CS 4700: Foundations of Artificial Intelligence

Fall 2017

Instructor: Prof. Haym Hirsh

Lecture 7

Extra Credit Opportunity: Lecture Today 4:15pm Gates G01

Learning to See Without a Teacher

Phillip Isola
UC Berkeley

Over the past decade, learning-based methods have driven rapid progress in computer vision. However, most such methods still require a human "teacher" in the loop. Humans provide labeled examples of target behavior, and also define the objective that the learner tries to satisfy. The way learning plays out in nature is rather different: ecological scenarios involve huge quantities of unlabeled data and only a few supervised lessons provided by a teacher (e.g., a parent). I will present two directions toward computer vision algorithms that learn more like ecological agents. The first involves learning from unlabeled data. I will show how objects and semantics can emerge as a natural consequence of predicting raw data, rather than labels. The second is an approach to data prediction where we not only learn to make predictions, but also learn the objective function that scores the predictions. In effect, the algorithm learns not just how to solve a problem, but also what exactly needs to be solved in order to generate realistic outputs. Finally, I will talk about my ongoing efforts toward sensorimotor systems that not only learn from provided data but also act to sample more data on their own.

Homework 1

- Available on course website
- Due: Tuesday, February 28, at the start of class, 8:40am
- Submission through Gradescope, instructions to be given

Today

- Informed Search (R&N Ch 3)
- 4701 – last 15 minutes

Thursday, February 23

- Adversarial Search (T&N Ch 3)

Special Cases of Best-First Search

- $g(s)$: Cost of going from initial state to s
- $h(s)$: Estimate of cost of going from s to a goal state

- $f(s) = g(s)$: Uniform Cost Search
- $f(s) = h(s)$: Greedy Best-First Search
- $f(s) = g(s) + h(s)$: A* Search

Special Cases of Best-First Search

- $g(s)$: Cost of going from initial state to s
- $h(s)$: Estimate of cost of going from s to a goal state

- $f(s) = g(s)$: Uniform Cost Search
- $f(s) = h(s)$: Greedy Best-First Search
- $f(s) = g(s) + h(s)$: A* Search

Special Cases of Best-First Search

- $g(s)$: Cost of going from initial state to s
- $h(s)$: Estimate of cost of going from s to a goal state

- $f(s) = g(s)$: Uniform Cost Search
- $f(s) = h(s)$: Greedy Best-First Search
- $f(s) = g(s) + h(s)$: A* Search

A* Search

- $f(s) = g(s) + h(s)$ is an *estimate* of the cost of a solution that goes through s

Notation:

- $h^*(s)$: The true (unknown) cost of going from s to the cheapest solution
- $f^*(s) = g(s) + h^*(s)$: The true (unknown) cost of the cheapest solution that goes through s

A* Search

Admissible heuristic evaluation functions h(s)

- h(s) is *admissible* if for all s $0 \leq h(s) \leq h^*(s)$
  - In other words, h never overestimates h*

Examples:

- Route finding:
  - Straight-line distance between s and goal
- 8-puzzle (and 15-puzzle and all tile-sliding puzzles like it):
  - Number of out of place tiles
  - Sum of how many spaces each tile is from where it should be

A* Search

If

- the branching factor b of a problem is finite,
- the costs of all operators are positive, and
- h(s) is admissible

Then

- A* search is guaranteed to find an optimal solution
- (Sometimes worded as “A* is admissible”)

A* Search

Consistent heuristic evaluation functions h(s)

- h(s) is *consistent* if for all s, $h(s) \leq cost(s,s') + h(s')$ for all successors s'
  - A form of the triangle inequality (the sum of the lengths of two sides of a triangle are greater than the length of the third side)
  - Does not refer to h*

If h is consistent then h is admissible, hence:

A* Search

If

- the branching factor b of a problem is finite,
- the costs of all operators are positive, and
- h(s) is admissible

Then

- A* search is guaranteed to find an optimal solution
- (Sometimes worded as “A* is admissible”)

A* Search

If

- the branching factor b of a problem is finite,
- the costs of all operators are positive, and
- h(s) is admissible

Then

- A* search is guaranteed to find an optimal solution
- (Sometimes worded as “A* is admissible”)

A* Search

If

- the branching factor b of a problem is finite,
- the costs of all operators are positive, and
- h(s) is consistent

Then

- A* search is guaranteed to find an optimal solution

A* Search

Consider any other search method that is given the same information as A* (initial state, goal, operators, and h) and is guaranteed to return an optimal solution:

A* is *optimally efficient:*

- No other search method is guaranteed to expand fewer nodes than A*
- (Ignoring the effects of randomly breaking ties between states with the same f(s) value)

A* Search

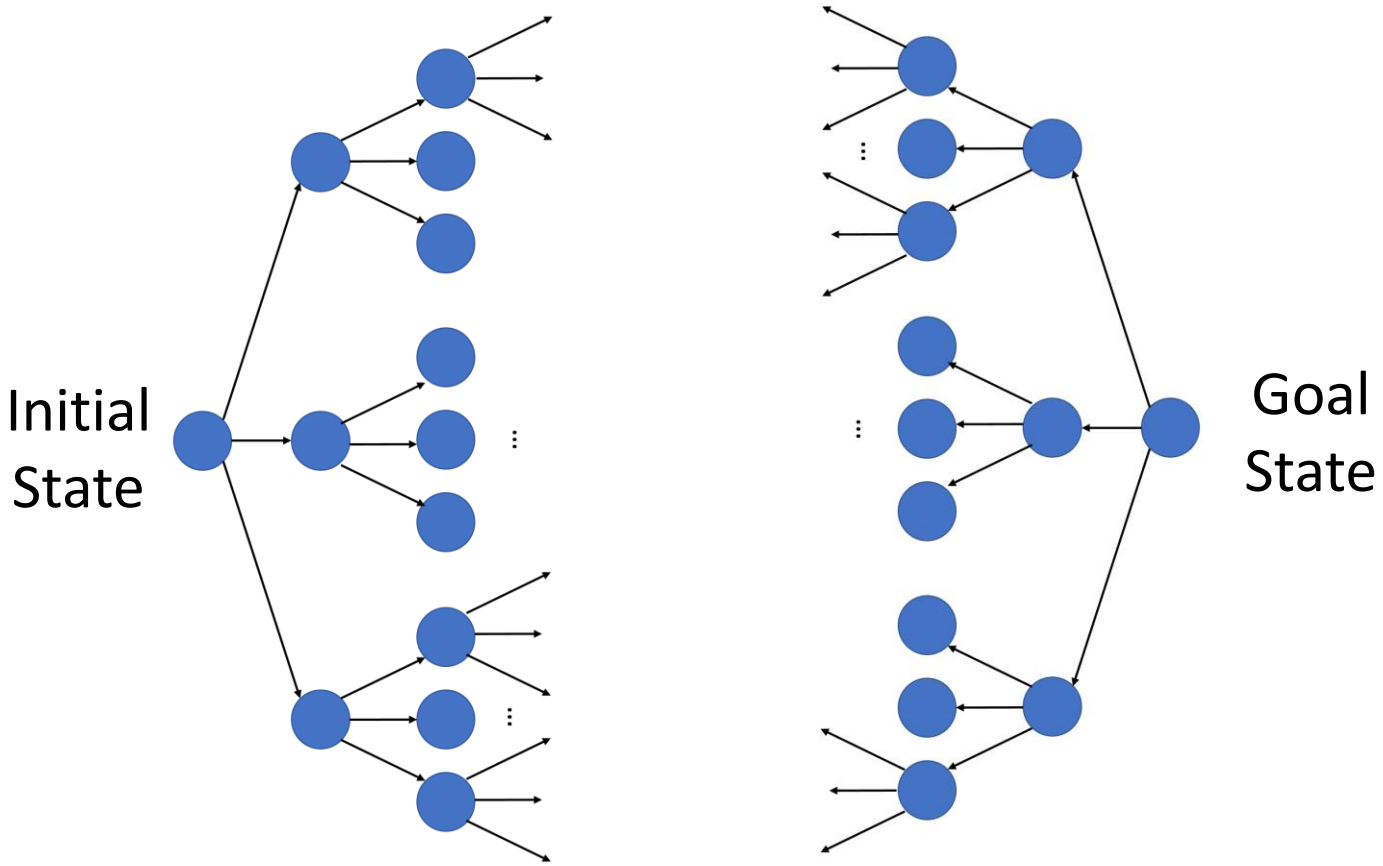
A* was introduced in 1966, has been studied extensively, and there are many other results. For example:

- Let Δ = h*(initialstate) – h(initialstate). If the search space is a tree with reversible actions, the runtime of A* is $O(b^{\Delta})$.

Other Search Methods

Other Search Methods
Uninformed Search

Bidirectional Search



Bidirectional Search

Perform a breadth-first search from both ends

- Interleave searches:
  - Expand 1 node of tree coming from initial state
  - Expand 1 node of tree coming from goal state
- At each node instead of testing goal(s) test whether s matches any node in the other search's open or closed lists

Analysis: Bidirectional Search

Criterion	DFS	BFS	Bidirectional Search
Complete?	No	Yes	Yes
Optimal?	No	Yes	Yes
Time?	∞	$O(b^d)$	$O(b^{\frac{d}{2}})$
Space?	$O(bd)$	$O(b^d)$	$O(b^{\frac{d}{2}})$

Other Search Methods
Informed Search

Branch-and-Bound Search

- Use with depth-first search on problems with f(s) = g(s) and thathave multiple solutions

- Keep track of the cheapest solution found thus far
- Don’t expand states whose value exceed this cheapest-thus-far value

Iterative Deepening Version of Best-First Search

- Modifies best-first search to have a cost bound:
  - Do not expand a state whose f value is greater than the given bound
  - Keep track of the lowest f value among all the states that exceeded the given bound
  - Use that value as the cost bound for the next iteration of the cost-bounded search

- When f(s) = g(s) this is called Iterative Lengthening Search
  - If costs of operators are all 1, this is Iterative Deepening Search
- When f(s) = g(s) + h(s) this is called Iterative Deepening A* (IDA*)

Simplified Memory-bounded A* (SMA*)

- Run A* as usual until memory is exhausted – no room to add a new state to the open list
- When memory is exhausted, delete from open the most expensive pending state, but save it’s value with its parent state
- When other parts of the space have been expanded the parent state can be expanded to find the state with that value

Hill-climbing Search

- Also known as greedy local search
- ("Hill *climbing*": Originally developed for problems where larger f is better")

```
hillclimbing(s,ops):
    If goal(s) then return s
    Else succs ← {s' | s' = apply(s,o) where o ∈ ops};
        best ←  argmin   f(s') ;
                s' ∈ succs
        hillclimbing(best)
```

Hill-climbing Search

- 8-queens puzzle
  - Initial state: 8 queens on board, 1 per column, all in row 1
  - Operators: Move queen i to a different row in its column
  - f(s) = # of pairs of queens that are attacking each other

Hill-climbing Search

- *Not* complete
- Main problem: Local optima
  - All of the states you can reach from the given state are worse than it, so you're at a "peak", but there are better states elsewhere in the search space
- Common solution approach: Random restart
  - Multiple possible initial states (example: 8-queens)
  - Select successors probabilistically based on f value

Simulated Annealing

- Similar to hillclimbing – keeps a single state

```
SA(s,ops):
    If goal(s) then return s
    Else pick a random o in ops; s' ← apply(o,s)
        If f(s') < f(s) then SA(s',ops)
        Else with some small probability keep s' anyway,
            otherwise pick another o
```