CS 5522: Artificial Intelligence II

Informed Search

Instructor: Alan Ritter
Ohio State University

[These slides were adapted from CS188 Intro to AI at UC Berkeley. All materials available at http://ai.berkeley.edu.]
Today

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search

- Graph Search
Recap: Search
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)

- **Search algorithm:**
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans
Example: Pancake Problem

Cost: Number of pancakes flipped
Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation $\sigma$ of the integers from 1 to $n$, let $f(\sigma)$ be the smallest number of prefix reversals that will transform $\sigma$ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all $\sigma$ in (the symmetric group) $S_n$. We show that $f(n) \leq (5n + 5)/3$, and that $f(n) \geq 17n/16$ for $n$ a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$. 
Example: Pancake Problem

State space graph with costs as weights
**General Tree Search**

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- **Action:** flip top two
  - **Cost:** 2

- **Path to reach goal:** Flip four, flip three
  - **Total cost:** 7
All these search algorithms are the same except for fringe strategies

- Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
- Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
- Can even code one implementation that takes a variable queuing object
Uniform Cost Search

- Strategy: expand lowest path cost

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

[Demo: contours UCS empty (L3D1)]
[Demo: contours UCS pacman small maze]
Video of Demo Contours UCS Empty
Video of Demo Contours UCS Pacman Small Maze

SCORE: 0
Informed Search
Search Heuristics

- A 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

\[ h(x) \]
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place
Greedy Search
Example: Heuristic Function

$h(x)$

Straight-line distance to Bucharest:

- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobroța: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hirsova: 151
- Iași: 226
- Lugoj: 244
- Mehadia: 241
- Neamț: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vâlcea: 193
- Sibiu: 253
- Timișoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
Greedy Search

- Expand the node that seems closest...

- What can go wrong?
Greedy 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

[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]
Video of Demo Contours Greedy (Empty)
Video of Demo Contours Greedy (Pacman Small Maze)
A* Search
A* Search
Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* $g(n)$
- **Greedy** orders by goal proximity, or *forward cost* $h(n)$

- **A* Search** orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager
When should A* terminate?

- Should we stop when we enqueue a goal?
  - No: only stop when we dequeue a goal
Is A* Optimal?

- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!
Admissible Heuristics
Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe.

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs.
Admissible Heuristics

- 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.

- Examples:

- Coming up with admissible heuristics is most of what’s involved in using $A^*$ in practice.
Optimality of A* Tree Search
Optimality of A* Tree Search

Assume:
- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:
- A will exit the fringe before B
Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor $n$ of A is on the fringe, too (maybe A!)
- Claim: $n$ will be expanded before B
  1. $f(n)$ is less or equal to $f(A)$

\[
\begin{align*}
  f(n) &= g(n) + h(n) & \text{Definition of f-cost} \\
  f(n) &\leq g(A) & \text{Admissibility of h} \\
  g(A) &= f(A) & h = 0 \text{ at a goal}
\end{align*}
\]
Optimality of A* Tree Search: Blocking

Proof:
- Imagine B is on the fringe
- Some ancestor $n$ of A is on the fringe, too (maybe A!)
- Claim: $n$ will be expanded before B
  1. $f(n)$ is less or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$

$$g(A) < g(B) \quad B \text{ is suboptimal}$$
$$f(A) < f(B) \quad h = 0 \text{ at a goal}$$
Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor $n$ of A is on the fringe, too (maybe A!)
- Claim: $n$ will be expanded before B
  1. $f(n)$ is less or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. $n$ expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

$f(n) \leq f(A) < f(B)$
Properties of A*
Properties of A*

Uniform-Cost

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

[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]
Video of Demo Contours (Empty) -- UCS
Video of Demo Contours (Empty) -- Greedy
Video of Demo Contours (Empty) - A*
Video of Demo Contours (Pacman Small Maze) - A*
Comparison

Greedy  Uniform Cost  A*
A* Applications
A* Applications

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

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]
Video of Demo Pacman (Tiny Maze) - UCS / A*
Video of Demo Empty Water Shallow/Deep - Guess Algorithm
Creating Heuristics

YOU GOT

HEURISTIC UPGRADE!
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.
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?
8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- \( h(\text{start}) = 8 \)
- This is a *relaxed-problem* heuristic

Start State

<table>
<thead>
<tr>
<th>7</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Goal State

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Statistics from Andrew Moore

<table>
<thead>
<tr>
<th>Average nodes expanded when the optimal path has...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...4 steps</td>
</tr>
<tr>
<td>UCS</td>
</tr>
<tr>
<td>TILES</td>
</tr>
</tbody>
</table>
What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

- Total Manhattan distance

- Why is it admissible?

- \( h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \)

<table>
<thead>
<tr>
<th>Average nodes expanded when the optimal path has...</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>
How about using the *actual cost* as a heuristic?
- Would it be admissible?
- Would we save on nodes expanded?
- What’s wrong with it?

With A*: a trade-off between quality of estimate and work per node
- As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself
Semi-Lattice of Heuristics
Trivial Heuristics, Dominance

- **Dominance:** \( h_a \geq h_c \) if
  \[
  \forall n : h_a(n) \geq h_c(n)
  \]

- **Heuristics form a semi-lattice:**
  - Max of admissible heuristics is admissible
    \[
    h(n) = \max(h_a(n), h_b(n))
    \]

- **Trivial heuristics**
  - Bottom of lattice is the zero heuristic
    (what does this give us?)
  - Top of lattice is the exact heuristic
Graph Search
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.
In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)
Graph Search

- **Idea:** never expand a state twice

- **How to implement:**
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but...
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set

- **Important:** store the closed set as a set, not a list

- **Can graph search wreck completeness?** Why/why not?

- **How about optimality?**
A* Graph Search Gone Wrong?

State space graph

Search tree

S (0+2)
A (1+4)    B (1+1)
  ↓        ↓
C (2+1)    C (3+1)
  ↓        ↓
G (5+0)    G (6+0)
Consistency of Heuristics

Main idea: estimated heuristic costs ≤ actual costs
- Admissibility: heuristic cost ≤ actual cost to goal
  \[ h(A) \leq \text{actual cost from A to G} \]
- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  \[ h(A) - h(C) \leq \text{cost(A to C)} \]

Consequences of consistency:
- The f value along a path never decreases
  \[ h(A) \leq \text{cost(A to C)} + h(C) \]
- A* graph search is optimal
Optimality of A* Graph Search
Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal
Optimality

- **Tree search:**
  - $A^*$ is optimal if heuristic is admissible
  - UCS is a special case ($h = 0$)

- **Graph search:**
  - $A^*$ optimal if heuristic is consistent
  - UCS optimal ($h = 0$ is consistent)

- Consistency implies admissibility

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems
A*: Summary
A*: Summary

- A* uses both backward costs and (estimates of) forward costs

- A* is optimal with admissible / consistent heuristics

- Heuristic design is key: often use relaxed problems
function Tree-Search(problem, fringe) return a solution, or failure

fringe ← Insert(make-node(initial-state[problem]), fringe)

loop do
  if fringe is empty then return failure
  node ← Remove-Front(fringe)
  if Goal-Test(problem, state[node]) then return node
  for child-node in Expand(state[node], problem) do
    fringe ← Insert(child-node, fringe)
  end
end
function \textsc{Graph-Search}(\textit{problem}, \textit{fringe}) return a solution, or failure
    \textit{closed} \leftarrow \text{an empty set}
    \textit{fringe} \leftarrow \text{INSERT(MAKE-NODE(INITIAL-STATE[\textit{problem}]), \textit{fringe})}
    \textbf{loop do}
        \textbf{if} \textit{fringe} is empty \textbf{then} return failure
        \textit{node} \leftarrow \text{REMOVE-FRONT(\textit{fringe})}
        \textbf{if} \text{GOAL-TEST(\textit{problem}, STATE[\textit{node}])} \textbf{then} return \textit{node}
        \textbf{if} STATE[\textit{node}] is not in \textit{closed} \textbf{then}
            add STATE[\textit{node}] to \textit{closed}
            \textbf{for} \textit{child-node} in \text{EXPAND(STATE[\textit{node}], \textit{problem})} \textbf{do}
                \textit{fringe} \leftarrow \text{INSERT(\textit{child-node}, \textit{fringe})}
            end
    end