G51FAI
Fundamentals of AI

Instructor: Siang Yew Chong

Heuristic Searches

Outline

- Defining Heuristic Search
  - $g(n)$, $h(n)$
- Best-First Search
  - Greedy vs. A*
  - Implementation
- Admissibility and Monotonicity
- Heuristics
  - Informedness
  - Relaxed Problems
  - Effective Branching Factor

Heuristic Search

- Add ***domain-specific information to select the better path*** along which to continue searching
- Sometimes known as ***informed search***, it is usually more efficient than blind searches
- Heuristic search works by ***deciding*** which is the ***next best (guess) node to expand*** (but ***no guarantee*** it is the ***best node*** along the ***best path/solution***)

Heuristics

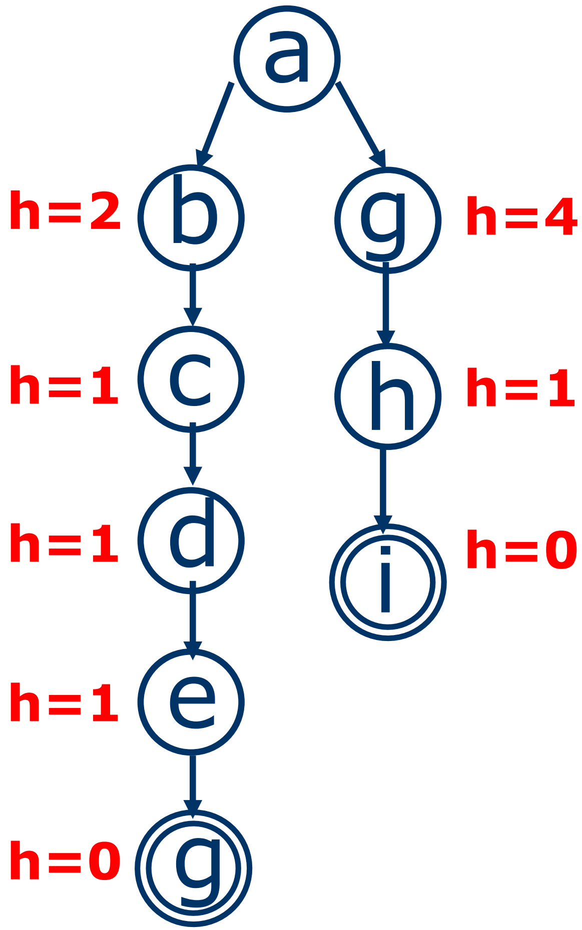
- Heuristic function $h(n)$ estimates the goodness of a node n
  - ***h(n) is the estimated cost (or distance) of minimal cost path from n to a goal state***
- All domain knowledge used in the search is encoded in $h(n)$, which is computable from the current state description
- General properties:
  - $h(n) \geq 0$ for all nodes n
  - $h(n) = 0$ implies n is a goal node

Best-First Search

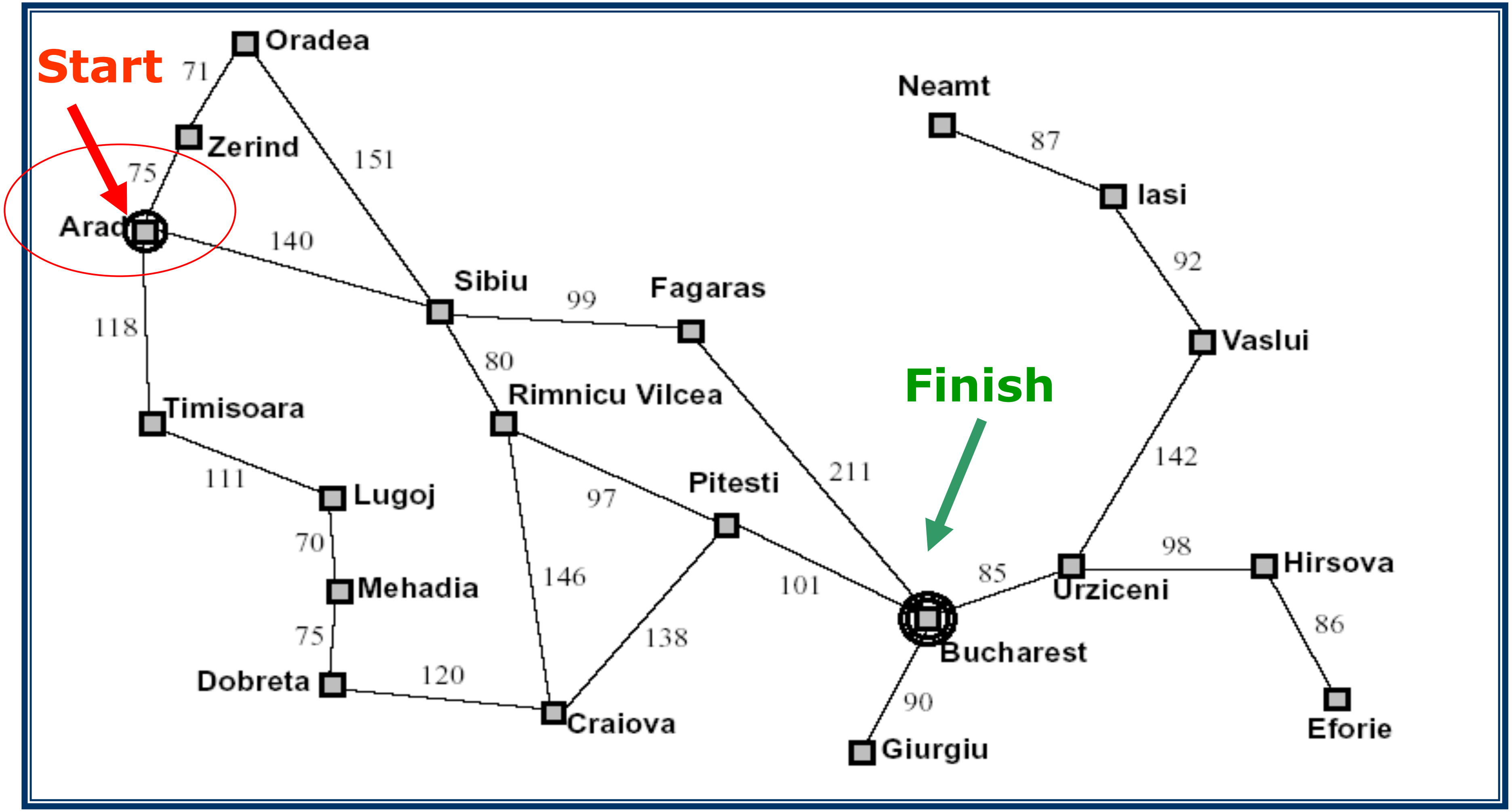
- This is a generic way of referring to the ***class of informed methods***
- Node selected for expansion based on an ***evaluation function*** $f(n)$, which ***incorporates heuristics*** in some way
- We get ***different searches*** using different $f(n)$
  - ***greedy search*** uses ***estimated cost*** from the ***current position to the goal*** or ***heuristic function*** $h(n)$
  - ***A* search*** uses a ***combination*** of the ***actual cost to reach the current node from root*** and the estimated cost, i.e. $g(n) + h(n)$
- Compare this with ***uniform cost search*** uses $g(n)$ only

Greedy Search

- Use as an evaluation function $f(n) = h(n)$, sorting nodes by increasing values of f
- Selects node to expand believed to be *closest* (hence “greedy”) to a goal node (i.e., select node with smallest f value)
- Not complete
- Not admissible, as in the example.
  - assuming all arc costs are 1, then greedy search will find goal g, which has a solution cost of 5
  - however, the optimal solution is the path to goal I with cost 3



Heuristic Search - Example



Heuristic Search - Example

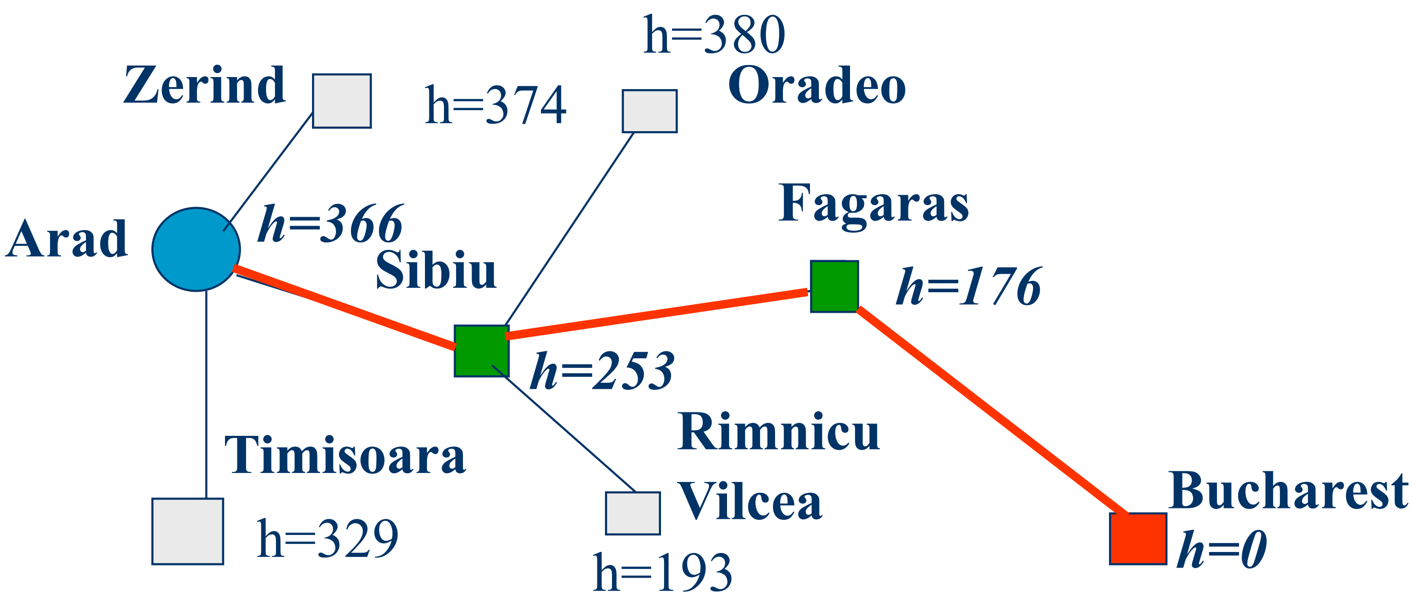
City	SLD
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

Town	SLD
Mehadai	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

SLD: Straight line distance between a given city and Bucharest

Greedy Search

The 1st node to be selected from the generated nodes is Sibiu because it is closer to Bucharest than either Zerind or Timisoara.



Greedy Search

Notice that : It finds solution without ever expanding a node that is NOT on the solution path.

The solution path is not optimal

Arad→ Sibiu→ Rimnicu Vilcea→ Pitesti→ Bucharest

with path cost of (140+80+97+101 = 418) is 32km LESS than

Arad→ Sibiu→ Fagaras→ Bucharest

(path cost = 140+99+211 = 450)

Greedy Search

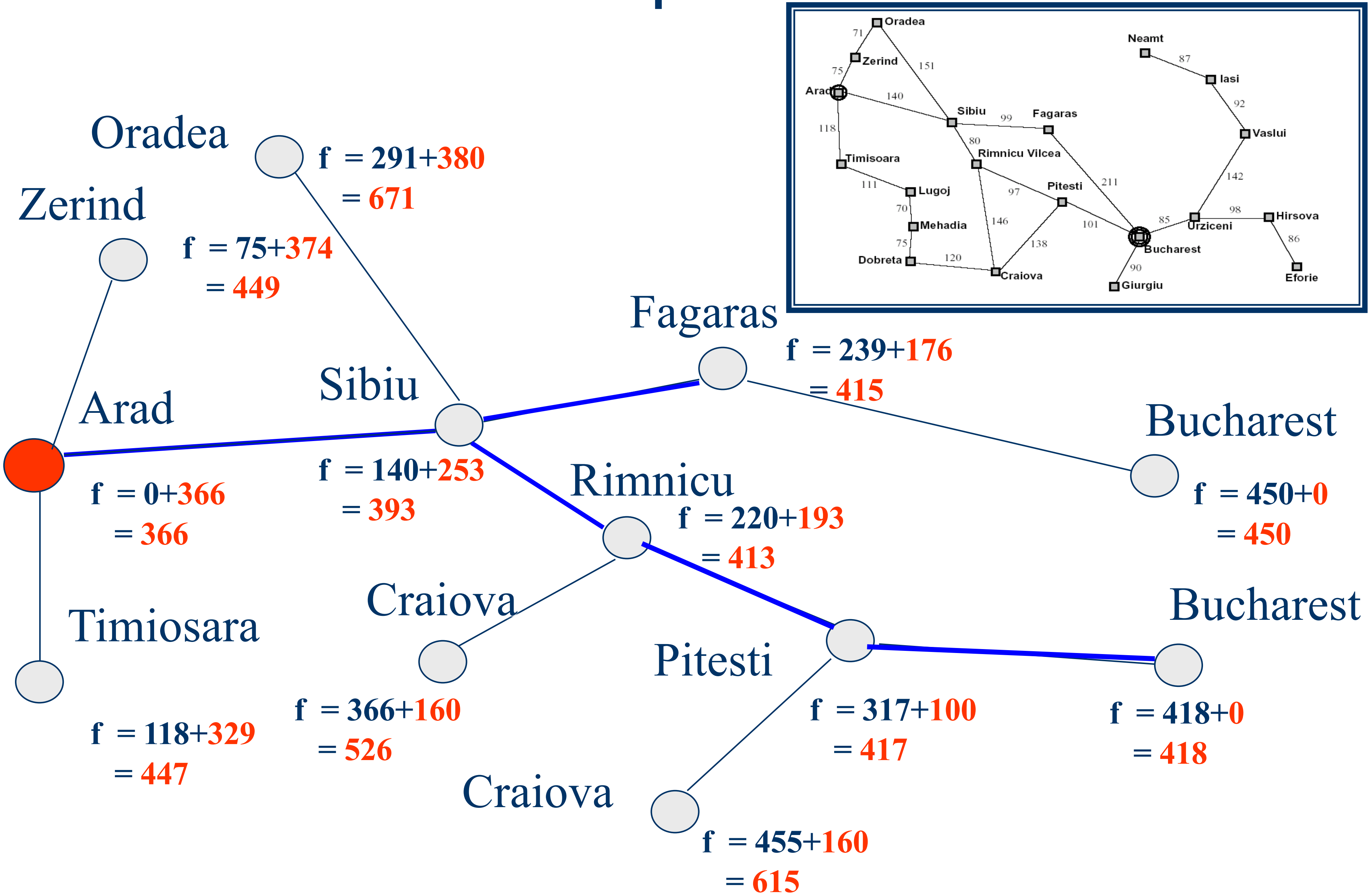
- It is only concerned with short term gains
- It is possible to get stuck in an infinite loop (consider being in Iasi and trying to get to Fagaras) unless mechanism for avoiding repeated states is in place
- It is not optimal
- It is not complete

Time and space complexity is $O(B^m)$; where *m* is the depth of the search tree

A* Search

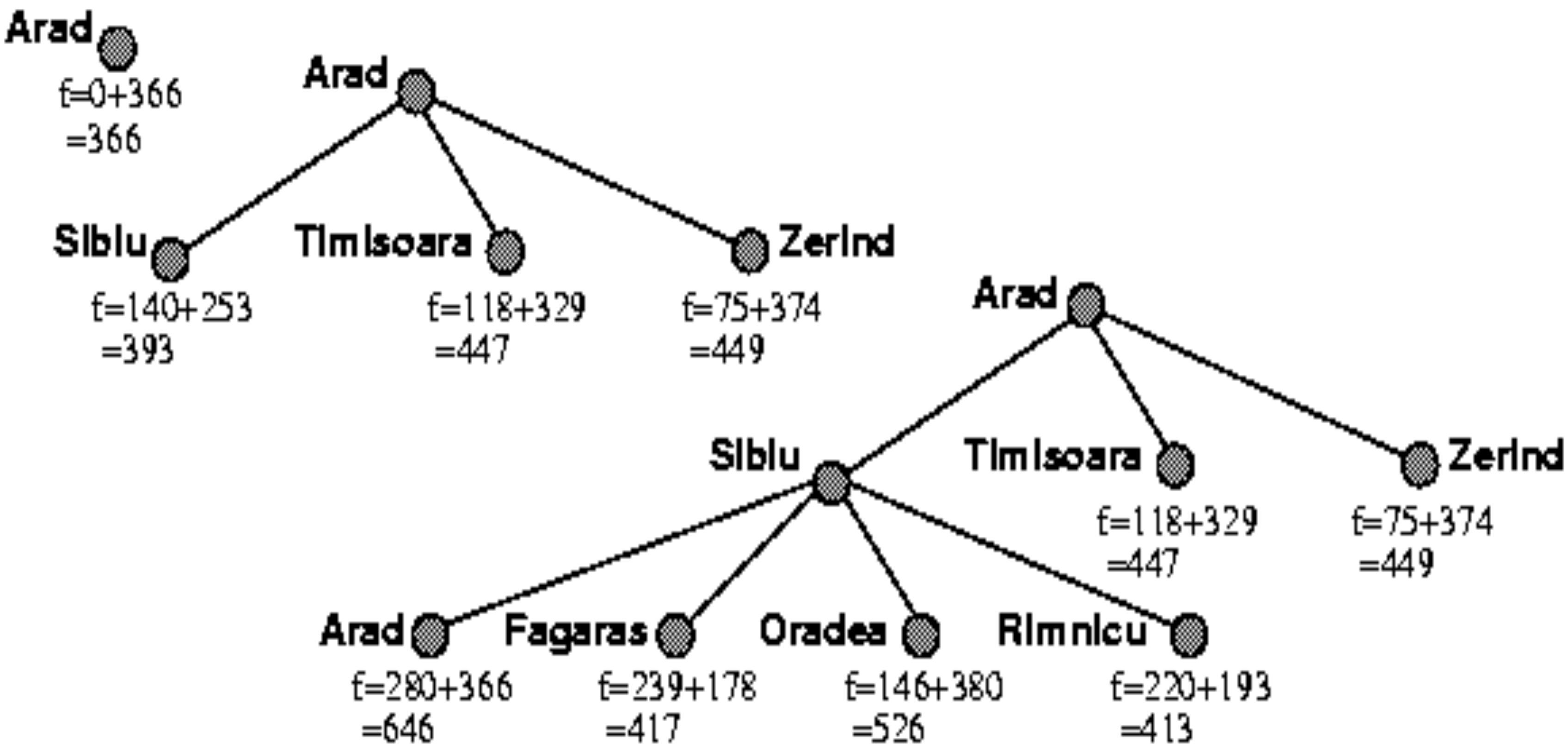
- Best-known form of best-first search
- Idea: avoid expanding paths that are already expensive.
- Evaluation function $f(n)=g(n) + h(n)$ → A*
  - $g(n)$ the cost (so far) to reach the node
  - $h(n)$ estimated cost to get from the node to the goal
  - $f(n)$ estimated total cost of path through n to goal
- It can be ***proved*** to be ***optimal*** and ***complete*** if heuristic function is ***admissible***, i.e. $h(n)$ never overestimate the cost to reach the goal: $h(n) <= h^*(n)$, true cost $h^*(n)$

A* Search - Example

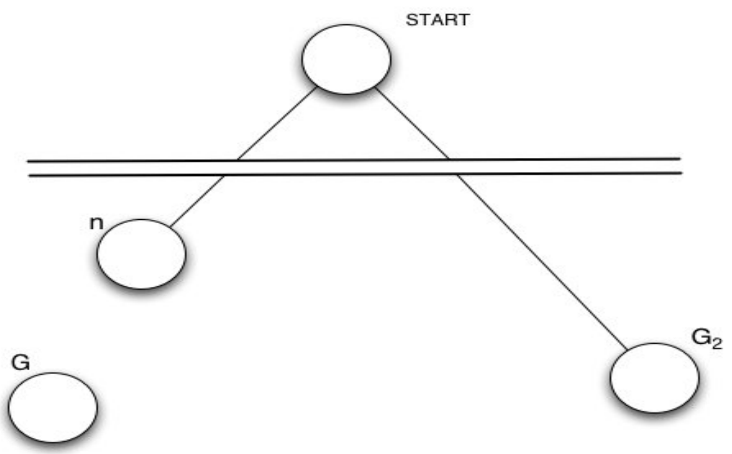


A* Search

Tree Search will give an optimal solution if h is admissible



Optimality of A* (Standard Proof)



- Suppose suboptimal goal G_2 in the queue
- Let n be an unexpanded node on the shortest path to optimal goal G.

$f(G_2)$	$= g(G_2)$	since $h(G_2) = 0$ (definition of G_2)
	$> g(G)$	since G_2 is suboptimal
	$>= f(n)$	since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

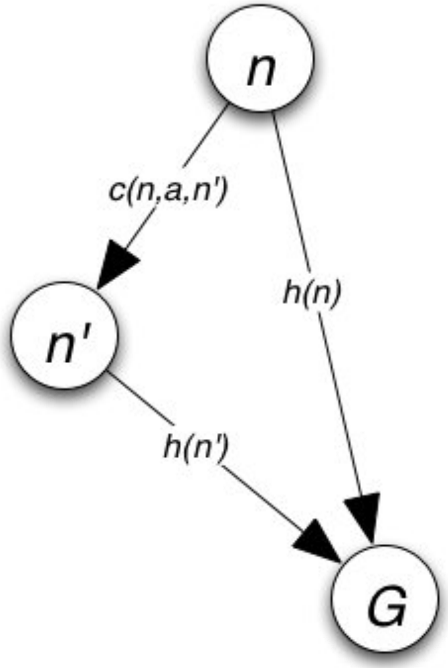
BUT … Graph Search

- Discards new paths to repeated state
  - previous proof breaks down
- Solution:
  - add extra bookkeeping i.e. remove more expensive of two paths
  - ensure that optimal path to any repeated state is always first followed
    - extra requirement on *h(n)*: consistency (monotonicity)
- If *h* is consistent, then A* using Graph-Search is optimal

Monotonicity (Consistency)

- A heuristic is consistent if

$$h(n) \le c(n,a,n') + h(n')$$



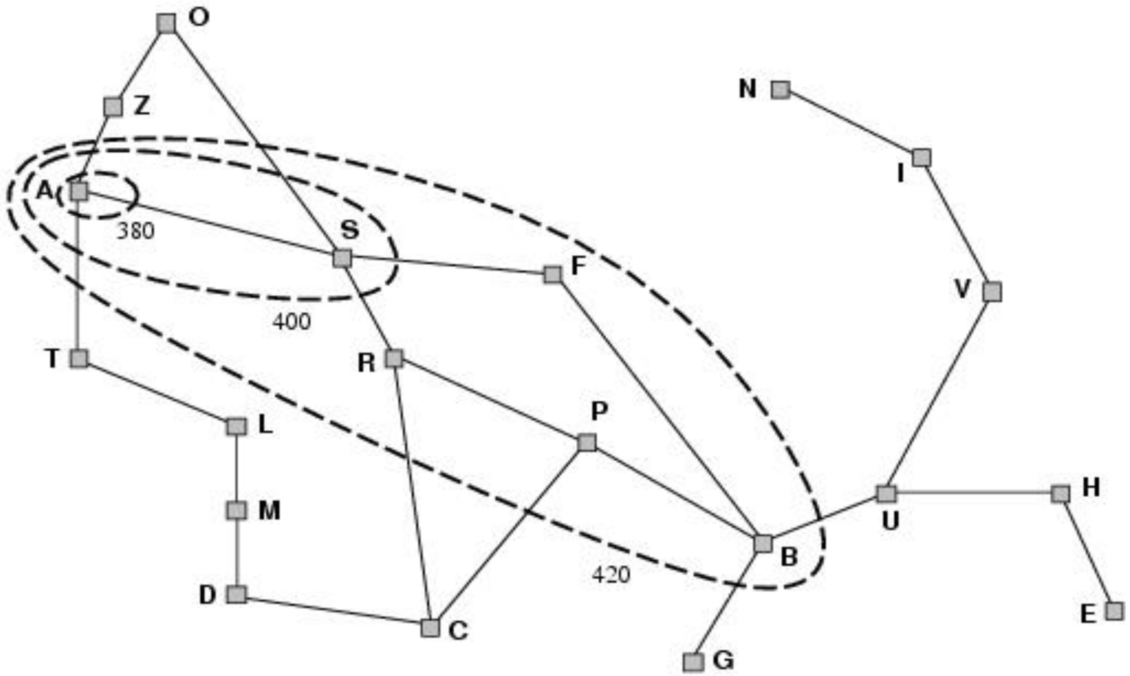
- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &\ge f(n) \end{aligned}$$

i.e. f(n) is nondecreasing along any path

- The first goal node selected for expansion must be an optimal solution

Optimality of A* Search



- A^* expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f <= f_i$ where $f_i < f_{i+1}$

Monotonicity and Admissibility

Any monotonic heuristics is also admissible.
Crucially, imposed metric property to search

This argument considers any path in the search space as a sequence of states s_1, s_2,……s_g, where s_1 is that start state and s_g is the goal. For a sequence of moves in this arbitrarily selected path, monotonicity dictates that:

s_1 to s_2	$h(s_1) - h(s_2) \leq c(s_1, a, s_2)$
s_2 to s_3	$h(s_2) - h(s_3) \leq c(s_2, a, s_3)$
s_3 to s_4	$h(s_3) - h(s_4) \leq c(s_3, a, s_4)$
	….
	….
s_{g-1} to s_g	$h(s_{g-1}) - h(s_g) \leq c(s_{g-1}, s_g)$

Summing each column and using the monotone property of $h(s_g) = 0$

Path s_1 to s_g $\quad h(s1) \leq g(s_g)$

8-Puzzle Example

Initial State

5	4	
6	1	8
7	3	2

Goal State

1	2	3
8		4
7	6	5

A* Algorithm

Typical solution is about twenty steps

Branching factor is approximately three. Therefore a complete search would need to search 3^{20} states. But by keeping track of repeated states we would only need to search 9! (362,880) states

But even this is a lot (imagine having all these in memory)

Our aim is to develop a heuristic that does not overestimate (it is admissible) so that we can use A* to find the optimal solution

Possible Heuristics

h_1 = the number of tiles that are in the wrong position (=7)

h_2 = the sum of the distances of the tiles from their goal positions using the Manhattan Distance (=18)

Both are admissible but which one is better?

Initial State

5	4	
6	1	8
7	3	2

Goal State

1	2	3
8		4
7	6	5

Fig 3.17 Breadth-first search of the 8-puzzle, showing order in which states were removed from open

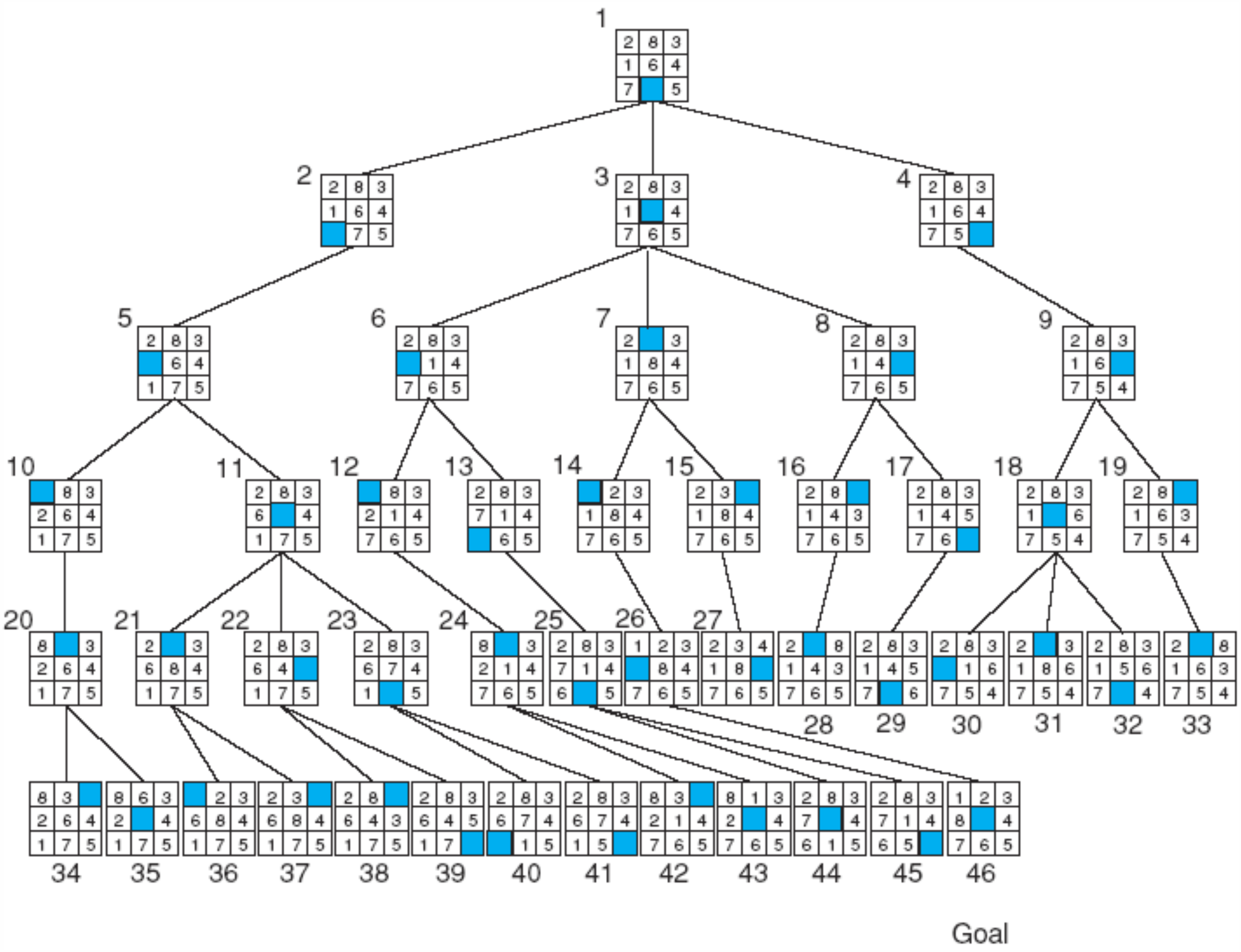
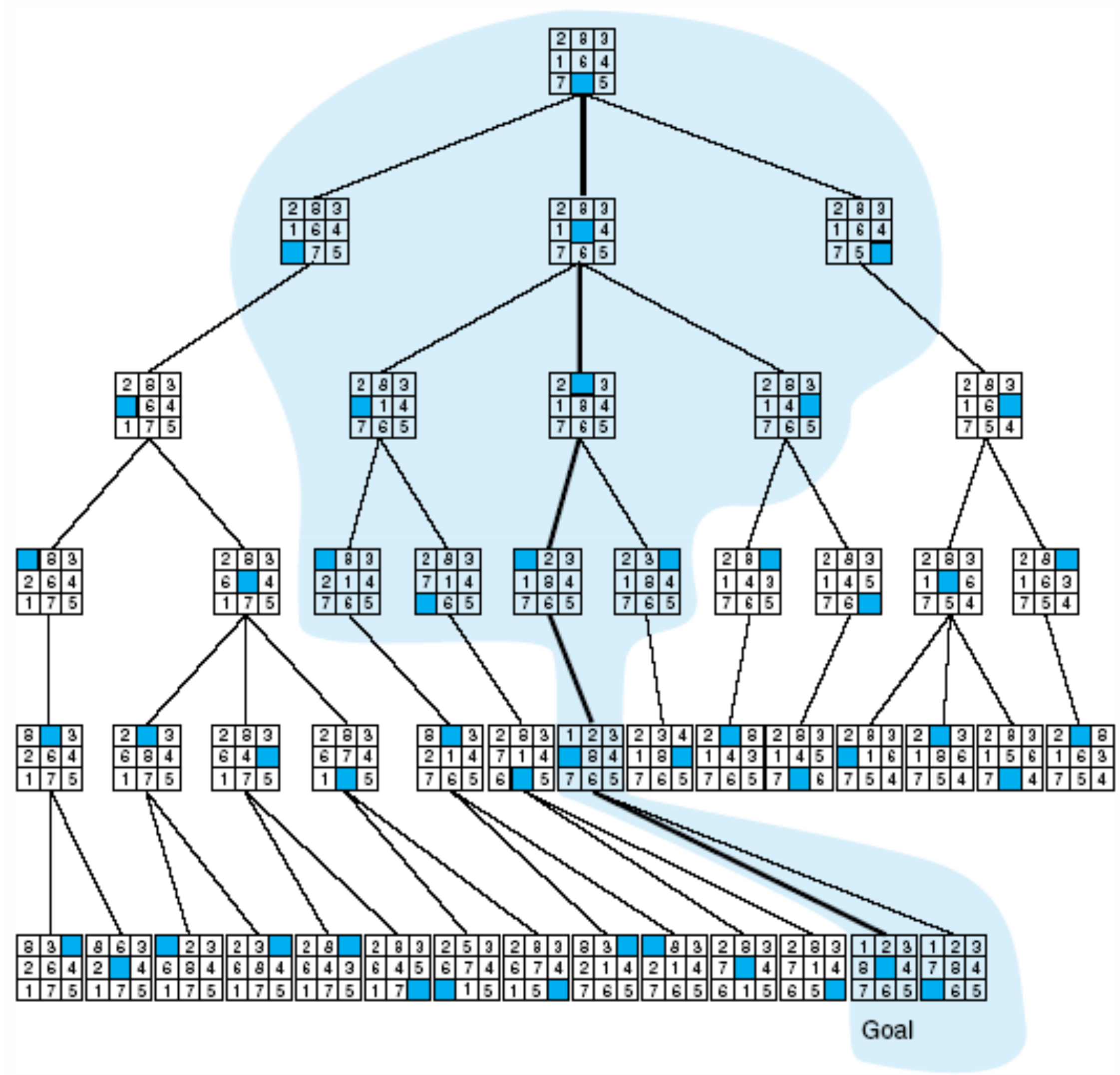


Fig 4.18 Comparison of state space searched using heuristic search with space searched by breadth-first search. The proportion of the graph searched heuristically is shaded. The optimal search selection is in bold. Heuristic used is f(n) = g(n) + h(n) where h(n) is tiles out of place



Informedness

For two A^* heuristics h_1 and h_2, **if $h_1(n) <= h_2(n)$, for all states n** in the search space, we say ***h_2 dominates h_1*** or heuristic h_2 is more informed than h_1.

Domination translate to efficiency: A^* using h_2 will never expand more nodes than A^* using h_1.

Hence it is always better to use a heuristic function with higher values, provided it does not over-estimate and that the computation time for the heuristic is not too large

Generating Heuristics with Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Test From 100 Runs with Varying Solution Depths

		Search Cost	
Depth	IDS	A*(h_1)	A*(h_2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
16		1301	211
18		3056	363
20		7276	676
22		18094	1219
24		39135	1641

h_2 looks better as fewer nodes are expanded. But why?

Effective Branching Factor (EBF)

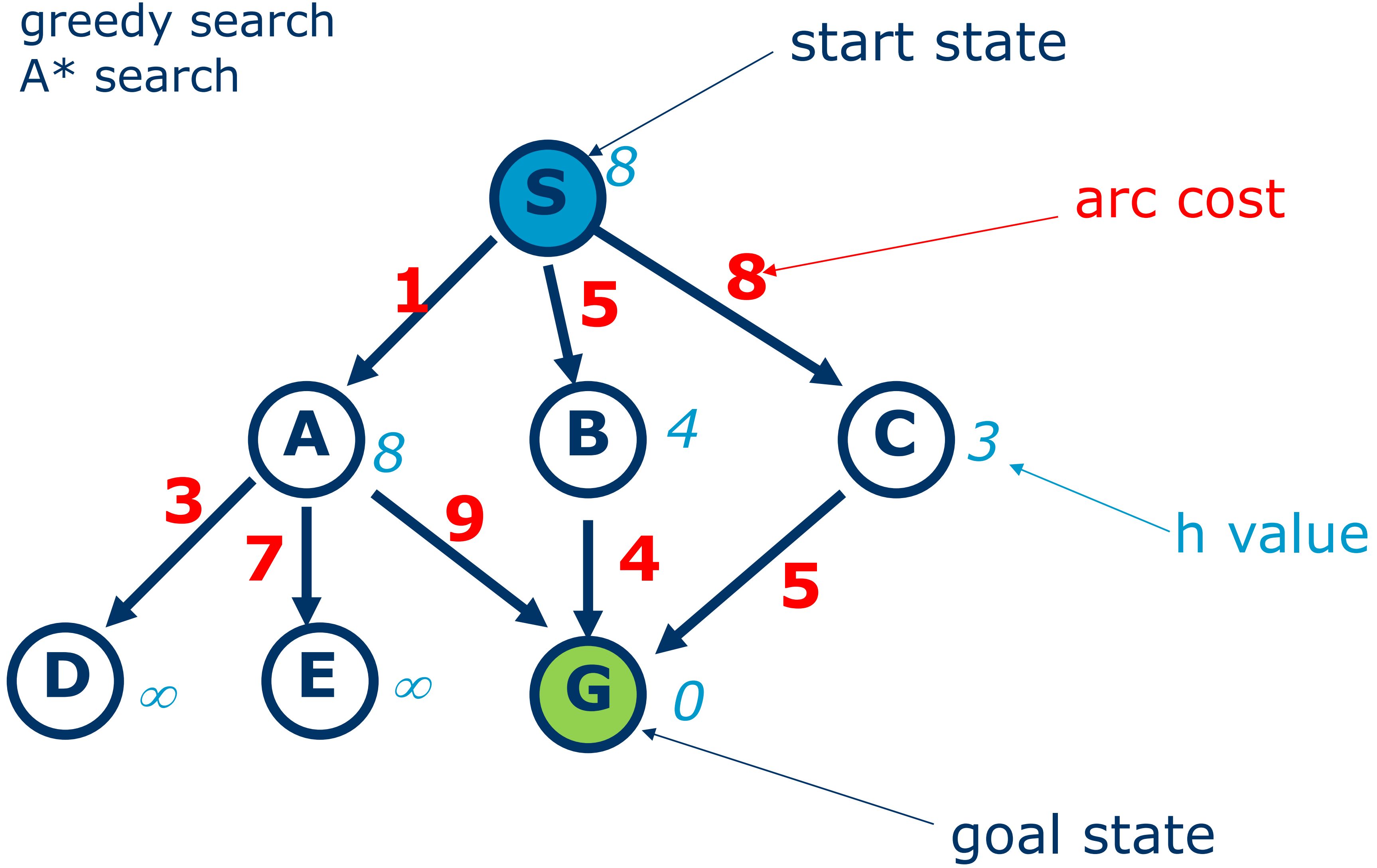
	Search Cost			EBF		
Depth	**IDS**	**A*(h_1)**	**A*(h_2)**	**IDS**	**A*(h_1)**	**A*(h_2)**
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23

- Effective branching factor: average number of branches expanded
- h_2 has a lower branching factor and so fewer nodes are expanded
- Therefore, one way to ***measure the quality of a heuristic is to find its average branching factor***
- h_2 has a lower EBF and is therefore the better heuristic

Example

Work out the solution path using:

- greedy search
- A* search



Summary

- Heuristic search
  - characteristics
  - h(n), g(n)
  - best-first-search
    - greedy-search
    - A*
- Conditions for Optimality
- Heuristics
  - informedness & EBF
  - relaxed problem

Acknowledgements

Most of the lecture slides are adapted from the same module taught in Nottingham UK
by
Professor Graham Kendall,
Dr. Rong Qu
and
Dr. Andrew Parker