Review of Search & Introducing Adversarial Search

16 September 2020
Programming assignment 1: Search
Out now!
Expect 4–5 programming assignments for the semester (and a final project/paper).

I want you to have enough time to finish them, not be rushed, so, *at least for now*: No due date.

But, beware of falling behind! Try to finish Assignment 1 within two weeks.
Collaboration encouraged!

You can work individually or in pairs.
A heuristic for cheating:

Anything that helps you learn is good.
Anything that helps you avoid having to learn is bad.
Recap: Search
Search problem

- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test
Search tree

Nodes: represent plans for reaching states
Plans have costs (sum of action costs)
A search algorithm systematically builds a search tree.

Ignoring cost, we can explore the search tree

- **breadth-first**, preferring the shortest paths (number of actions)
- **depth-first**, preferring to explore all the ways of continuing a choice before trying another
- **iterative deepening**: repeat depth-first search up to a depth limit and keep increasing the limit, achieving the best of breadth-first (find the closest goal) and depth-first (low memory)
These search algorithms are the same except for frontier strategies.

Conceptually, all fringes are *priority queues* (i.e., collections of nodes with attached priorities).

(DFS and BFS are a bit faster using simple stacks and queues instead of priority queues.)
Search algorithms can be:

*Complete*: finds a plan if one exists

*Cost-optimal*: finds least-cost plans
Uniform-cost search

*Strategy*: Explore increasing cost contours

*The good*: Complete and optimal!
SCORE: 0
Uniform-cost search

**Strategy**: Explore increasing cost contours

**The good**: Complete and optimal!

**The bad**:

- Explores options in every “direction”
- No information about goal location
Informed search
A search *heuristic* is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing
A search *heuristic* is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing
A search **heuristic** is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing
A search *heuristic* is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing
Example: Heuristic function

$h(x)$
Greedy best-first search
Example: Heuristic function

\[ h(x) \]
Greedy best-first search expands the node that seems closest.
Greedy best-first search expands the node that seems closest.
Greedy best-first search expands the node that seems closest.
Greedy best-first search expands the node that seems closest.
Greedy best-first search expands the node that seems closest.
Greedy best-first search expands the node that seems closest.

What can go wrong?
Greedy best-first search expands the node that seems closest.

What can go wrong?

Greedy: \[140 + 99 + 211 = 450\]

Better: \[140 + 80 + 97 + 101 = 418\]
Greedy best-first search

**Strategy:** expand a node that you think is closest to a goal state

**Heuristic:** estimate of distance to nearest goal for each state

**A common case:** Best-first takes you straight to the (wrong) goal
Greedy best-first search

**Strategy**: expand a node that you think is closest to a goal state

**Heuristic**: estimate of distance to nearest goal for each state

**A common case**: Best-first takes you straight to the (wrong) goal

**Worst-case**: like a badly-guided DFS
SCORE: 0
A* search
A* search
A* search
A* search

UCS
A* search

UCS

Greedy
A* search

UCS

Greedy

A*
Is $A^*$ optimal?

![Diagram with nodes S, A, and G with distances and heuristics](image)

- $S$ to $A$: 1, $h = 7$
- $A$ to $G$: 3, $h = 6$
- $G$ to $S$: 5, $h = 0$

For $A^*$ to be optimal, $h$ must not overestimate the true cost of reaching the goal.
Is A* optimal?

A

G

S

h = 6

h = 7

h = 0

1

3

5

1

3

5

h

f

g

h

f
Is A* optimal?

<table>
<thead>
<tr>
<th></th>
<th>g</th>
<th>h</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Is $A^*$ optimal?

```
<table>
<thead>
<tr>
<th>g</th>
<th>h</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
```

Diagram:
- Node $S$ with $h = 7$
- Node $A$ with $h = 6$
- Node $G$ with $h = 0$
- Edge $S ightarrow A$ with weight 1, $h = 7$
- Edge $A ightarrow G$ with weight 3, $h = 0$
- Edge $G ightarrow S$ with weight 5
Is A* optimal?
Is A* optimal?

```
\begin{align*}
\text{S} & \quad h = 7 \\
 & \quad \quad 1 \quad \quad \quad A \\
A & \quad h = 6 \\
 & \quad \quad 3 \quad \quad \quad \text{G} \\
& \quad \quad \quad \quad \quad 5 \\
\text{G} & \quad h = 0 \\
& \quad \quad \text{S} \\
\end{align*}
```

\[ f_s = 7, \quad f_A = 7, \quad f_G = 5 \]

\[ g_s = 0, \quad g_A = 1, \quad g_G = 5 \]
Is A* optimal?

What went wrong?

<table>
<thead>
<tr>
<th></th>
<th>g</th>
<th>h</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>S→A</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>S→G</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
Is A* optimal?

What went wrong?

Actual bad goal cost < estimated good goal cost
Is $A^*$ optimal?

What went wrong?

Actual bad goal cost < estimated good goal cost

We need estimates to be less than actual costs! (admissibility)
Admissible heuristics
Inadmissible (pessimistic) heuristics sometimes overestimate remaining cost. They break optimality by trapping good plans on the frontier.

Admissible (optimistic) heuristics never overestimate the remaining cost; they’re a lower bound. They slow down bad plans but never outweigh true costs.
A heuristic $h$ is **admissible** (optimistic) if

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.

Example:

Coming up with admissible heuristics is most of what’s involved in using A* in practice.
A heuristic $h$ is \textit{admissible} (optimistic) if

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.

Example:

Coming up with admissible heuristics is most of what’s involved in using $A^*$ in practice.
A* with an uninformatve heuristic, e.g., one that always returns 0, is equivalent to:

a. breadth-first search  
b. depth-first search  
c. uniform-cost search  
d. greedy search  
e. none of these
Properties of A*
Properties of A*
UCS vs A* contours

Uniform-cost expands equally in all "directions".

A* expands mainly toward the goal, but does hedge its bets to ensure optimality.
A* applications

Video games
Pathing / routing problems
Resource planning problems
Robot motion planning
Language analysis
Machine translation
Speech recognition

…
Creating admissible heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available.
Creating admissible heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to relaxed problems, where new actions are available.
Creating admissible heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to relaxed problems, where new actions are available.

Inadmissible heuristics are often useful too.
Adversarial search
Checkers

1950  First computer player.
1994  First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame.
2007  Checkers solved!
Chess

1997 Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
Go

Branching factor \( b > 300 \). Classic programs use pattern knowledge bases.

2016  Alpha GO defeats human champion using Monte Carlo (randomized) Tree Search and a learned evaluation function.
Pac-Man
Behavior from computation
Video: Mystery Pac-Man
Video: Mystery Pac-Man
Adversarial games
Types of games

Many different kinds of games!

Axes:

- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a *strategy* (policy) which recommends a move from each state
Zero-sum games

Agents have opposite utilities (values on outcomes)

Lets us think of a single value that one maximizes and the other minimizes

Adversarial, pure competition

General games

Agents have independent utilities (values on outcomes)

Cooperation, indifference, competition, and more are all possible

More later on non-zero-sum games
Deterministic games

Many possible formalizations, one is:

**States**: $S$ (start at $s_0$)

**Players**: $P = \{1\ldots N\}$ (usually take turns)

**Actions**: $A$ (may depend on player / state)

**Transition function**: $S \times A \rightarrow S$

**Terminal test**: $S \rightarrow \{\text{True, False}\}$

**Terminal utilities**: $S \times P \rightarrow R$

Solution for a player is a **policy**: $S \rightarrow A$
Adversarial search
Single-agent trees
Single-agent trees
Single-agent trees
Single-agent trees
Single-agent trees
Single-agent trees
Value of a state
Value of a state

The best achievable outcome (utility) from that state
Value of a state

Value of a state: The best achievable outcome (utility) from that state

Terminal states: $V(s) = \text{known}$
Value of a state

**Value of a state:**
The best achievable outcome (utility) from that state

**Non-terminal states:**
\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]

**Terminal states:**
\[ V(s) = \text{known} \]
Acknowledgments

The lecture incorporates material from:

Pieter Abbeel, Dan Klein, et al., University of California, Berkeley;
ai.berkeley.edu