



CE213 Artificial Intelligence – Lecture 4

Heuristic Search Strategies

None of the blind search strategies does something that any human solver would do:

i.e., consider how close states appear to be to the goal when deciding what to try next.

To do this we will have to provide two things:

A way of assessing how close a state is to the goal

A way of using this additional information in the search

Heuristic and Evaluation Function

Suppose that in addition to the state space formalisation of a problem, the solver also has a function that can estimate how close a state is to a goal state.

Such function is called ***evaluation function or heuristic***:

$$h : \mathcal{S} \rightarrow \mathbb{R} \quad (\mathbb{R} : \text{the set of real numbers})$$

We use heuristics all the time in everyday life.

e.g., Suppose you are looking for W.H. Smiths in a strange town.

A useful heuristic is that it is probably pretty close to Marks and Spencers.

An evaluation function doesn't have to be perfect to be useful.

Heuristic Search Strategies

Heuristic search strategies are those that use a heuristic evaluation function in determining **which node to expand next**.
(a key question to be answered by any search strategy)

Much of AI is concerned with heuristics

They make problems with exponential time complexity tractable, by reducing search space or avoiding searching part of state space where there is surely no goal state.

— Benefit from using heuristics

Route Finding Problems

SatNav

- Use GPS to determine where you are now.
- Use heuristic search to find best route to where you want to be.

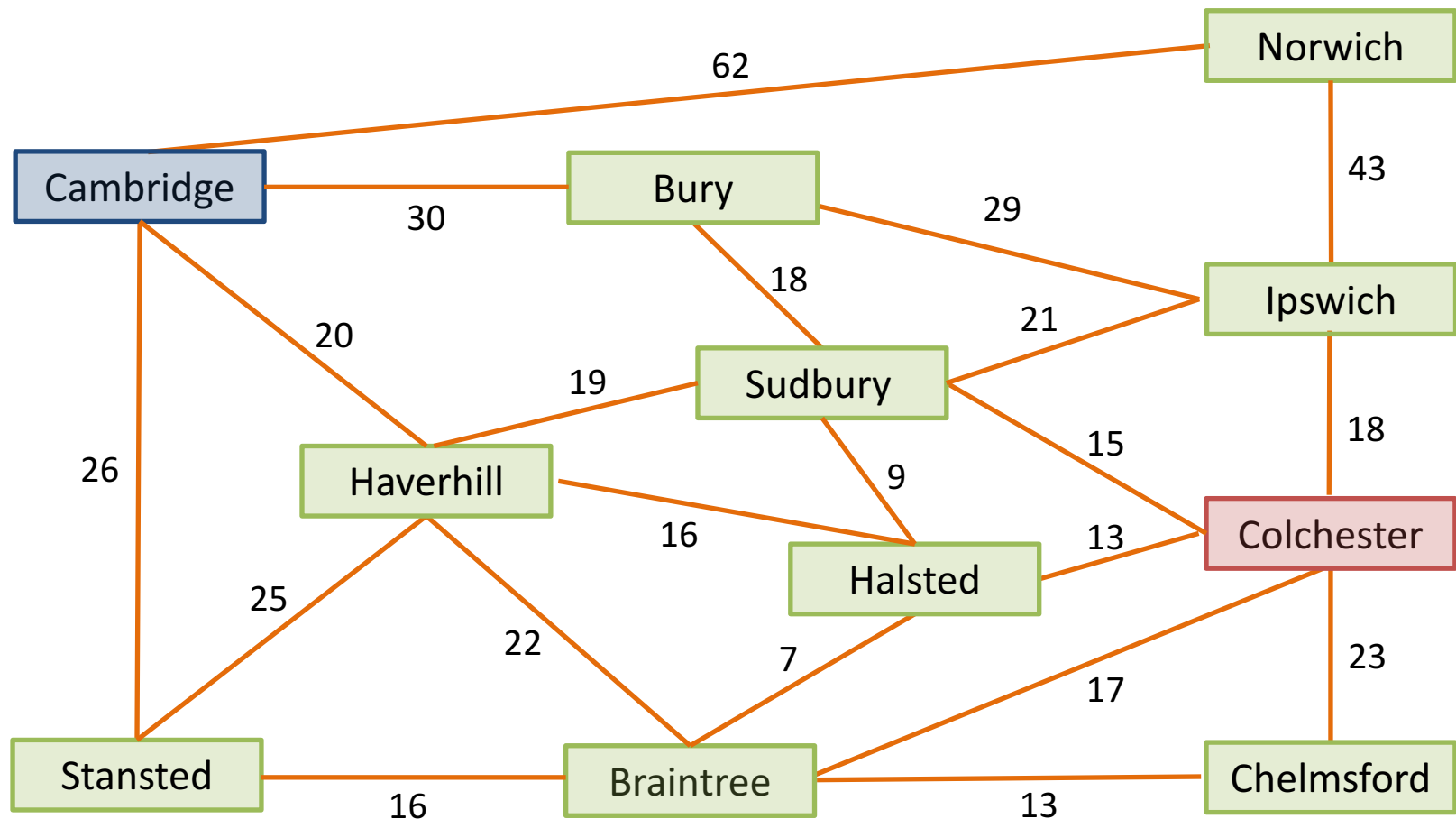
Given a starting point and a destination

Return the 'best' route

An Example:

Suppose we want to drive from Colchester to Cambridge

Major Roads in East Anglia (as a **state space**)



Heuristic Distance Estimates

Straight line distances of towns from Cambridge

Town	Direct Distance
Braintree	33
Bury	26
Cambridge	0
Chelmsford	36
Colchester	40
Halsted	29
Haverhill	16
Ipswich	45
Norwich	59
Stansted	24
Sudbury	28

$$h : \mathcal{S} \rightarrow \mathcal{R}$$

(a lookup table defining an evaluation function)

Greedy Search

Greedy search is the simplest and most obvious form of heuristic search:

It always chooses the node that is closest to a goal state in terms of the heuristic, regardless of the distance (cost) between the initial state (root) and the current state (node).

Greedy Search (2)

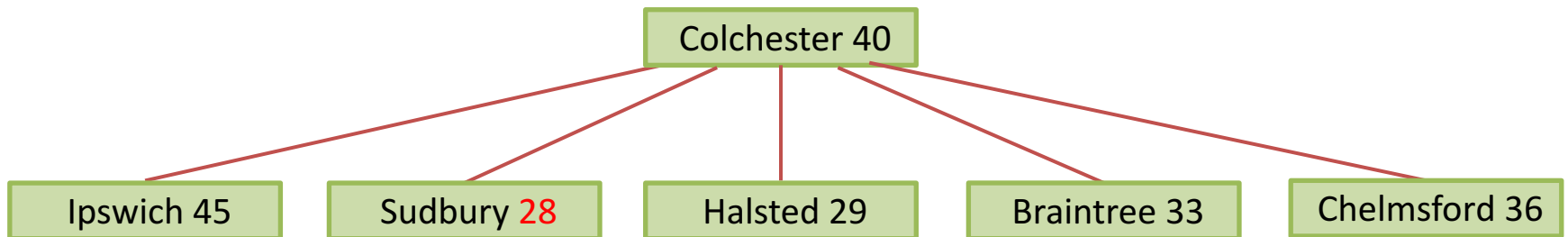
Applying this to the route finding problem:

Clearly the root node is Colchester:

Colchester 40

For convenience, the distance heuristic is indicated in the node.

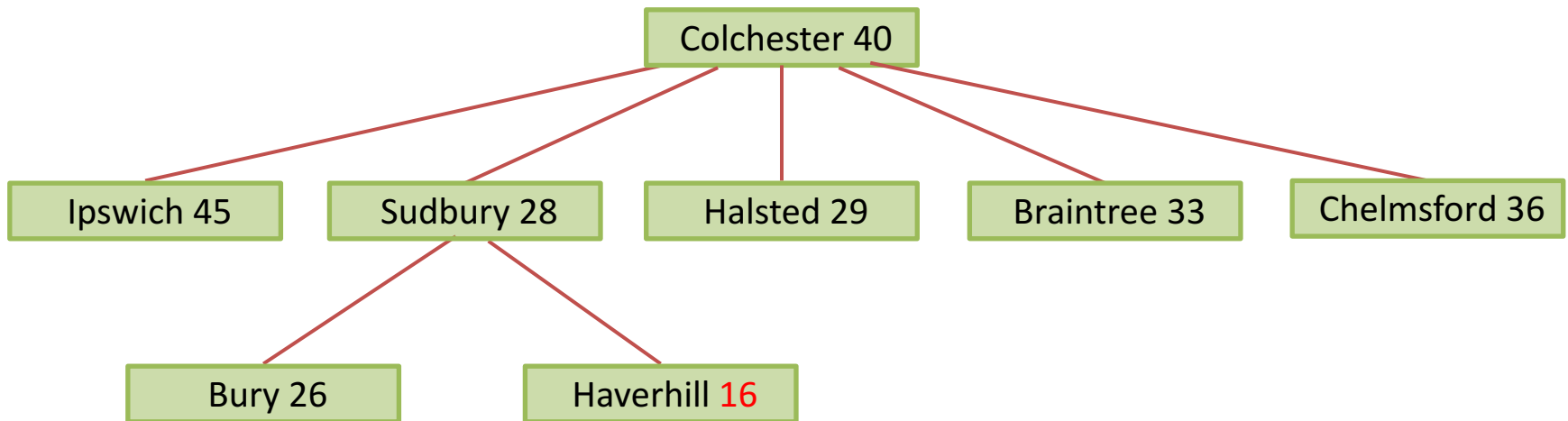
Expanding this gives:



The heuristic indicates that Sudbury is closest to the goal among the 5 unexpanded nodes, so this node is expanded next:

Greedy Search (3)

Expanding Sudbury:

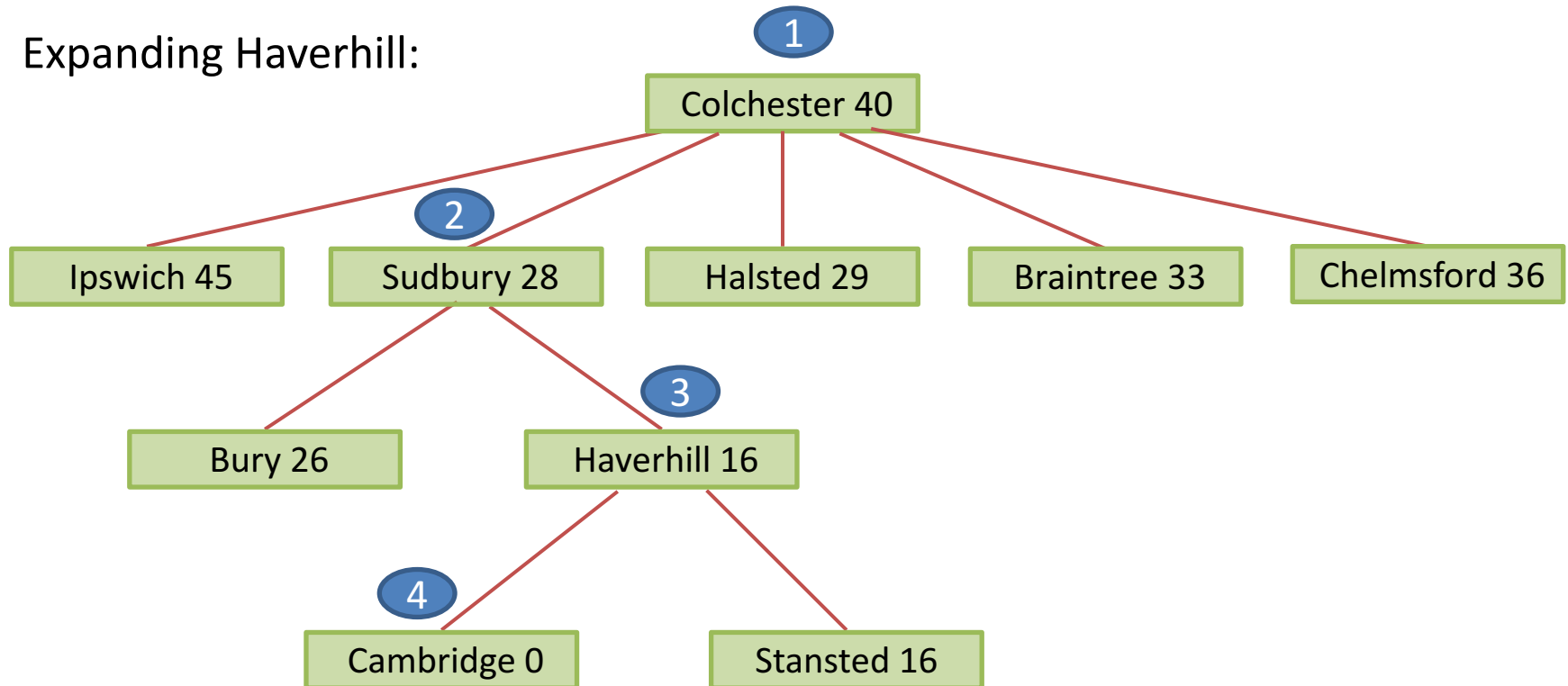


New nodes for towns that have already been reached have not been added.

At this point, the heuristic indicates that Haverhill is closest to the goal among the **6** unexpanded nodes (They are at different depth), so this node is chosen for expansion:

Greedy Search (4)

Expanding Haverhill:



And the problem is solved: Colchester-Sudbury-Haverhill-Cambridge
(4 nodes have been selected for expansion.)

Greedy Search (5)

Two features of greedy search should be noted:

- The solver moves fairly directly toward a solution
Nodes such as Norwich are never reached.
Nodes such as Ipswich and Chelmsford are never expanded.
So the search time is kept low.

- The solution is quite good, but **not optimal**.

Route found:

Colchester-Sudbury-Haverhill-Cambridge

Length 54 miles

Shortest route is only 49 miles (Colchester-Halsted-Haverhill-Cambridge)

Pseudo Code for Greedy Search

```
Create an empty list UnexpNodes;  
Add initial node to UnexpNodes;  
WHILE (UnexpNodes not empty)  
    N = The item n in UnexpNodes, whose heuristic  
        value  $h(n)$  is smallest  
    Remove N from UnexpNodes  
    IF N is goal  
    THEN Produce Solution and Return(Success) ;  
    Expand N to produce list of successors of N;  
    Add successors to tail of UnexpNodes; 1)  
Return(Failure)
```

Note 1: If a second route is found to a node that has already been reached before, it is not necessary to add it to the unexpanded nodes list.

A* Search

Why does the greedy search fail to find an optimal solution?

Because it takes no account of the **cost of reaching a node from the root**, and makes decisions purely on the basis of the estimated **cost to go from the current node to the goal**.

But we already have uniform cost search that takes account of cost of reaching a node from the root.

Why not combine greedy search and uniform cost search?

If we do this, we have ***A* Search***.

A* Search (2)

Suppose:

$g(n)$ is actual **cost of getting to node n** from initial state (root).

$h(n)$ is heuristic estimate of **cost of getting to goal from node n** .

Then we have three alternative search strategies:

Uniform Cost:

Expand the node with smallest $g(n)$

Greedy:

Expand the node with smallest $h(n)$

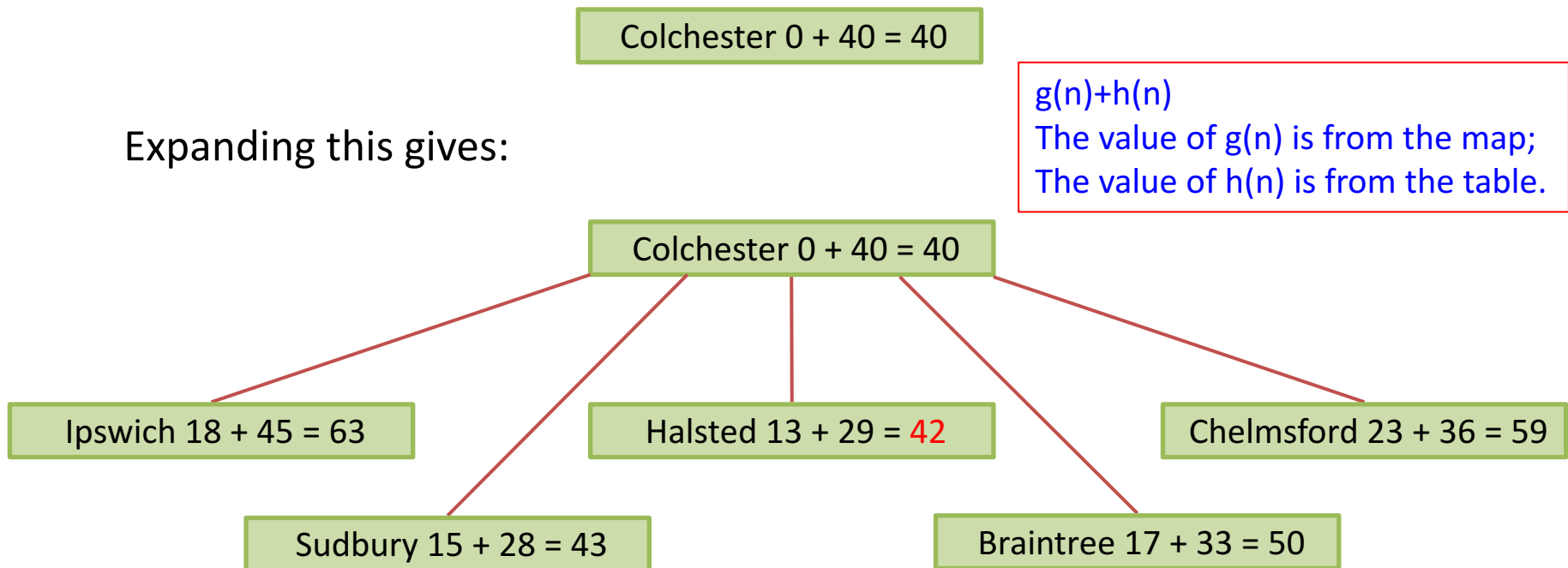
A*:

Expand the node with smallest $g(n) + h(n)$

A* Search (3)

Applying A* Search to the route finding problem.

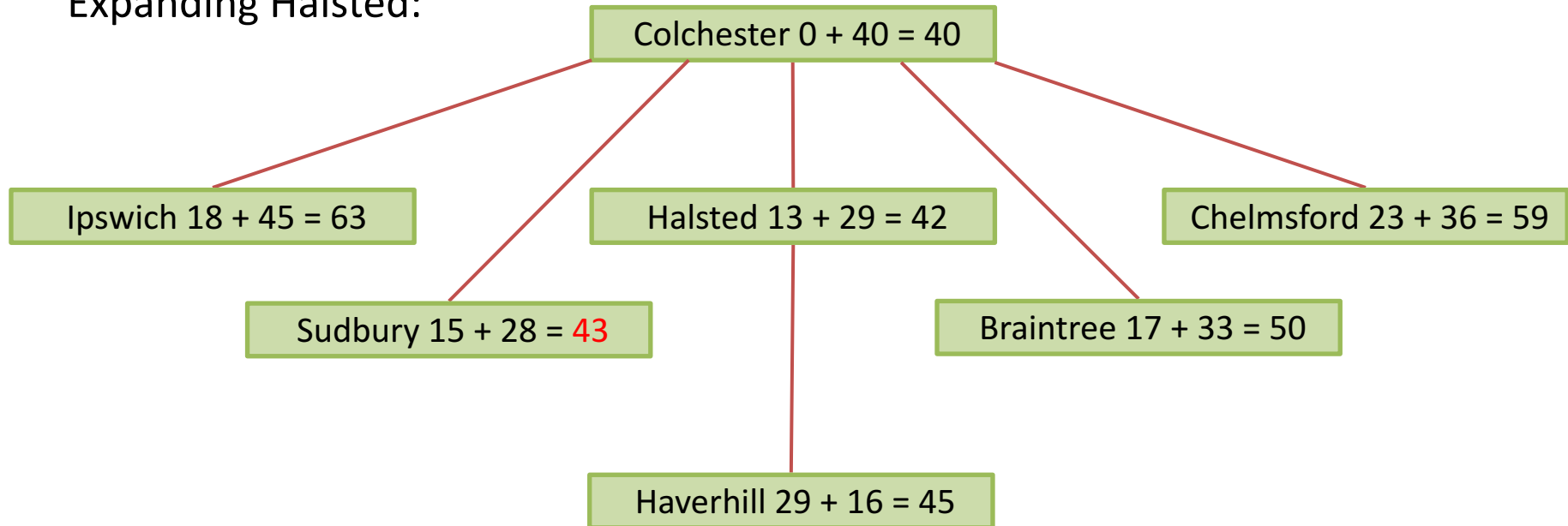
As before, Colchester is the root node:



The lowest value of $g(n) + h(n)$ is at Halsted so this node is expanded next:

A* Search (4)

Expanding Halsted:

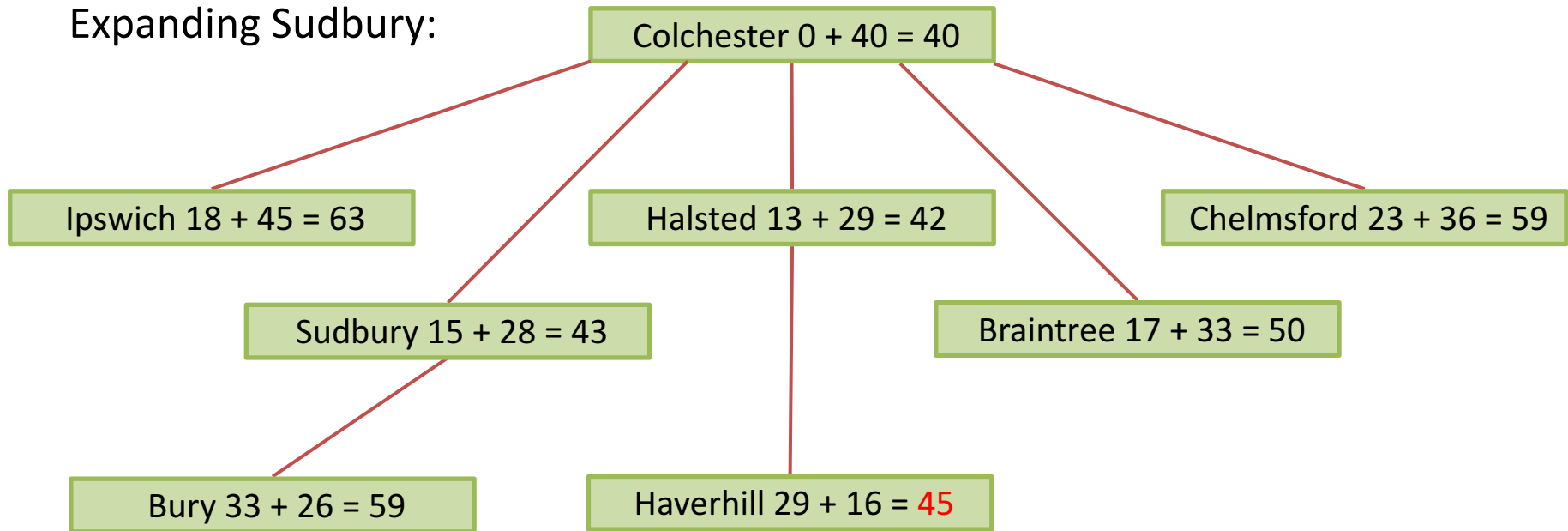


This time we have not included the new nodes for places already reached (e.g. Sudbury and Braintree) only because the new paths to them have higher cost than those already found.

The lowest value of $g(n) + h(n)$ (among the **5** unexpanded nodes) is now at Sudbury, so this node is expanded next:

A* Search (5)

Expanding Sudbury:

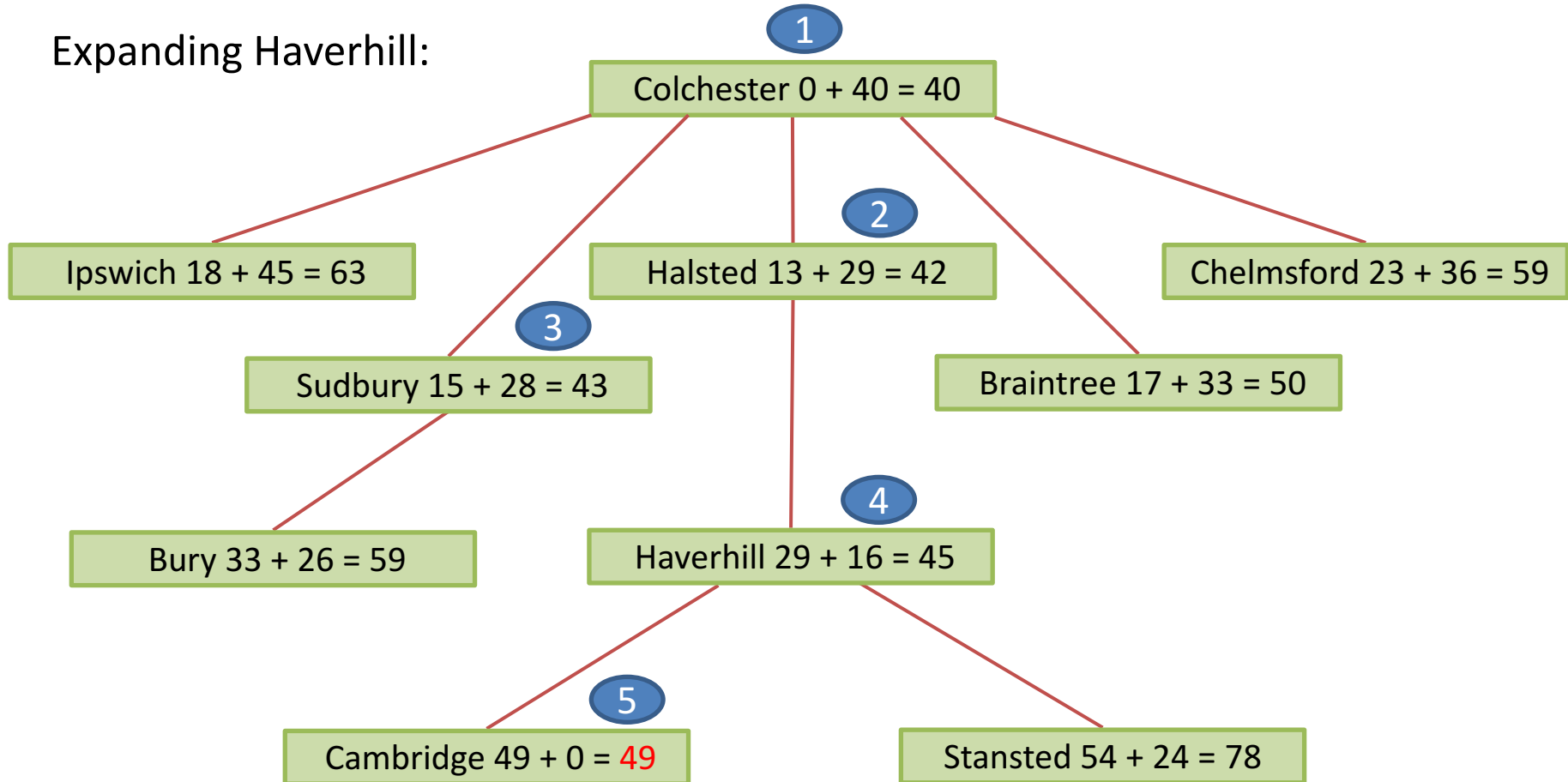


Note that we do not include the new path to Haverhill because it is longer (34) than the one already discovered (29).

Now the lowest value of $g(n) + h(n)$ is at Haverhill, so this node is expanded next: (5 nodes available for expansion: some are at depth 1 and some at depth 2)

A* Search (6)

Expanding Haverhill:



Finally the node for Cambridge itself is selected for expansion (lowest value among **6** unexpanded nodes). Thus, A* search stops.

(5 nodes have been selected for expansion.)

A* Search (7)

We have found a solution.

Colchester-Halsted-Haverhill-Cambridge

Length 49 miles

And this time it is an optimal solution.

It is important that the search waits until the goal node is selected for expansion before declaring a solution has been found.

Otherwise the solution may not be optimal. We will give examples to show this in the class.

Pseudo Code for A* Search

```
Create an empty list UnexpNodes;  
Add initial node to UnexpNodes;  
WHILE (UnexpNodes not empty)  
    N = The item n in UnexpNodes with  $g(n)+h(n)$  being smallest  
    Remove N from UnexpNodes  
    IF N is goal  
    THEN Produce Solution and Return(Success) ;  
    Expand N to produce list of successors of N;  
    IF cheaper routes to any node in UnexpNodes have been found  
    THEN replace their costlier parent nodes with the cheaper  
    ones;  
    Add successors to tail of UnexpNodes;  
Return(Failure) ;
```

$h(n)$ is a function returning a heuristic estimate of the cost of reaching the goal from node n .

$g(n)$ is a function returning the actual cost of reaching node n from the initial state (root).

Properties of A* Search

The behaviour of A* Search depends on the heuristic used. In practice, admissible heuristics are often used.

An ***admissible*** heuristic is one that **never overestimates** the true cost of reaching the goal (e.g., direct distance is admissible).

It can be proved that, ***provided an admissible heuristic is used***, A* search is Complete and Optimal

It is desirable that A* Search adopts admissible heuristics. However, this may not be guaranteed, and thus A* search could be non-optimal in some applications.

Properties of A* Search (2)

Time and Space Complexity

The **time** taken by A* search depends on how accurate the heuristic is.

In the best case, with a perfect heuristic, the search goes straight to the solution.

With an uninformative heuristic (e.g., $h = 0$ for all nodes) the method reduces to uniform cost search.


With a misleading heuristic it can be worse.

The **space** required grows exponentially with solution path length unless the heuristic is very accurate. This is the main limitation on the use of A* search.

The main advantage of A* search lies in its possible low time complexity (depending on accurate heuristic).

Some Comparative Results

Breadth First: Colchester, Braintree, Haverhill, Cambridge.

 **59 miles** 11 nodes expanded

Depth First: Colchester, Ipswich, Norwich, Cambridge.

123 miles 5 nodes expanded

Uniform Cost: Colchester, Halsted, Haverhill, Cambridge.

49 miles 10 nodes expanded 

Greedy: Colchester, Sudbury, Haverhill, Cambridge.

54 miles **4 nodes expanded**

A*: Colchester, Halsted, Haverhill, Cambridge.

49 miles **5 nodes expanded**

N.B. Results for depth first are extremely sensitive to order in which roads are considered. Uniform cost search took much more time to find the same route as A*.

Hill Climbing (for self study)

All the search strategies we have examined before have assumed that we could explore several possibilities before choosing what to do.

- One option didn't seem to be leading us to a solution, so we went back and tried an alternative. (e.g., depth first)
- We always considered alternatives before pursuing one of them further. (e.g., breadth first)

This is often unrealistic.

- You often don't know the transitions available until you actually reach a node.
- The number of transitions may be so large that it would be impossible to consider all the transitions from a particular node.

Re-visiting the Colchester-Cambridge problem

If you didn't have a map (no state transitions defined) or a table of town-town distances (no quantitative heuristic), then you couldn't search for a route in advance.

What could you do?

Pick the road that seems to be heading in the right direction (we may call it qualitative heuristic).

When you get to the next town, repeat.

(Of course this requires a rough knowledge of the spatial arrangement of the towns)

This is a simple example of what is called ***hill climbing***.

Hill Climbing (3)

Why is it called hill climbing?

Suppose you want to climb a mountain on a misty day.

How could you select a route?

REPEAT

Select direction that goes most steeply uphill.

Advance 5 metres (5 is just an example)

UNTIL All directions lead downhill (at a peak).

Will this always work?

It will always get you to a summit within 5 metres

But it may not be the highest point of the mountain.

Such a subsidiary peak is called a ***local maximum***.

A local maximum is higher than its immediate surroundings but may not be the ***global maximum***.

Hill Climbing (4)

Hill Climbing

- Finds local maxima.
- May not find the global maximum
- Is a heuristic search technique
- Explores a single route through state space.
- Is the only practical approach in some real problems.

Hill Climbing and Greedy Search:

Is greedy search an example of hill climbing?

Not quite:

- Greedy search allows back tracking.
- Hill climbing does not.

Summary

*How to obtain
reliable and
accurate heuristics?*

Greedy Search

Expand node with smallest $h(n)$

Quick but not optimal

A* Search

Expand node with smallest $g(n)+h(n)$

Optimal (if the heuristic is admissible)

Efficient with a good heuristic

Hill Climbing (for self study)

Only practical approach in some real problems

May not find the global maximum