CS 486 — Lecture 3: Heuristic Search

1 Why Heuristic Search

- A heuristic search algorithm uses heuristics to estimate how close a state is to a goal.
- We define the search heuristic, $h(n)$, to be an estimate of the cost of the cheapest path from node n to a goal node.
- $h(n)$ is some arbitrary, non-negative function that would be problem-specific.
- If n is the goal node, then $h(n) = 0$.
- We need $h(n)$ to be simple to compute — otherwise why use it?

2 Greedy BFS (Best-first search)

- The frontier is a priority queue ordered by $h(n)$, and expand the node with the lowest heuristic cost.
- Note that GBFS will not necessarily find a solution/terminate!
- Nor does it necessarily find the *optimal* path!

3 A*

- The frontier is a priority queue ordered by $cost(n) + h(n)$.
- Expand the node with the lowest combined cost.
- Combines the ideas of LCFS and GBFS.
- One of the issues of heuristic functions are that it's really hard to determine things like runtimes.
- We state that if $h(n)$ is admissible — it will never overestimate the cost of the cheapest path from node n to the goal node — then the solution found by A* will be optimal.
- In other words, it is admissible if the heuristic's cost is always bounded by the *actual* optimal cost.
- Furthermore, we state that A* is *optimally* efficient.

4 Search Heuristic Design

- For something like the 8-puzzle, both a Manhattan Distance and a Misplaced Tile (number of tiles that are not in their goal positions) are admissible heuristic functions.
- First, define a relaxed problem by simplifying or removing constraints on the original problem.
- Then, solve the relaxed problem without search.
- Finally, the cost of the optimal solution to the relaxed problem is an admissible heuristic for the original problem.
- Consider the 8-puzzle:
  - A tile can move from square A to B if A and B are adjacent, and if B is empty.
  - We can define both of the previous heuristics based on what constraints we relax!

- But which heuristic might be better?
- We actually want a heuristic with a *higher* value — this means it's closer to h^*, the true optimal solution for the constrained problem.
- We can define a heuristic $h_1(n)$ as a "dominating heuristic" compared to $h_2(n)$ if $(\forall n(h_1(n) \geq h_2(n)))$ and $\exists n(h_1(n > h_2(n)))$.
- **Theorem**: if $h_1(n)$ dominates $h_2(n)$, A^* using h_1 will never expand more nodes than A^* using h_2.

5 Pruning

- Cycle checking/pruning is required to prevent getting stuck in a cycle.
- We have to ensure the cycle is not part of the least-cost path.
- For something like DFS (that only checks 1 path at a time), our cycle checking can be done in constant time at the cost of space — store whether a node is on the current path we're exploring; then we can quickly check!
- For something that tracks many paths, then when we generate a neighbour, we can check if the neighbour already appears in the current path. This would mean the check is linear in the length of the path.
- But what about multiple-path pruning? We don't want to repeat paths to the same node if we found a better path.
- Cycle checking is actually just a case of multi-path prune!
- This requires us storing all nodes we have found paths to.
- Is it possible to find accidentally prune the optimal solution? That's something we have to be careful of!
- For A^*, it *is* possible, unfortunately, if we use it, and thus would no longer be optimal!
- One thing we can do to deal with it is remove all paths from the frontier that use the longer path.
- Another option is to change the initial segment of the paths on the frontier to use the shorter path.
- Or, you could use a searching algorithm that will *guarentee* that the least-cost path to a node is found in the first place (ie: LCFS).
- Can we do the last option for A^*?
- Remember, we *do* control the heuristic — the admissible definition requires that our path to the goal node is bounded.
- We constrain further — now, we state that the path to *any* node must be bounded.
- This is called the monotone restriction: for any directed arc from m to n, then $h(m) - h(n) \leq cost(m, n)$.
- If this holds, then we say the heuristic is consistent.
- This is a strictly stronger requriement to admissibility (as the arc from s to g is included). This also ensures that A^* with multi-path pruning is optimal.