

1 Administrivia

- We now have a class mail list – cs527@cs.unm.edu: I recommend that everyone subscribe to it.
- The first reading (for next Tuesday) is available via the class web page. It's Minsky's "Steps Toward Artificial Intelligence" from 1961. Note that this is about 10 years after Turing wrote his original paper on the possibilities of thinking machines.

2 Intro Search, cont'd

- Last week, we discussed elementary search methods – DFS and BFS
- We pointed out that these types of *uniformed* search are hideously inefficient and expand far too many nodes in the search graph, largely because they can't tell when they're going in the wrong direction
- We'll talk in a moment about how to guide the search more effectively, but for the moment let's consider a couple of other things you can do to improve efficiency locally.
- What was the strength and weakness of
 - BFS? (Finds the optimal path whenever arc costs are non-negative, but requires huge time and space)
 - DFS? (Conservative space, can find goal quickly if you're lucky, but can get lost on infinite paths and isn't guaranteed to find optimal path).
- One suggestion was to trade off between DFS and BFS – this leads to the notion of *iterative deepening* – do a complete DFS down to depth k . Then, if the goal isn't found, extend k by 1 and restart the search.
- This seems like it's inefficient – you're reexamining states many times. But it turns out not to be too bad. *Why?*
- Answer is that b/c search tree expands exponentially, almost all time is always spent in bottom ply.
- Specifically, if branching factor is B , and the search completes when it reaches depth d , then the B nodes at level 1 (children of the root) are generated d times, the B^2 nodes at level 2 are generated $d - 1$ times, etc. This leads to:

$$R(B, d) = (d)B + (d - 1)B^2 + (d - 2)B^3 + \dots (1)B^d = \sum_{k=1}^d (d - k + 1)B^k$$

- The solution to this recurrence, of course, is $O(B^d)$

- Thus, this method has the advantage of finding the solution in a depth-first fashion (requiring only $O(Bd)$ space rather than the $O(B^d)$ space needed by BFS). As with BFS, when the cost function is nondecreasing, this is guaranteed to find the minimal cost path.
- Another trick is *bidirectional search*: search simultaneously from the start state and goal state until some branches intersect.
- Advantage of this is massive speed improvement. Assuming that start and goal are separated by d ply and that the forward and backward searches meet exactly in the middle, how much time does this take?
- Answer: $O(B^{d/2} + B^{d/2}) = O(B^{d/2})$. Of course, you have to keep at least one of the search trees fully in memory to make sure that you know when you re-encounter it, so the space is $O(B^{d/2})$ as well.
- Big problem: how do you generate states backward? This is not always easy given only forward generative rules... E.g., in chess, your goal is “a winning board”. So you have a huge number of goals, and a huge number of predecessors (there’s more to it than running each move rule backward – why?)

3 Heuristic Search, A*

- **Reading:** Ch 4
- “Heuristic” from greek, “to find”.
- Roughly speaking, an imprecise guide to solution. A measure of one partial solution being closer to the goal than another.
- Formally, we simply assign a function $f : \mathcal{S} \rightarrow \mathbb{R}$ that indicates proximity to the solution.
- Use f to guide the search process and influence which nodes are examined next – want to head in the direction of *smallest* f
- We require only that that $f(t) = 0$ for goal t
- Other than that, we want $f(x) \leq f(y)$ to express that state x is closer to t (the goal state) than y is.
- Note that if this *actually* holds, in all cases, then f is an exact guide to the goal and we can just follow f and be assured of finding the goal with no side-tracking. We’ll call this the “true value function” f^*
- In practice, we don’t know the true value, so we use a rough approximation to it and we have to accept that we’ll be wrong sometimes.
- Example: 8-puzzle. One heuristic is number of tiles out of place.
- Q: is this a *good* heuristic? Why or why not?

- The heuristic gives us a guide/evaluation; the second thing we need is a procedure for exploiting that – a search procedure.
- We've already talked about the simplest: hill-climbing
- Slightly more sophisticated: so-called **best-first** search (something of a misnomer).
- C.f. slide on best-first-search
- Problem with this is that you don't account for how long it took you to find a particular node (more precisely, the cost to get to that node)
- You might end up finding the goal, but only by the longest possible path
- Small modification improves this dramatically
- Replace simple evaluation function f with

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

Where $g(x)$ measures the distance from the start to x and $h(x)$ estimates the distance from x to the goal.

- Clearly, $g(x)$ is easy to get ahold of; $h(x)$ is harder