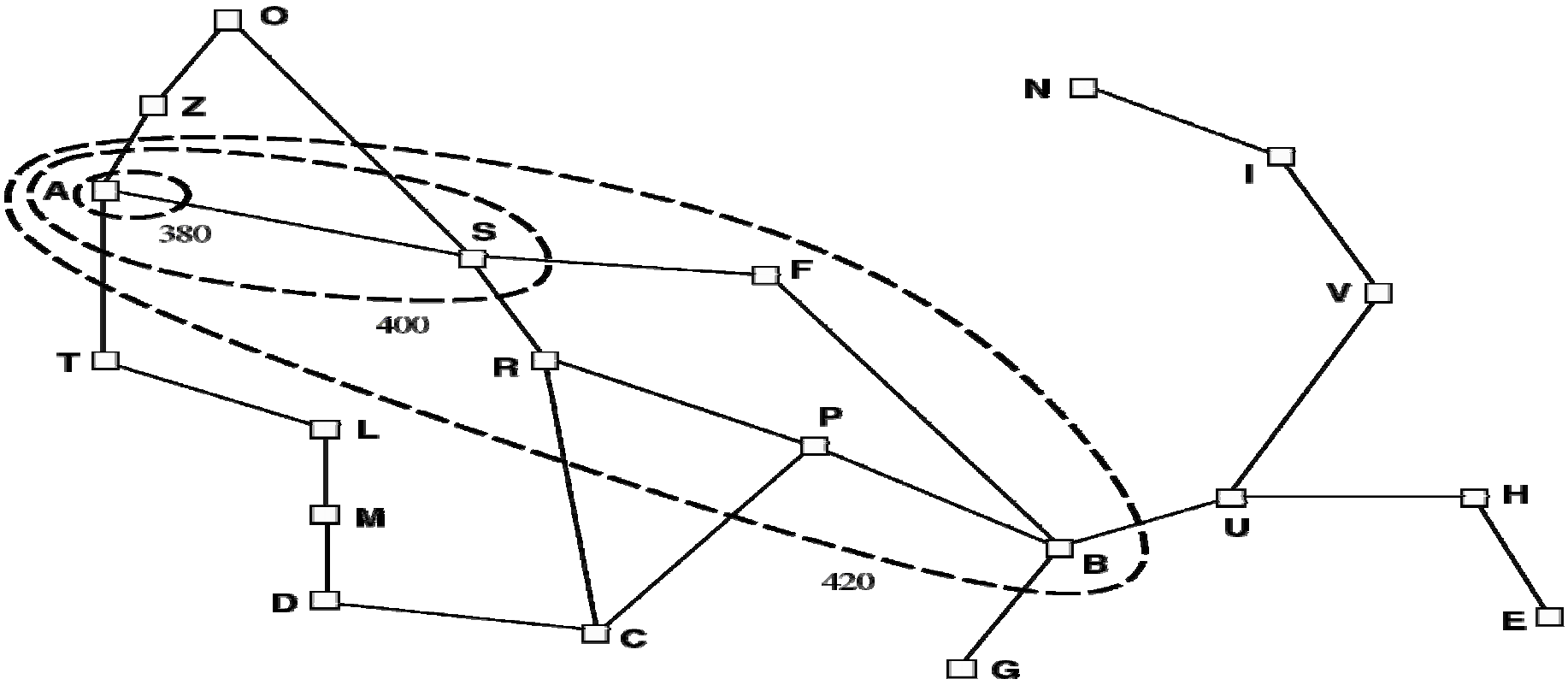
Lecture 9

- Heuristic search, continued

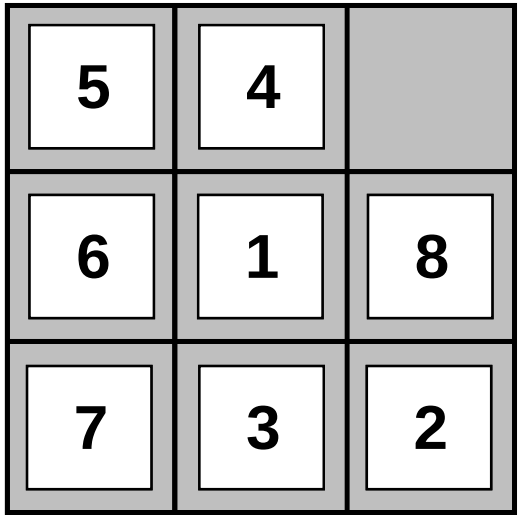
A* revisited

- Reminder:
  with A* we want to find the best-cost (***C)*** path to the goal **first.**
  - To do this, all we have to do is make sure our cost estimates are **less than** the actual costs.
  - Since our desired path has the lowest actual cost, no other path can be **fully** expanded first, since at some point it's estimated cost will have to be higher that the cost ***C.***

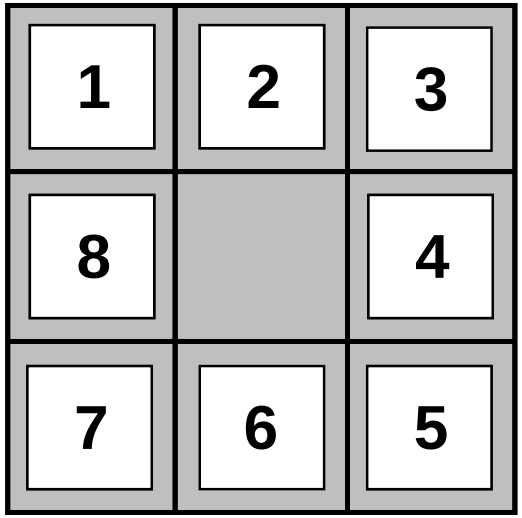


CS-424 Gregory Dudek

Heuristic Functions: Example



Start State



Goal State

8-puzzle

h_C = number of missplaced tiles

h_M = Manhattan distance

- Admissible?
- Which one should we use?

Choosing a good heuristic

$$h_C <= h_M <= h_{\text{opt}}$$

- Prefer h_M

Note: Expand all nodes with e(n) = g(n) + h(n) < e^*

- So, $g(n) < e^* - h(n)$, higher h means fewer n's.

- When one *h* funtion is larger than another, we say it is **more informed.**

- Aside: How would we get an h_{opt} ??

?

Inventing Heuristics

- Automatically

- A tile can move from sq A to sq B if
- A is adjacent to B and B is blank.
- (a) A tile can move from sq A to sq B if A is adjacent to B.
- (b) A tile can move from sq A to sq B if B is blank.
- (c) A tile can move from sq A to sq B.

- If all admissible, combine them by taking the max.

IDA*

- Problem with A*: like BFS, uses too much memory.

- Can combine with iterative deepening to limit depth of search.
- Only add nodes to search list L is they are within a limit on the *e cost*.

- iterative deepening (ID) + A* = IDA*

Further improvement: Simplified Memory-Bounded A* : SMA*

A Different Approach

- So far, we have considered methods that systematically explore the full search space, possibly using **principled** pruning (A* etc.).

- The current best such algorithms (IDA* / SMA*) can handle search spaces of up to 10^{100} states or around 500 binary valued variables.
  (These are ``ballpark " figures only!)

And if that's not enough?

- What if we have 10,000 or 100,000 variables / search spaces of up to $10^{30,000}$ states?

- A completely different kind of method is called for:

Local Search Methods

or

Iterative Improvement Methods

When?

Applicable when we're interested in the Goal State not in how to get there.

E.g. N-Queens, VLSI layout, or map coloring.

Iterative Improvement

- Simplest case:
  - Start with some (initial) state
  - Evaluate it's quality
  - Apply a perturbation ∂ to move to an adjacent state
  - Evaluate the new state
  - Accept the new state sometimes, based on decision $D(\partial)$

∂ can be random

D(.) can be always.

Or….

Can we be smarter?

- Can we do something smarter than
  A) moving randomly
  B) accepting every change?

A) …..
move in the “right” direction

B)
accept changes that get us towards a better solution

Hill-climbing search

- Often used when we have a solution/path/setting and we want to improve it, and we have a *suitable space:*
  an **iterative improvement algorithm.**

- Always move to a successor that that increases the desirability (can minimize or maximize cost).
  - Expand children of current node.
  - Sort them by expected distance to goal (closest first).
  - Put them all at the front of L.
- Depth first flavor.

Hill-climbing problems

- Hill climbing corresponds to function optimization by simply moving uphill along the gradient.
  - Why isn't this a reliable way of getting to a global maximum?

- 1. Local maxima
- 2. Plateaus
- 3. Ridges

  - E.g. Trajectory planning.
  - Scrabble.

Gradient ascent

- In continuous spaces, there are better ways to perform hill climbing.

  - Acceleration methods
    - Try to avoid sliding along a ridge
  - Conjugate gradient methods
    - Take steps that do not counteract one another.

Stochastic search

- Simply moving uphill (or downhill) isn't good enough.
  - Sometime, have to go against the obvious trend.

- Idea: **Randomly make a "non-intuitive" move.**
  - Key question: **How often**?

- One answer (simple intuition):
- Act more randomly when you are lost, less randomly when you're not.

Simulated Annealing

- Initially act more randomly.
- As time passes, we assume we are doing better and act less randomly.
- Analogy: associate each state with an energy or "goodness".
- Specifics:
  - Pick a random successor s^2 to the current state s^1.
  - Compute ∂E = energy(s^2) - energy(s^1).
  - If ∂E good (positive) go to the new state.
  - If ∂E bad (negative), still take it sometime¨, with probability $e^{(\partial E/T)}$
- **Annealing schedule specifies how to change T over time.**

GA's

Genetic Algorithms

Plastic transparencies & blackboard.

CS-424 Gregory Dudek

Adversary Search

Basic formalism

- 2 players.
- Each has to beat the other (competing objective functions).
- Complete information.
- Alternating moves for the 2 players.

Minimax

- Divide tree into *plies*
- Propagate information up from terminal nodes.
- Storage needs grow exponentially with depth.