Heuristic Search

Informed Search

CIS 32

Functionally

Today:

Heuristic (Informed) Search

From Improvements to Uninformed Search...

- More advanced problem solving techniques:
  - Depth limited search
  - Iterative deepening
  - Bidirectional search
- These improved on basic techniques like breadth-first and depth-first search.
- Still aren't powerful enough to give solutions for realistic problems.
- Are there more improvements we can make?

(What if we include state-space specific knowledge to our search process?)

... to Informed Search

Aims of this lecture:

- To show how applying some knowledge of the problem can help.
  - Introduce *heuristics* — rules of thumbs
- Introduce *heuristic search*: rules of thumbs which dictate which node to expand on the fringe
  - – *uniform-cost search*
  - – *greedy search*
  - – *A* search*

Heuristic (Informed) Search

- Whatever search technique we use, *exponential time complexity*.
- Tweaks to the algorithm (depth-limited etc...) will not reduce this to polynomial.
- We need **problem specific knowledge** to guide the search.
- Simplest form of problem specific knowledge is *heuristic*.
- Usual implementation in search is via an **evaluation function** which indicates *desirability* of expanding node.

Uniform Cost Search

- Recall we have a *path cost function (g(x))*,

$$g : Nodes \rightarrow R$$

  which gives cost to each path.

- Why not expand the *cheapest* path first?
- Intuition: cheapest is likely to be best!

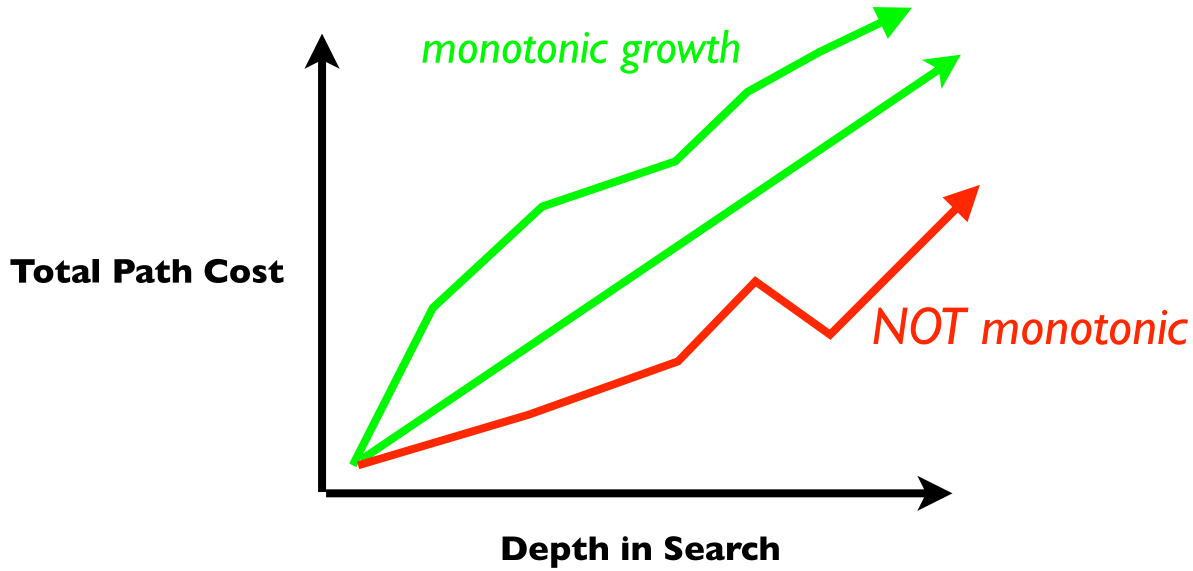
Uniform Cost Algorithm

```
agenda = initial state;
while agenda not empty do
{
   take node from agenda such that
   g(node) = min { g(n) | n in agenda}
   new nodes = apply operations to node;
   if goal state in new nodes then {
      return solution;
   }
   else add new nodes to agenda
}
```

Uniform Cost Search

• Uniform cost search guaranteed to find cheapest solution *assuming path costs grow* ***monotonically***.

• In other words, adding another step to the solution makes it more costly.



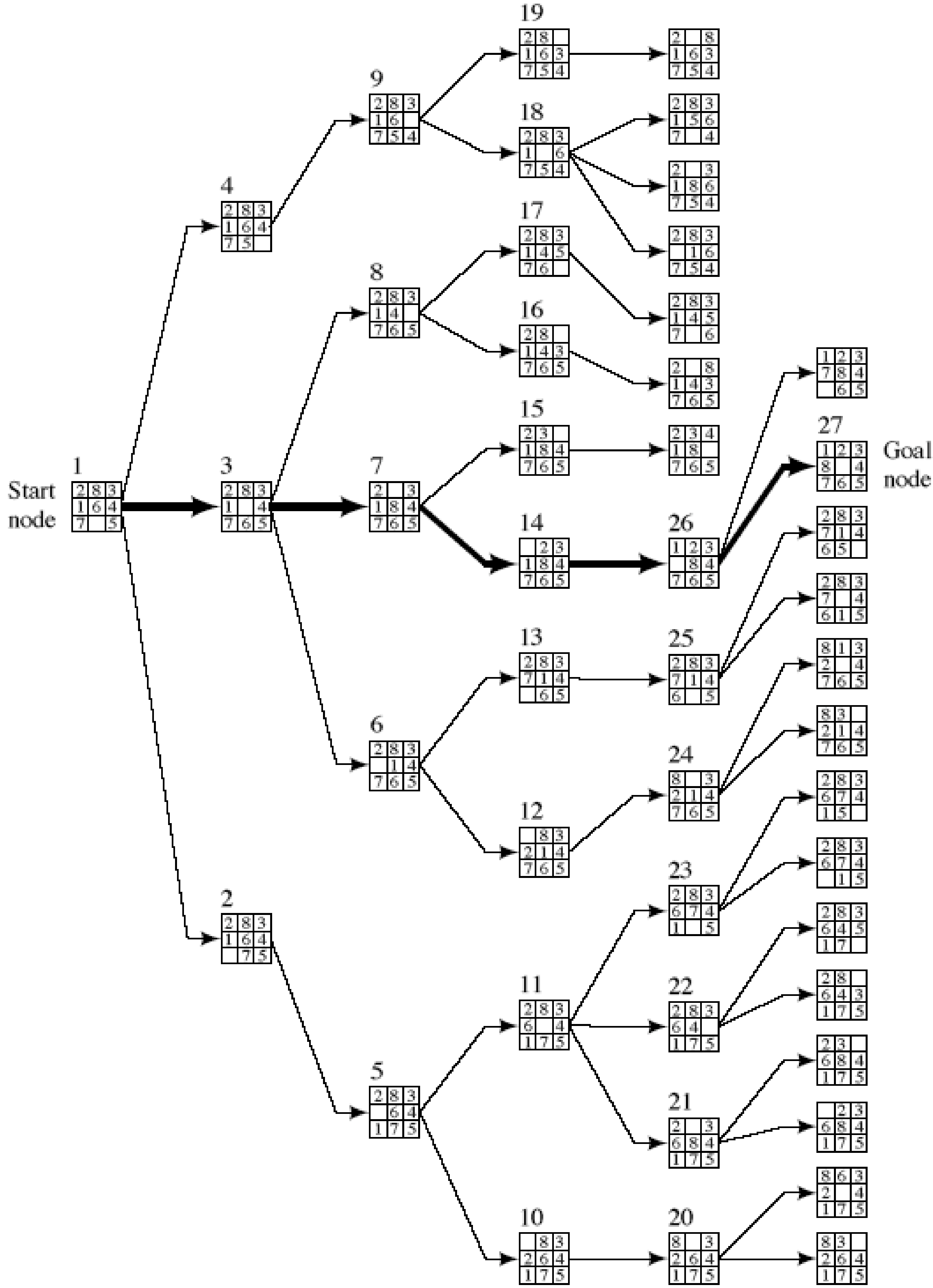
• If path costs *don't* grow monotonically, then exhaustive search is required.

8-puzzle Example

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

Search Space is:

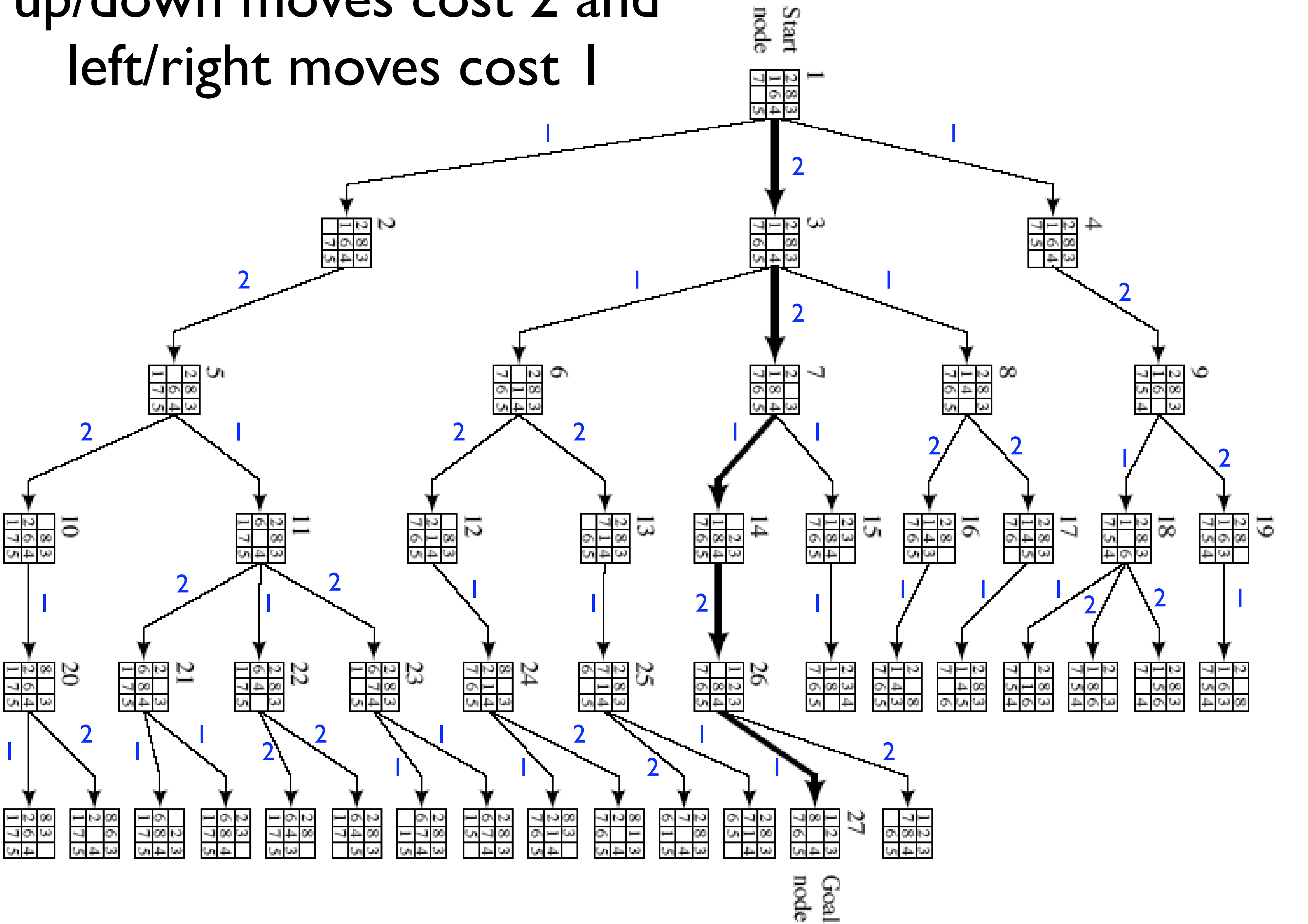


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Order of Expansion

- States would be expanded in the order:
    1. 1
    2. 2, 3, 4
    3. 5, 6, 7, 8, 9
    4. 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
    5. . . .
- Note that this is just like *breadth first search* (because the path costs are **all the same**).

up/down moves cost 2 and left/right moves cost 1



States would be expanded in the order:

1. 1
2. 2, 3, 4
3. 5
4. 9
5. 6, 7, 8
6. . . .

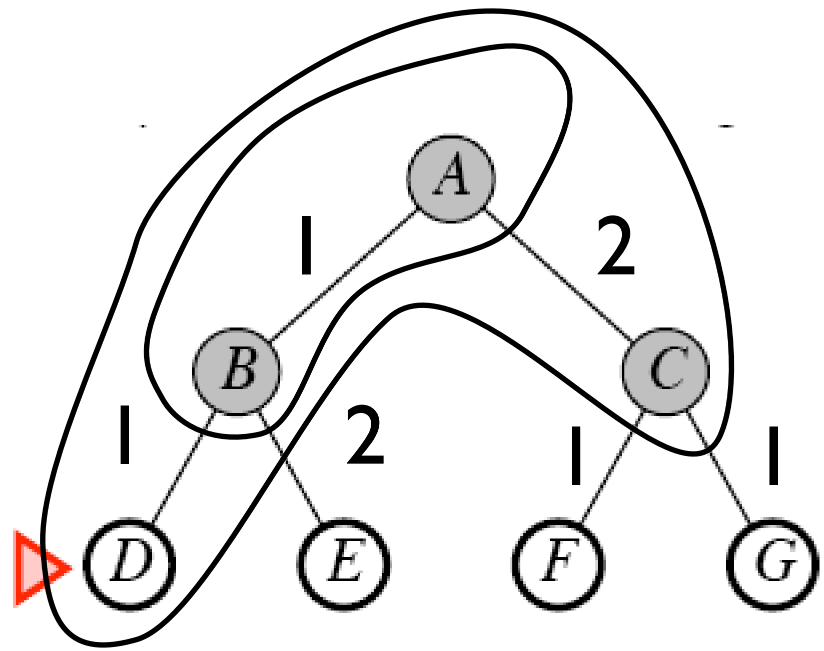
Uniform Cost Search

Costs are associated to traversing the nodes (i.e. doing an operation)

Take into consideration the cost of the operation.

Explore nodes according to a uniform cost "depth"

Every operation having a cost of 1 is *breath-first search*.



Operation costs can be thought of as contours

Greedy Search

- Most heuristics *estimate cost of cheapest path from node to solution.*
- We have a *heuristic function*,

$$h : Nodes \rightarrow R$$

which estimates the distance from the node to the goal.

- **Example**: In route finding, heuristic might be straight line distance from node to destination.
- Heuristic is said to be *admissible* if it never *overestimates* cheapest solution.

Admissible = optimistic.

- Greedy search involves *expanding node with cheapest expected cost to solution*.

Greedy Search Algorithm

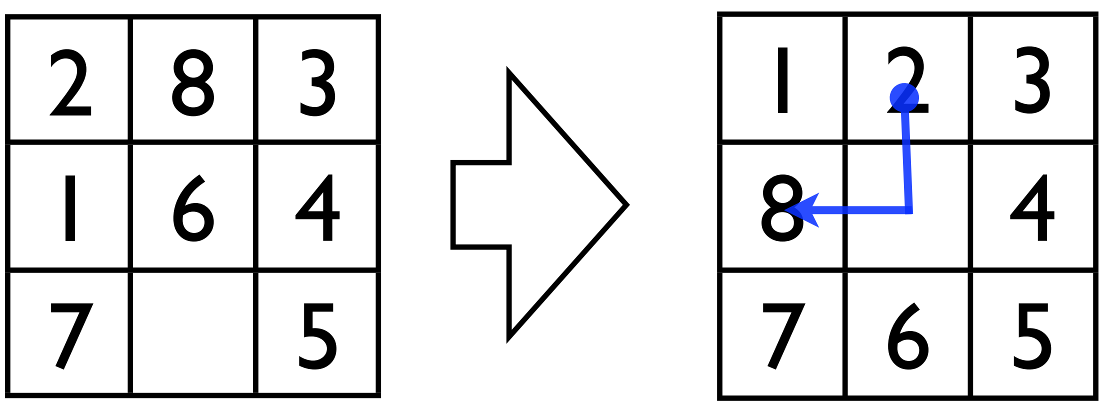
```
agenda = initial state;
while agenda not empty do
{
   take node from agenda such that
   h(node) = min { h(n) | n in agenda}
   new nodes = apply operations to node;
   if goal state in new nodes then {
      return solution;
   }
   else add new nodes to agenda
}
```

Greedy Search

- Greedy search finds solutions quickly.
- Doesn't always find best.
- Susceptible to *false starts*.
  - Chases good looking options that turn out to be bad.
- Only looks at *current* node. Ignores past!
- Also *myopic* (shortsighted).

Heuristics for 8-puzzle

- For the 8-puzzle one good heuristic is:
  - count tiles out of place.
- Another is:
  - Manhattan blocks' distance
- The latter works for other problems as well:
  - Robot navigation.

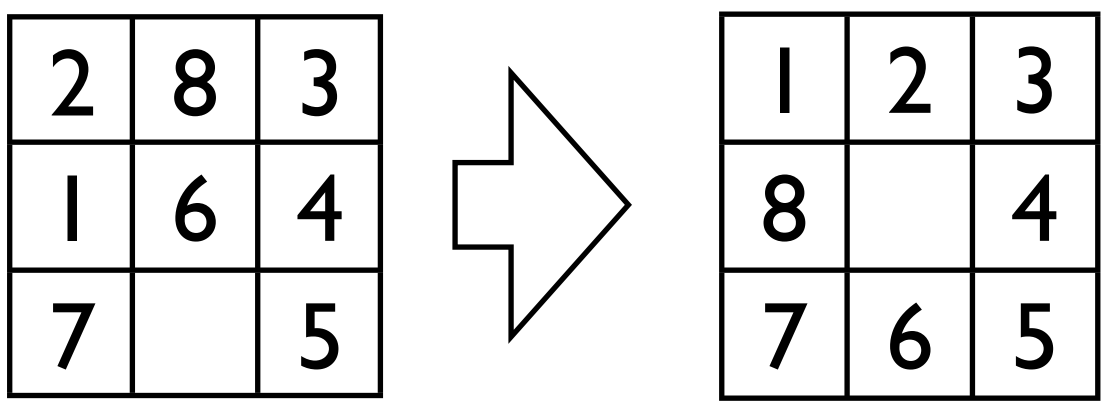


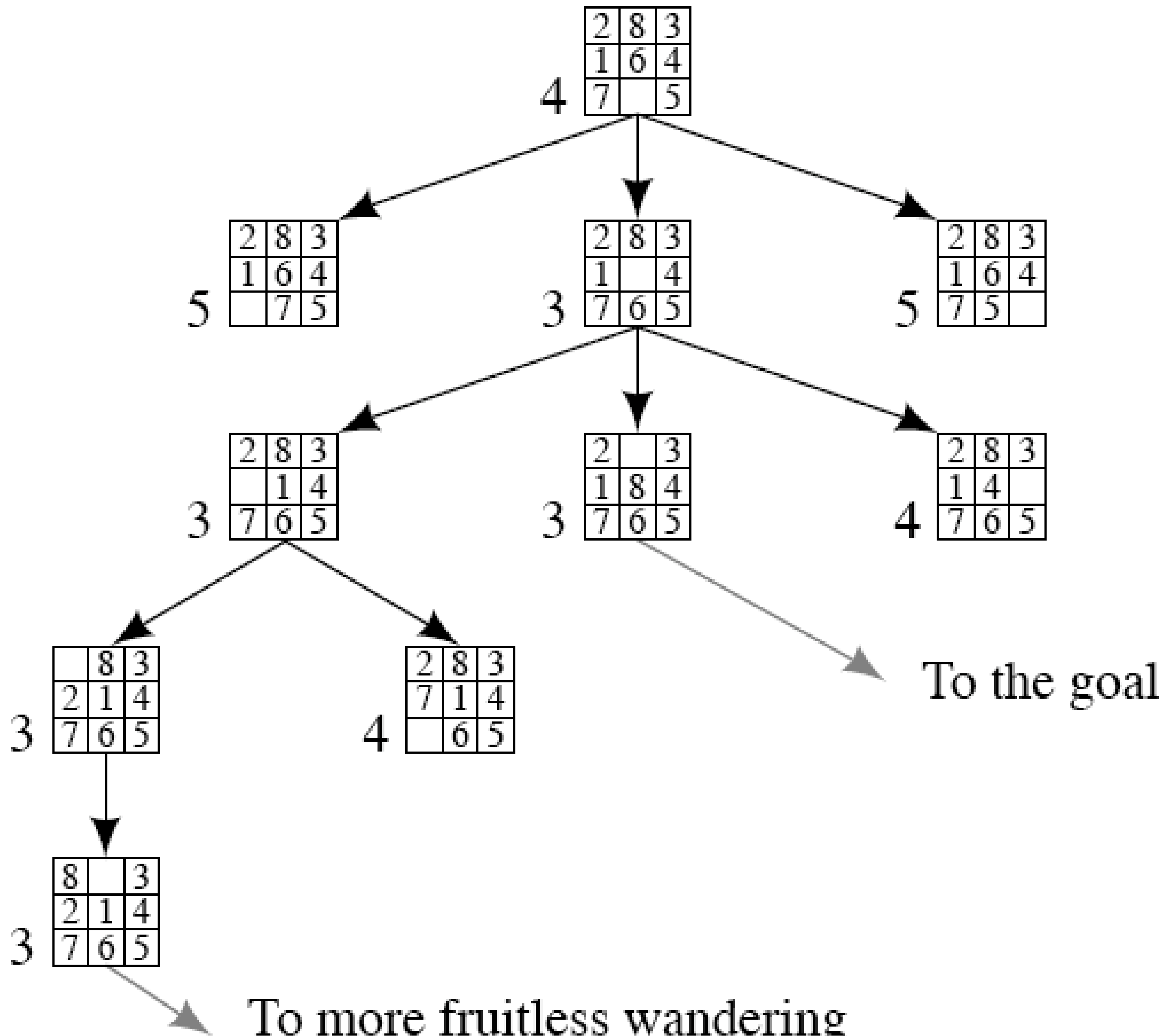
Heuristics for 8-puzzle

- For the 8-puzzle one good heuristic is:
  - count tiles out of place. $h = 6$
- Another is:

(equal by coincidence only)

  - Manhattan blocks' distance $h = 1+2+1+1+1 = 6$
- The latter works for other problems as well:
  - Robot navigation.




To the goal
To more fruitless wandering

A* Search

- A* is very efficient search strategy.
- Basic idea is to *combine*

uniform cost search
and
greedy search.

- We look at the *cost so far* (***past***) and the *estimated cost to goal* (***future***).
- Gives heuristic *f*:

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

where

  - *g*(n) is path cost of n;
  - *h*(n) is expected cost of cheapest solution from n.

- Aims to *minimize overall cost.*

A* search

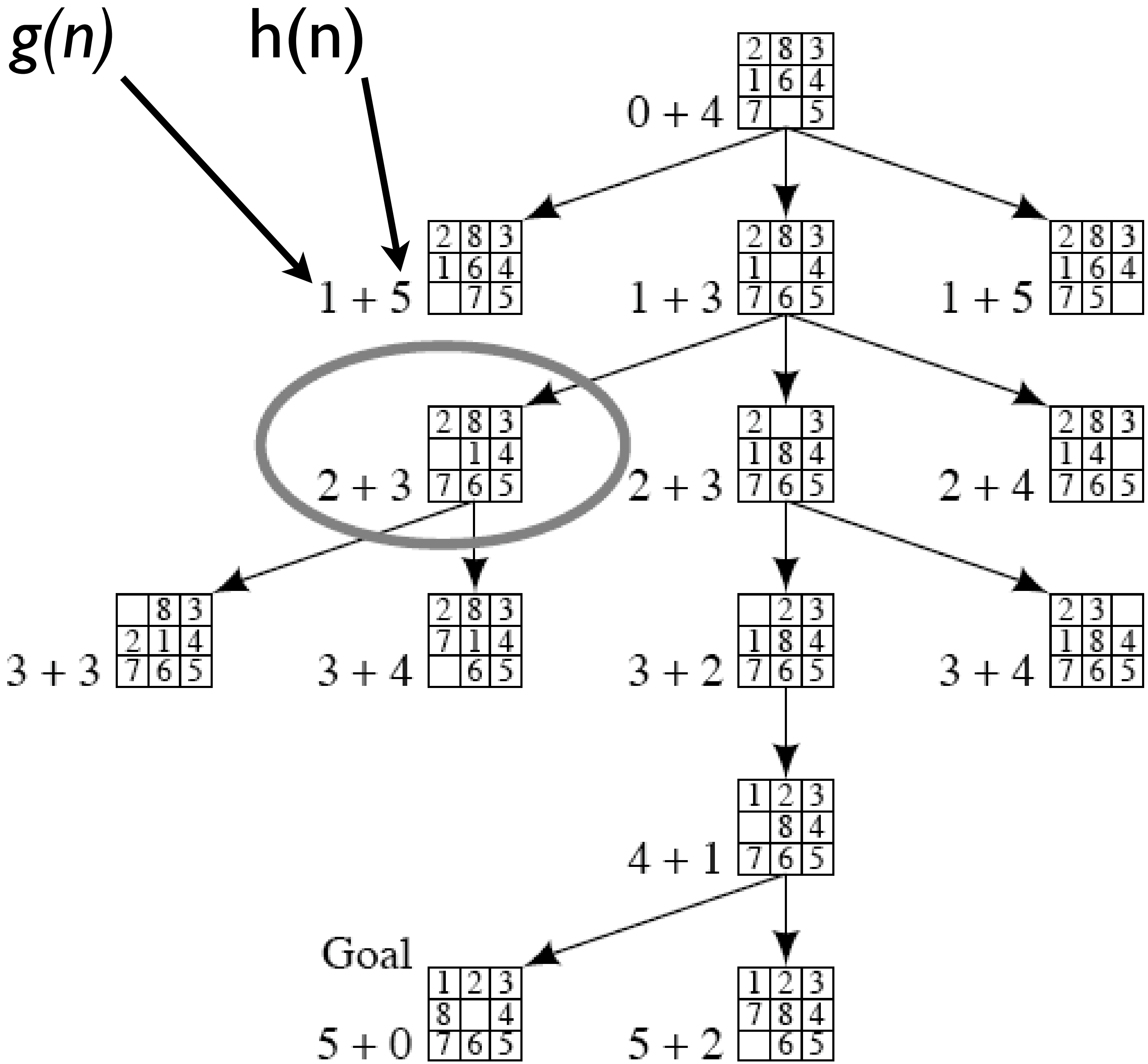
```
agenda = initial state;
while agenda not empty do
{
   take node from agenda such that
   f(node) = min { f(n) | n in agenda}
   where f(n) = g(n) + h(n)
     new nodes = apply operations to node;
   if goal state in new nodes then {
     return solution;
   }
   else add new nodes to agenda
}
```

8-puzzle : A* Search Heuristic Function

- Considering the 8-puzzle (for the last time :-):
- We combine:
  - Path cost function *g(n)*:

    number of moves.
  - Heuristic function *h(n)*:

    tiles out of places.
- This gives the following search.


g(n)
h(n)
0 + 4
1 + 5
1 + 3
1 + 5
2 + 3
2 + 3
2 + 4
3 + 3
3 + 4
3 + 2
3 + 4
4 + 1
Goal
5 + 0
5 + 2

The optimality of A*

- A* is optimal in precise sense — it is guaranteed to find a minimum cost path to the goal.
- There are a set of conditions under which A* will find such a path:

    1. Each node in the graph has a finite number of children.
    2. All arcs have a cost greater than some positive ϵ.
    3. For all nodes in the graph *h(n)* always *underestimates* the true distance to the goal.

- The key here is number 3.— the notion of *admissibility*.
- We will express this by saying a *heuristic h(•)* is admissible if

$$h(n) \leq h_T(n)$$

More informed search

IF two versions of A*, A_1^* and A_2^* use different functions h_1 and h_2,

AND

$h_1(n) < h_2(n)$, for all non-goal nodes,

THEN

we say that A_2^* is more informed than A_1^*.

The better informed A* is, the less nodes it has to expand to find the minimum cost path.

- As an example of ”more informed” consider the 8-puzzle:

– tiles out of place; and

– Manhattan blocks distance.

- We need $h(n)$ to underestimate $h_T(n)$ to ensure *admissibility*.

- But, the closer the estimate, the easier it is to reject nodes which are not on the optimal path.

- This means less nodes need to be searched.

Iterative deepening A*

- When we do heuristic search, we search some portion of the full search space (i.e. "*Focussed breadth first search*").
- So, we can still hit *intractability*.
- Adapting iterative deepening can help us.
- Instead of a *depth limit*, we impose a *cost limit*, and do a depth first search until it is exceeded.
- Then we *backtrack*, and extend the limit if we don't find the goal.

Basic Algorithm of IDA*

- The initial cost cut off is set tc $f(n_0)$
- This is just the estimated cost of finding a solution $h(n_0)$. (g(n_0) = 0)
- This will never overestimate the cost, so is a good start point.
- If this cost-limit does not provide a solution, the next cost limit:
- (If the heuristic is a good one), the cost of the cheapest path to the goal will be the lowest $f(n)$ of an unexpanded node.
- So we set the new cost bound to this.
- This, then is *iterative deepening A** (IDA*).

Summary

- This lecture has looked at some techniques for refining the search space:
  - uniform cost search;
  - greedy search; and
  - A* search.
- When these work they explore just the relevant part of the search space.
- There are also techniques that go further than those we have studied.

(Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* 1984)

- There are three directions we will take from here:
  - Adversarial search
  - Learning the state space.
  - Adding in more knowledge about the domain.