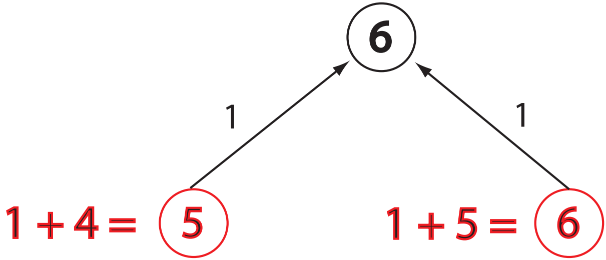
Evaluation change for open node, state 4



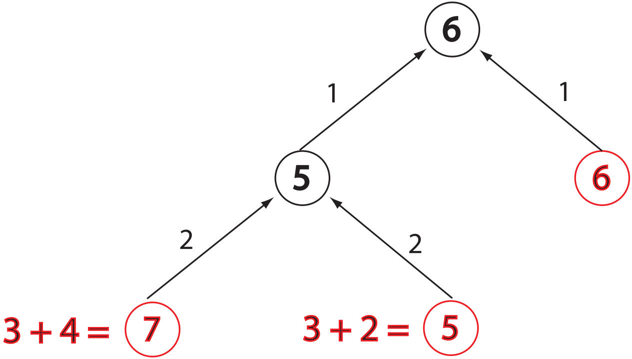
Evaluation change for closed node, state 1

g		h		f
0	+	6	=	6

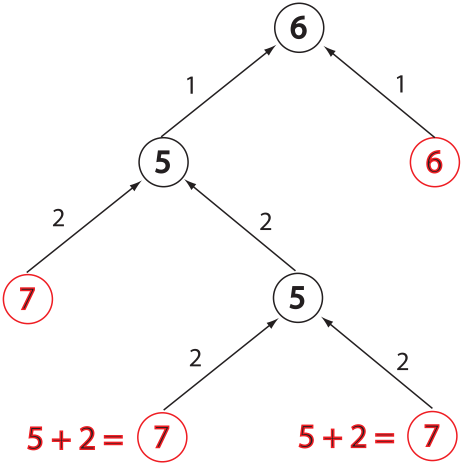
Evaluation change for closed node, state 2



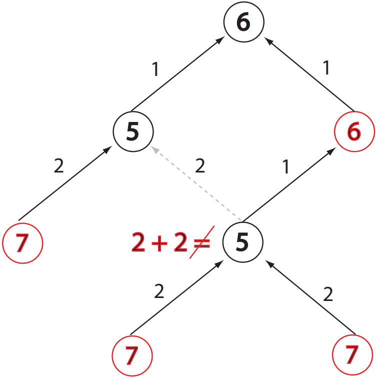
Evaluation change for closed node, state 3



Evaluation change for closed node, state 4



Evaluation change for closed node, state 5



Optimality

How reliable is A* search?

The g(n) value is always right, since it measures the cost of a path which has already been identified. (This is normally the path length.)

The h value is an *estimate*.

But provided it never *overestimates* the cost, the search is guaranteed to be optimal.

The requirement that h not overestimate the cost is known as the *admissibility criterion*.

Why admissibility guarantees optimality

The cost of node at the end of a solution path is guaranteed to be correct. At this point, $h(n) = 0$, so $f(n) = g(n)$.

If h never overestimates, the cost of a node at the end of a *sub-optimal* solution path must be greater than the cost of any node on an optimal solution path.

Assuming A* always expands the node with the lowest cost, search will continue until the optimal solution path is identified.

Unfortunately, in practice, the admissibility criterion may not be satisfied.

Heuristic function for the 8-puzzle

Russell and Norvig investigate the performance of two possible heuristic functions for the 8-puzzle problem (see pp. 101-103).

Heuristic function for the 8-puzzle

Russell and Norvig investigate the performance of two possible heuristic functions for the 8-puzzle problem (see pp. 101-103).

- h1 = the number of tiles that are in the wrong position.

Heuristic function for the 8-puzzle

Russell and Norvig investigate the performance of two possible heuristic functions for the 8-puzzle problem (see pp. 101-103).

- h1 = the number of tiles that are in the wrong position.
- h2 = the sum of the ‘city-block’ distances of the tiles from their goal positions.

Heuristic function for the 8-puzzle

Russell and Norvig investigate the performance of two possible heuristic functions for the 8-puzzle problem (see pp. 101-103).

- h1 = the number of tiles that are in the wrong position.
- h2 = the sum of the 'city-block' distances of the tiles from their goal positions.

They use empirical performance evaluations and the concept of **branching factor**, to demonstrate that that h2 is considerably more effective than h1.

They also show that A* search (using either heuristic) is orders of magnitude faster than ordinary iterative-deepening search, the best of the 'uninformed' bunch.

This result, showing the supremacy of A* search over uninformed iterative-deepening search, is common across most search problems.

.

Summary

Summary

- The problem of the supersized search space

Summary

- The problem of the supersized search space
- The need for knowledge

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search
- Evaluation with A*

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search
- Evaluation with A*
- Node updates required in A* search

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search
- Evaluation with A*
- Node updates required in A* search
- Optimality of A*

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search
- Evaluation with A*
- Node updates required in A* search
- Optimality of A*
- Admissibility criterion

Summary

- The problem of the supersized search space
- The need for knowledge
- Evaluation functions
- Best-first search
- A* search
- Evaluation with A*
- Node updates required in A* search
- Optimality of A*
- Admissibility criterion

Questions

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?
- How can the total number of states in the 8-puzzle be calculated?

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?
- How can the total number of states in the 8-puzzle be calculated?
- How would you describe a search process whose state evaluation function was 100% accurate.

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?
- How can the total number of states in the 8-puzzle be calculated?
- How would you describe a search process whose state evaluation function was 100% accurate.
- At what point of processing in best-first search is a search node transferred from OPEN list to CLOSED.

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?
- How can the total number of states in the 8-puzzle be calculated?
- How would you describe a search process whose state evaluation function was 100% accurate.
- At what point of processing in best-first search is a search node transferred from OPEN list to CLOSED.
- Why should best-first search avoid adding to OPEN nodes which are already on that list?

Questions

- Where heuristic search is used to obtain a solution path, we need a representation of nodes which allows extra bits of information to be associated with the state itself. What are those bits of information. What sort of data-structure might be appropriate for node representation?
- How can the total number of states in the 8-puzzle be calculated?
- How would you describe a search process whose state evaluation function was 100% accurate.
- At what point of processing in best-first search is a search node transferred from OPEN list to CLOSED.
- Why should best-first search avoid adding to OPEN nodes which are already on that list?

More questions

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?
- How many components of cost are used in the calculation of an A* evaluation.

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?
- How many components of cost are used in the calculation of an A* evaluation.
- What impact does the use of a path-oriented evaluation function have on the implementation of best-first search?

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?
- How many components of cost are used in the calculation of an A* evaluation.
- What impact does the use of a path-oriented evaluation function have on the implementation of best-first search?
- On what grounds could one say that A* is not an optimal search method?

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?
- How many components of cost are used in the calculation of an A* evaluation.
- What impact does the use of a path-oriented evaluation function have on the implementation of best-first search?
- On what grounds could one say that A* is not an optimal search method?
- What is the so-called *admissability criterion* for evaluation functions in A*.

More questions

- Why should A* bother to update evaluations of nodes on CLOSED?
- What would motivate the use of A* search rather than the simpler, best-first search?
- How many components of cost are used in the calculation of an A* evaluation.
- What impact does the use of a path-oriented evaluation function have on the implementation of best-first search?
- On what grounds could one say that A* is not an optimal search method?
- What is the so-called *admissability criterion* for evaluation functions in A*.

More questions

More questions

- Why does admissability ensure optimality?

More questions

- Why does admissability ensure optimality?
- Explain why R&N's h2 heuristic function is so much more effective for the 8-puzzle problem than their h1 function.

More questions

- Why does admissability ensure optimality?
- Explain why R&N's h2 heuristic function is so much more effective for the 8-puzzle problem than their h1 function.
- How might one go about inventing a good heuristic function for a novel search problem?

More questions

- Why does admissability ensure optimality?
- Explain why R&N's h2 heuristic function is so much more effective for the 8-puzzle problem than their h1 function.
- How might one go about inventing a good heuristic function for a novel search problem?
- Why does R&N's goal state for the 8-puzzle have the hole in the top-left corner?

More questions

- Why does admissability ensure optimality?
- Explain why R&N's h2 heuristic function is so much more effective for the 8-puzzle problem than their h1 function.
- How might one go about inventing a good heuristic function for a novel search problem?
- Why does R&N's goal state for the 8-puzzle have the hole in the top-left corner?

Exercises

Exercises

- Assuming the goal state for the 8-puzzle is

Exercises

- Assuming the goal state for the 8-puzzle is

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

draw out the search tree down to four levels of search (i.e., four levels plus the start node) using this as the start state. The tree should only contain one instance of any given state.

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

Exercises

- Assuming the goal state for the 8-puzzle is

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

draw out the search tree down to four levels of search (i.e., four levels plus the start node) using this as the start state. The tree should only contain one instance of any given state.

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

- How many distinct states does this depth-limited search take into account?

Exercises

- Assuming the goal state for the 8-puzzle is

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

draw out the search tree down to four levels of search (i.e., four levels plus the start node) using this as the start state. The tree should only contain one instance of any given state.

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

- How many distinct states does this depth-limited search take into account?
- Estimate the branching factor for this space and compare the predicted space complexity (at four levels) with the actual space complexity.

Exercises

- Assuming the goal state for the 8-puzzle is

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

draw out the search tree down to four levels of search (i.e., four levels plus the start node) using this as the start state. The tree should only contain one instance of any given state.

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

- How many distinct states does this depth-limited search take into account?
- Estimate the branching factor for this space and compare the predicted space complexity (at four levels) with the actual space complexity.

Exercises cont.

Exercises cont.

- Identify the best path, i.e., the path whose final state is closest to the goal state.

Exercises cont.

- Identify the best path, i.e., the path whose final state is closest to the goal state.
- Re-draw the search space using

Exercises cont.

- Identify the best path, i.e., the path whose final state is closest to the goal state.
- Re-draw the search space using

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

as the starting state. In what way is this problem qualitatively different to the original?

Exercises cont.

Exercises cont.

- Look at Russell and Norvig's analysis of heuristic functions for the 8-puzzle (section 4.2) and check the accuracy of their illustration of the way costs for heuristic h2 are calculated (i.e., $h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$).

Exercises cont.

- Look at Russell and Norvig's analysis of heuristic functions for the 8-puzzle (section 4.2) and check the accuracy of their illustration of the way costs for heuristic h2 are calculated (i.e., h2 $= 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$).
- The A* algorithm is complicated by the need to update predecessor links (e.g., the need to occasionally move nodes back from CLOSED to OPEN). Provide an example to illustrate why the algorithm needs to do this.

Exercises cont.

- Look at Russell and Norvig's analysis of heuristic functions for the 8-puzzle (section 4.2) and check the accuracy of their illustration of the way costs for heuristic h2 are calculated (i.e., $h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$).
- The A* algorithm is complicated by the need to update predecessor links (e.g., the need to occasionally move nodes back from CLOSED to OPEN). Provide an example to illustrate why the algorithm needs to do this.
- How could the A* algorithm be adapted so as to produce a tree representation (e.g., structured list) of the space searched.

Exercises cont.

- Look at Russell and Norvig's analysis of heuristic functions for the 8-puzzle (section 4.2) and check the accuracy of their illustration of the way costs for heuristic h2 are calculated (i.e., h2 $= 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$).
- The A* algorithm is complicated by the need to update predecessor links (e.g., the need to occasionally move nodes back from CLOSED to OPEN). Provide an example to illustrate why the algorithm needs to do this.
- How could the A* algorithm be adapted so as to produce a tree representation (e.g., structured list) of the space searched.

Resources

Resources

- Russell and Norvig sections 4.1 and 4.2.

Resources

- Russell and Norvig sections 4.1 and 4.2.
- Handbook of AI, Volume 1, C3a and C3b.

Resources

- Russell and Norvig sections 4.1 and 4.2.
- Handbook of AI, Volume 1, C3a and C3b.