Lecture 5
Heuristic Search and A*
(Ch: 3.6, 3.6.1)
Announcements

• Assignment 1 posted last Friday in Connect
  • Due Friday, Jan 27, 7pm
  • Please carefully read and follow the instructions on cover sheet

• If you have not done them yet, do all the “Graph Searching exercises” available at http://www.aispace.org/exercises.shtml
  • Suggestion: look at solutions only after you have tried hard to solve them!
Lecture Overview

Recap of Lecture 4

- Best First Search
- A*
  - Admissible heuristic
  - Completeness and Optimality
- Branch and Bound (time permitting)
Search Strategies are different with respect to how they:

A. Check what node on a path is the goal

B. Initialize the frontier

C. Add/remove paths from the frontier

D. Check if a state is a goal
Search Strategies are different with respect to how they:

A. Check what node on a path is the goal

B. Initialize the frontier

C. Add/remove paths from the frontier

D. Check if a state is a goal
Depth-First Search, DFS

• explores each path on the frontier until its end (or until a goal is found) before considering any other path.
• the frontier is a last-in-first-out stack
Breadth-first search (BFS)

- BFS explores all paths of length $l$ on the frontier, before looking at path of length $l + 1$
- The frontier is a first-in-first-out queue
### DFS vs. BFS

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>NO</td>
<td>NO</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>BFS</td>
<td>YES</td>
<td>YES</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>

**Key Idea:** re-compute elements of the frontier rather than saving them.
Iterative Deepening DFS (IDS) in a Nutshell

- Use DFS to look for solutions at depth 1, then 2, then 3, etc
  - For depth D, ignore any paths with longer length
  - Depth-bounded depth-first search

If no goal re-start from scratch and get to depth 2

If no goal re-start from scratch and get to depth 3

If no goal re-start from scratch and get to depth 4
Analysis of Iterative Deepening DFS (IDS)

- Time complexity: we showed that it is still $O(b^m)$, with limited overhead compared to BSF
- Space complexity: it does DFS

Complete?

Optimal?
### DFS vs. BFS

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>NO</td>
<td>NO</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>BFS</td>
<td>YES</td>
<td>YES</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>YES</td>
<td>YES</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
</tbody>
</table>

But what if we have costs associated with search actions (arcs in the search space?)
Search with Costs

Def.: The **cost of a path** is the sum of the **costs of its arcs**

\[
cost(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)
\]

In this setting we usually want to find the solution that **minimizes cost**

Def.: A search algorithm is **optimal** if when it finds a solution, it is the **best one**: it has the lowest path cost
Lowest-Cost-First Search (LCFS)

- **Lowest-cost-first search** finds the path with the *lowest cost* to a goal node.
- At each stage, it selects the path with the *lowest cost* on the frontier.
- The frontier is implemented as a priority queue ordered by path cost.

```
Algorithm Selected: Lowest Cost First
CURRENT PATH:
UBC -> KD -> MP -> KB
Path to last Goal Node: UBC -> JB -> KB -> DT -> SP (Goal) Cost: 11.0
Nodes expanded: 12
NEW FRONTIER:
Node: BBY Path Cost: 11.0
Path: UBC -> KD -> MP -> BBY

Node: KB Path Cost: 11.0
Path: UBC -> JB -> KB

Node: AP Path Cost: 12.0
Path: UBC -> JB -> KB -> DT

Node: BBY Path Cost: 13.0
Path: UBC -> JB -> KB

Node: BBY Path Cost: 13.0
Path: UBC -> JB -> KB

Node: BBY Path Cost: 13.0
Path: UBC -> JB -> KB

Node: BBY Path Cost: 13.0
Path: UBC -> JB -> KB

Node: BBY Path Cost: 13.0
Path: UBC -> JB -> KB

Node: SRY Path Cost: 30.0
Path: UBC -> KD -> MP -> RM -> SRY
```
Analysis of Lowest-Cost Search

- **Time complexity:** if the maximum path length is $m$ and the maximum branching factor is $b$
  - The time complexity is $O(b^m)$
  - In worst case, must examine every node in the tree because it generates all paths from the start that cost less than the cost of the solution
Analysis of LCFS

• What is the space complexity, if the maximum path length is \( m \) and the maximum branching factor is \( b \)?

A. \( O(b^m) \)
B. \( O(m^b) \)
C. \( O(bm) \)
D. \( O(b+m) \)
Analysis of Lowest-Cost Search

• Space complexity
  • Space complexity is $O(b^m)$:
  • E.g. uniform cost: just like BFS, in worst case frontier has to store all nodes that are $m-1$ steps away from the start node
## Summary of Uninformed Search

<table>
<thead>
<tr>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y (shortest)</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>Y</td>
<td>Y (shortest)</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>LCFS</td>
<td>Y (Costs &gt; $\varepsilon &gt; 0$)</td>
<td>Y (Least Cost)</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>
Lecture Overview

- Recap of Lecture 4

Heuristic search

- Best First Search
- A*  
  ✓ Admissible heuristic
  ✓ Completeness and Optimality
- Branch and Bound
Summary of Uninformed Search (cont.)

• Why are all the search strategies seen so far are called uninformed?
  • Because they do not consider any information about the states and the goals to decide which path to expand first on the frontier
    • They are blind to the goal

• In other words, they are general and do not take into account the specific nature of the problem.
Blind search algorithms do not take into account the goal until they are at a goal node.

Often there is extra knowledge that can be used to guide the search:
- an estimate of the distance/cost from node $n$ to a goal node.

This estimate is called a search heuristic.
More formally

Def.: A search heuristic $h(n)$ is an estimate of the cost of the optimal (cheapest) path from node $n$ to a goal node.

- $h$ can be extended to paths: $h(⟨n_0,...,n_k⟩) = h(n_k)$
- $h(n)$ should leverage readily obtainable information (easy to compute) about a node.
Example: finding routes

- What could we use as $h(n)$?
Example: finding routes

- What could we use as $h(n)$? E.g., the straight-line (Euclidian) distance between source and goal node.
Example 2

Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move up, down, left, right from tile to tile

Cost: number of moves

Possible h(n)?
Example 2

Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move up, down, left, right from tile to tile

Cost: number of moves

Possible h(n)? Manhattan distance ($L_1$ distance) between two points $(x_1, y_1), (x_2, y_2)$:

- sum of the (absolute) difference of their coordinates
  
  $|x_2 - x_1| + |y_2 - y_1|$
Lecture Overview

• Recap of Lecture 4
• Heuristic search
  • Best First Search
  • A*
    ✓ Admissible heuristic
    ✓ Completeness and Optimality
• Branch and Bound
**Best First Search (BestFS)**

- Idea: always choose the path on the frontier with the smallest h value.
- BestFS treats the frontier as a priority queue ordered by h.
- **Greedy** approach: expand path whose last node seems closest to the goal - chose the solution that is *locally* the best.

Let’s see how this works in Alspace: in the Search Applet toolbar

- select the “Vancouver Neighborhood Graph” problem
- set “Search Options -> Search Algorithms” to “Best-First”.
- select “Show Node Heuristics” under “View”
- compare number of nodes expanded by BestFS and LCFS
Analysis of BestFS

• Complete?

A. YES  B. NO
Analysis of BestFS

- Complete?

  B. NO

- See the “misleading heuristics demo” example in AISPACE
Analysis of BestFS

• Optimal?

A. YES  B. NO
Analysis of BestFS

• Optimal?

B. NO

• Try this example in AISPACE or example “ex-best.txt” from schedule page (save it and then load using “load from file” option)
Analysis of BestFS

• Time and space Complexity: $O(b^m)$
  • Worst case (bad $h(n)$): has to explore all nodes and keep related partial paths on the frontier

• Why would one want to use Best First Search?
  • Because if the heuristics is good it can find the solution very fast.
  • For another example, see AIspace, Delivery problem graph with C1 linked to o123 (cost 3.0)
What’s Next?

• Thus, having estimates of the distance to the goal can speed things up a lot
  • but by itself it can also mislead the search (i.e. Best First Search)

• On the other hand, taking only path costs into account allows LCSF to find the optimal solution
  • but the search process is still uniformed as far as distance to the goal goes.
How can we more effectively use $h(p)$ and $\text{cost}(p)$?

Shall we select from the frontier the path $p$ with:

A. Lowest $\text{cost}(p) - h(p)$

B. Highest $\text{cost}(p) - h(p)$

C. Highest $\text{cost}(p) + h(p)$

D. Lowest $\text{cost}(p) + h(p)$
How can we more effectively use $h(p)$ and $cost(p)$?

 Shall we select from the frontier the path $\rho$ with:

A. Lowest $\text{cost}(\rho) - h(\rho)$

B. Highest $\text{cost}(\rho) - h(\rho)$

C. Highest $\text{cost}(\rho) + h(\rho)$

D. Lowest $\text{cost}(\rho) + h(\rho)$
• Recap of Lecture 4
• Heuristic search
  • Best First Search
  • A*
    ✓ Admissible heuristic
    ✓ Completeness and Optimality
  • Branch and Bound

Slide 36
A* Search

- A* search takes into account both
  - the **cost** of the path \( p \) to a node \( c(p) \)
  - the **heuristic value** of that path \( h(p) \) (i.e. *the h value of the node n at the end of p*)

- Let \( f(p) = c(p) + h(p) \).
  - \( f(p) \) is an **estimate** of the cost of a path from the start to a goal via \( p \)

A* always chooses the path on the frontier with the lowest **estimated** distance from the start to a goal node constrained to go via that path.
Computing f-values

F-value of ubc → kd → jb?

A. 6  B. 9  C. 10  D. 11
Computing f-values

F-value of ubc $\rightarrow$ kd $\rightarrow$ jb?

C.10
Algorithm Selected: A*

PREVIOUS PATH2:
- UBC

NEW FRONTIER:
- Node: KD  Path Cost: 3.0  h(KD): 6.0  f-value: 9.0  Path: UBC --> KD
A* Search

- A* search takes into account both
  - the cost of the path to a node $c(p)$
  - the heuristic value of that path $h(p)$.

- Let $f(p) = c(p) + h(p)$.
  - $f(p)$ is an estimate of the cost of a path from the start to a goal via $p$.

A* always chooses the path on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that path.
Optimality of A*

A* is complete (finds a solution, if one exists) and optimal (finds the optimal path to a goal) if:

- arc costs are $> \varepsilon > 0$
- $h(n)$ is admissible

The book also mentions explicitly that the branching factor $b$ has to be finite, which we have been assuming by default (without this condition even BFS would not be complete).
Lecture Overview

• Recap of Lecture 4
• Best First Search
• A*
  Admissible heuristic
  • Completeness and Optimality
• Branch and Bound
Def.: Let $c(n)$ denote the cost of the optimal path from node $n$ to any goal node. A search heuristic $h(n)$ is called **admissible** if $h(n) \leq c(n)$ for all nodes $n$, i.e. if for all nodes it is an **underestimate** of the cost to any goal.

- Example: is the straight-line distance (SLD) admissible?
Admissibility of a heuristic

Def.:
Let \( c(n) \) denote the cost of the optimal path from node \( n \) to any goal node. A search heuristic \( h(n) \) is called **admissible** if \( h(n) \leq c(n) \) for all nodes \( n \), i.e. if for all nodes it is an underestimate of the cost to any goal.

- Example: is the straight-line distance admissible? **YES**

- The shortest distance between two points is a line.
Admissibility of a heuristic

Def.:
Let $c(n)$ denote the cost of the optimal path from node $n$ to any goal node. A search heuristic $h(n)$ is called **admissible** if $h(n) \leq c(n)$ for all nodes $n$, i.e. if for all nodes it is an underestimate of the cost to any goal.

**example:**
the goal is Urzizeni (red box), but all we know is the straight-line distances (sld) to Bucharest (green box)

- Possible $h(n) = \text{sld}(n, \text{Bucharest}) + \text{cost(Bucharest, Urzineni)}$
- Admissible?
  - A. Yes
  - B. No
  - C. It depends
Admissibility of a heuristic

Def.: Let \( c(n) \) denote the cost of the optimal path from node \( n \) to any goal node. A search heuristic \( h(n) \) is called **admissible** if \( h(n) \leq c(n) \) for all nodes \( n \), i.e. if for all nodes it is an underestimate of the cost to any goal.

**Example:**
the goal is Urzizeni (red box), but all we know is the straight-line distances to Bucharest (green box)

- Possible \( h(n) = \text{sld}(n, \text{Bucharest}) + \text{cost}(\text{Bucharest}, \text{Urzineni}) \)
- Admissible?
  - **NO**
  - Actual cost of going from **Vastul** to **Urzineni** is shorter than this estimate
Example 2

- **Search problem**: robot has to find a route from start to goal location on a grid with obstacles
- **Actions**: move *up*, *down*, *left*, *right* from tile to tile
- **Cost**: number of moves
- **Possible h(n)?** *Manhattan distance* \((L_1 \text{ distance})\) between two points \((x_1, y_1), (x_2, y_2)\):
  - sum of the (absolute) difference of their coordinates
  - \(|x_2 - x_1| + |y_2 - y_1|\)

ADMISSIBLE?
Example 2

- **Search problem**: robot has to find a route from start to goal location on a grid with obstacles
- **Actions**: move *up*, *down*, *left*, *right* from tile to tile
- **Cost**: number of moves
- **Possible h(n)?** *Manhattan distance* ($L_1$ distance) between two points $(x_1, y_1), (x_2, y_2)$:
  - sum of the (absolute) difference of their coordinates $|x_2 - x_1| + |y_2 - y_1|$

**ADMISSIBLE?**
Yes. Manhattan distance is the shortest path between any two tiles of the grid given the actions available and no walls. Including the walls will force the agent to take some extra steps to avoid them
An admissible heuristics for the 8-puzzle is?

A. Number of misplaced tiles plus number of correctly place tiles

B. Number of misplaced tiles

C. Number of correctly placed tiles

D. None of the above
Example 3: Eight Puzzle

• Admissible $h(n)$:
  
  **Number of Misplaced Tiles**: One needs at least that many moves to get the board in the goal state.

  ![Start State](image1.png) ![Goal State](image2.png)

  A and C clearly generate overestimates (e.g. when all tiles are in the correct position with respect to the goal above, except for 4 which is in the center).
Example 3: Eight Puzzle

• Another possible $h(n)$:
  Sum of number of moves between each tile's current position and its
goal position (we can move over other tiles in the grid)

Start State

Goal State

\[
\begin{array}{ccc}
5 & 4 & \ \ \\
6 & 1 & 8 \\
7 & 3 & 2 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 2 & 3 \\
8 & 4 & \ \\
7 & 6 & 5 \\
\end{array}
\]

Sum (}
Example 3: Eight Puzzle

- Another possible $h(n)$:
  Sum of number of moves between each tile's current position and its goal position

\[
\begin{align*}
\text{Start State} & : & 5 & 4 & 1 & 2 & 6 & 7 & 3 & 8 \\
\text{Goal State} & : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{sum} & : & 2 & 3 & 3 & 2 & 4 & 2 & 0 & 2 & = 18
\end{align*}
\]

Admissible?

A. Yes   B. No   C. It depends
Example 3: Eight Puzzle

- Another possible $h(n)$:
  Sum of number of moves between each tile's current position and its goal position

\[
\begin{array}{cccccccc}
5 & 4 & & & & & & \\
6 & 1 & 8 & & & & & \\
7 & 3 & 2 & & & & & \\
\end{array}
\quad
\begin{array}{cccccccc}
1 & 2 & 3 & & & & & \\
8 & 4 & & & & & & \\
7 & 6 & 5 & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
\text{sum } (2 \ 3 \ 3 \ 2 \ 4 \ 2 \ 0 \ 2) = 18
\]

Admissible? YES! One needs to make at least as many moves to get to the goal state when constrained by the grid structure
How to Construct an Admissible Heuristic

• Identify relaxed version of the problem:
  - where one or more constraints have been dropped (e.g., fewer restrictions on the actions)
• Grid world: the agent can
• Driver: the agent can
• 8 puzzle:
  - “number of misplaced tiles”: tiles can
  - “sum of moves between current and goal position”: tiles can

Why does this lead to an admissible heuristic?
  - The problem gets \ 
  -
How to Construct an Admissible Heuristic

- Identify **relaxed version** of the problem:
  - where one or more constraints have been dropped
  - problem with fewer restrictions on the actions
- **Grid world**: the agent **can move through walls**
- **Driver**: the agent **can move straight**
- **8 puzzle**:
  - “number of misplaced tiles”:
    tiles can move everywhere and occupy same spot as others
  - “sum of moves between current and goal position”:
    tiles can occupy same spot as others

Why does this lead to an admissible heuristic?

- The problem only gets **easier**!
- Less costly to solve it
How to Construct a Heuristic (cont.)

• You should identify constraints which, when dropped, make the problem easy to solve
  • important because heuristics are not useful if they're as hard to solve as the original problem!

This was the case in our examples

**Robot**: allowing the agent to move through walls. Optimal solution to this relaxed problem is *Manhattan distance*

**Driver**: allowing the agent to move straight. Optimal solution to this relaxed problem is *straight-line distance*

**8puzzle**: tiles can move anywhere. Optimal solution to this relaxed problem is *number of misplaced tiles*
Apply basic properties of search algorithms:
- completeness, optimality, time and space complexity

<table>
<thead>
<tr>
<th>Search Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>N (Y if no cycles)</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>LCFS (when arc costs available)</td>
<td>Y Costs &gt; $\varepsilon$</td>
<td>Y Costs &gt;=0</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Best First (when $h$ available)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Apply basic properties of search algorithms:
- completeness, optimality, time and space complexity

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>N (Y if no cycles)</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>LCFS</td>
<td>Y Costs &gt; $\epsilon$</td>
<td>Y Costs &gt;=0</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Best First</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>
Learning Goals for Search (up to today)

- Apply basic properties of search algorithms:
  - completeness, optimality
  - time and space complexity
- Select the most appropriate search algorithms for specific problems.
  - Depth-First Search vs. Bredth-First Search vs. Iterative Deepening vs. Least-Cost-First Search, Best First Search
- Define/read/write/trace/debug different search algorithms
- Construct heuristic functions and discuss their admissibility for specific search problems
• Do practice exercises 3C and 3D
• Read Ch 3.7 (More sophisticated search)

• Start working on Assignment 1! You can already do many of the questions