HEURISTIC SEARCH II

Recap

The last lectures introduced

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

Overview

Aims of this lecture:

- To show how applying some knowledge of the problem can help.
- Introduce *heuristics* — rules of thumb.
- Introduce *heuristic search*: guided by rules of thumb which help to decide which node to expand:
  - *best-first search*;
  - *greedy search*;
  - *A* search*.

Heuristic (Informed) Search

- Whatever search technique we use, *exponential time complexity*.
- Tweaks to the algorithm 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*,

$$g : Nodes \to R$$

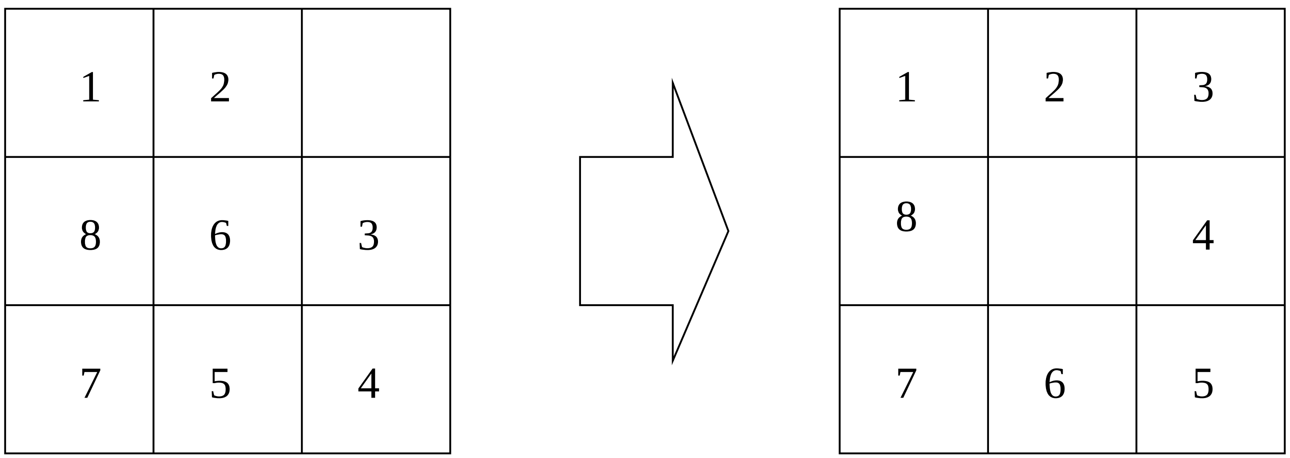
  which gives cost to each path.
- Why not expand the *cheapest* path first?
- Intuition: cheapest is likely to be best!

- General algorithm for uniform search:

```
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 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.

- Once again we can illustrate this on the 8-puzzle:



- For this set up…

- . . . the search will be as follows:

Greedy Search

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

$$h : Nodes \to R$$

  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.

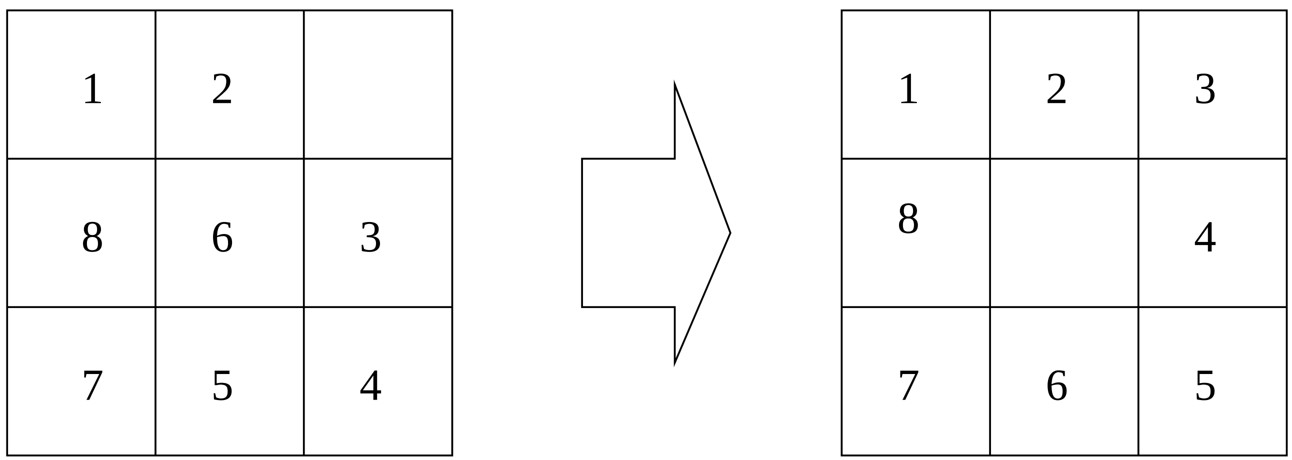
- Greedy search involves *expanding node with cheapest expected cost to solution*.

- General algorithm for greedy search:

```
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 finds solutions quickly.
- Doesn't always find best.
- Susceptible to false starts.
- Only looking at *current* node. Ignores past!
- *Short sighted*.

- For the 8-puzzle:



a good heuristic is the *Manhattan blocks' distance*

- Can also use the "tiles out of place" heuristic.

- Using this, the search will be as follows:

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* and the *estimated cost to goal*.
- Gives heuristic f:

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

  where

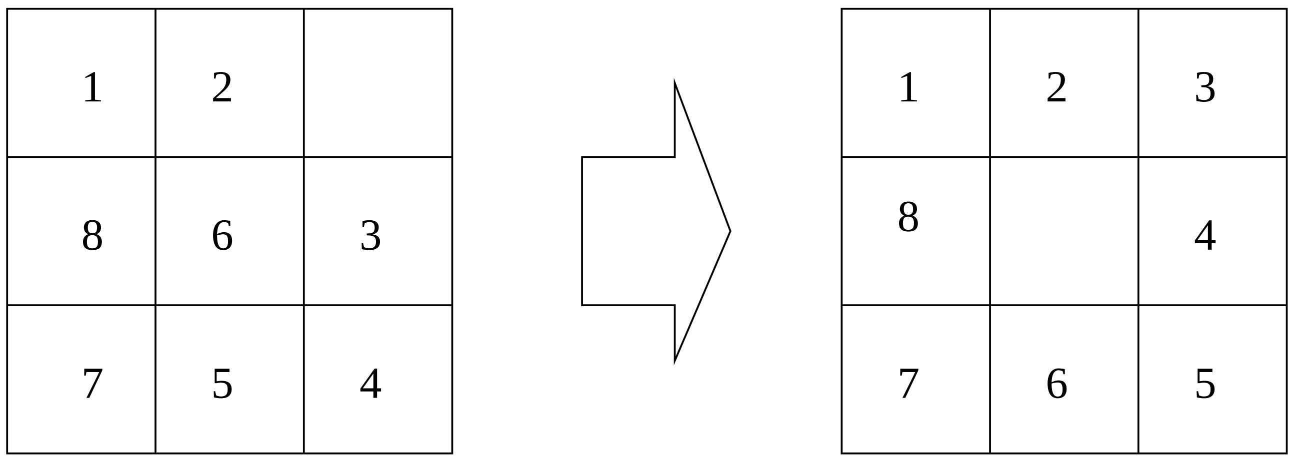
  – $g(n)$ is path cost of n;
  – $h(n)$ is expected cost of cheapest solution from n.

- Aims to mimimise *overall cost*.

- General algorithm for 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
}
```

- Considering the 8-puzzle (for the last time :-):



- We combine the *Manhattan blocks' distance* heuristic function, with a path cost which counts the number of moves.

- Using this, the search will be as follows:

Summary

- This lecture has looked at some techniques for refining the search space:
- 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.
  - iterative deepening A* search
- There are two directions we will take from here:
  - Adversarial search
  - Learning the state space.
  - Adding in more knowledge about the domain.