Search 2

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

February 2023
This Lecture

- Informed search (a.k.a. heuristic search)
- Application in game-playing
This Lecture

- Informed search (a.k.a. heuristic search)
- Application in game-playing
Generic Search Template: Pseudocode

- Primary data element is a Node, which a tuple of the form 

\[(\text{state}, \text{pathFromStartState}, \text{pathCost})\].
Generic Search Template: Pseudocode

- Primary data element is a **Node**, which a tuple of the form 
  
  \[(state, pathFromStartState, pathCost)\].

- At every stage of the search,
  - some states have been **explored**
  - some states remain **unexplored**, and
  - The *Frontier* is a set of nodes due for imminent expansion.
Generic Search Template: Pseudocode

- Primary data element is a **Node**, which a tuple of the form
  
  \((state, pathFromStartState, pathCost)\).

- At every stage of the search,
  - some states have been **explored**
  - some states remain **unexplored**, and
  - The **Frontier** is a set of nodes due for imminent expansion.

```
Frontier ← \{Node(startState, (startState), 0)\}.

Repeat for ever:

  Select a node \(n\) from Frontier.
  //Expand \(n\).
  If isGoal(n.state):
      Return \(n\).
  For each action \(a\) available from \(n.state\):
      \(s ← NextState(n.state, a)\).
      \(c ← Cost(n.state, a)\).
      \(n' ← Node(s, n.path + (a, s), n.pathCost + c)\).
      Merge \(n'\) with Frontier. //Typically insertion; might allow deletions.
```
Generic Search Template: Pseudocode

- Primary data element is a **Node**, which a tuple of the form

  
  $$(state, pathFromStartState, pathCost)$$.

- At every stage of the search,
  - some states have been **explored**
  - some states remain **unexplored**, and
  - The **Frontier** is a set of nodes due for imminent expansion.

```
Frontier ← \{Node(startState, (startState), 0)\}.
Repeat for ever:
    Select a node \(n\) from Frontier.//How is this selection made?
    //Expand \(n\).
    If isGoal\(n\).state):
        Return \(n\).
    For each action \(a\) available from \(n\).state:
        \(s ← \text{NextState} (n\).state, a)\).
        \(c ← \text{Cost} (n\).state, a)\).
        \(n' ← \text{Node} (s, n.path + (a, s), n.pathCost + c)\).
        Merge \(n'\) with Frontier.//Typically insertion; might allow deletions.
```
Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.

  Powai

  Mahim
Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.

First you expand the Powai node.

- Mahim

Which node will you expand next? L&T and Hiranandani are geographically closer to Mahim: should that count?
Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.

First you expand the Powai node.
Which node will you expand next?

- Mahim
Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.

First you expand the Powai node.
- Which node will you expand next?
- L&T and Hiranandani are geographically closer to Mahim: should that count?

- Mahim
Heuristic Functions and A* Search Algorithm

- A **heuristic** function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$. 
Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.\text{state}.x - \text{Mahim}.x)^2 + (n.\text{state}.y - \text{Mahim}.y)^2}.$$
Heuristic Functions and A* Search Algorithm

- A **heuristic** function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

  $$h(n) = \sqrt{(n.\text{state}.x - \text{Mahim}.x)^2 + (n.\text{state}.y - \text{Mahim}.y)^2}.$$ 

- Recall that in LCFS, we expand

  $$\arg\min_{n \in \text{Frontier}} g(n).$$
Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$ 

- Recall that in LCFS, we expand

$$\arg\min_{n \in \text{Frontier}} g(n).$$ 

- In A* search, we expand

$$\arg\min_{n \in \text{Frontier}} (g(n) + h(n)).$$
Heuristic Functions and A* Search Algorithm

- A **heuristic** function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$  

- Recall that in LCFS, we expand

$$\text{argmin}_{n \in \text{Frontier}} g(n).$$

- In A* search, we expand

$$\text{argmin}_{n \in \text{Frontier}} (g(n) + h(n)).$$

- $g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).
A heuristic function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

$h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - \text{Mahim}.x)^2 + (n.state.y - \text{Mahim}.y)^2}.$$ 

Recall that in LCFS, we expand

$$\arg\min_{n \in \text{Frontier}} g(n).$$

In A* search, we expand

$$\arg\min_{n \in \text{Frontier}} (g(n) + h(n)).$$

$g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).

The addition of $h(n)$ makes A* an informed or heuristic search algorithm.
Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^*(n)$, the optimal path-cost-to-goal of (the state in) node $n$.

- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$ 

- Recall that in LCFS, we expand

$$\arg\min_{n \in \text{Frontier}} g(n).$$

- In A* search, we expand

$$\arg\min_{n \in \text{Frontier}} (g(n) + h(n)).$$

- $g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).
- The addition of $h(n)$ makes A* an informed or heuristic search algorithm.

- A* search was originally conceived for robotic path planning.
A heuristic $h$ is **admissible** if for all nodes $n$,

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

where $c^*(n)$ is the optimal cost-to-goal of $n.state$. 

Is straight line distance an admissible heuristic for navigation? Yes.

For a given task, which is the best heuristic function to use?
Admissible Heuristics

- A heuristic $h$ is **admissible** if for all nodes $n$,

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

where $c^*(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result.** If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).
Admissible Heuristics

- A heuristic $h$ is **admissible** if for all nodes $n$,

\[ 0 \leq h(n) \leq c^*(n), \]

where $c^*(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result.** If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have **optimal** path-cost from the start state (and the algorithm can terminate).

- Is straight line distance an admissible heuristic for navigation?
Admissible Heuristics

- A heuristic $h$ is **admissible** if for all nodes $n$,

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

where $c^*(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result.** If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have **optimal** path-cost from the start state (and the algorithm can terminate).

- Is straight line distance an admissible heuristic for navigation?
  Yes.
Admissible Heuristics

- A heuristic $h$ is **admissible** if for all nodes $n$,
  \[ 0 \leq h(n) \leq c^*(n), \]
  where $c^*(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result.** If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).

- Is straight line distance an admissible heuristic for navigation?
  Yes.

- For a given task, which is the best heuristic function to use?
Effect of Heuristic

Start

Destination
Effect of Heuristic

\[ h(n) = c^*(n) \]. Will only expand nodes along optimal path! But \( c^*(n) \) not known!
Effect of Heuristic

\[ h(n) = 0. \text{ Identical to LCFS.} \]
Effect of Heuristic

Intermediate/typical $h(n)$ expands fewer nodes than LCFS.
Questions

- How to come up with an effective admissible heuristic for a task?
Questions

How to come up with an effective admissible heuristic for a task?

For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What’s a good heuristic for 15-puzzle?

<table>
<thead>
<tr>
<th>Start state</th>
<th>Goal state</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 2 3 12</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>9 11 1 10</td>
<td>5 6 7 8</td>
</tr>
<tr>
<td>● 5 4 14</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>15 8 7 6</td>
<td>13 14 15  ●</td>
</tr>
</tbody>
</table>

Sum of Manhattan distances between each number's position in start state and its position in goal state.
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What’s a good heuristic for 15-puzzle?

  Start state | Goal state
  --- | ---
  13 2 3 12 | 1 2 3 4
  9 11 1 10 | 5 6 7 8
  5 4 14 | 9 10 11 12
  15 8 7 6 | 13 14 15

  Sum of Manhattan distances between each number’s position in start state and its position in goal state.
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What’s a good heuristic for 15-puzzle?

- Sum of Manhattan distances between each number’s position in start state and its position in goal state.

- Can we make do with inadmissible heuristics?
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What's a good heuristic for 15-puzzle?

  ![15-puzzle grid]

  Start state  
  
  Goal state  

  Sum of Manhattan distances between each number’s position in start state and its position in goal state.

- Can we make do with inadmissible heuristics?
  Yes—example coming up in next section. But try to avoid.
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What’s a good heuristic for 15-puzzle?

<table>
<thead>
<tr>
<th>Start state</th>
<th>Goal state</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 2 3 12</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>9 11 1 10</td>
<td>5 6 7 8</td>
</tr>
<tr>
<td>● 5 4 14</td>
<td>9 10 11 12</td>
</tr>
<tr>
<td>15 8 7 6</td>
<td>13 14 15 ●</td>
</tr>
</tbody>
</table>

  Sum of Manhattan distances between each number’s position in start state and its position in goal state.

- Can we make do with inadmissible heuristics?
  Yes—example coming up in next section. But try to avoid.

- Is A* search used widely in practice?
Questions

- How to come up with an effective admissible heuristic for a task?
  For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What’s a good heuristic for 15-puzzle?

  ![15-puzzle diagram]

  Start state
  \[
  \begin{array}{cccc}
  13 & 2 & 3 & 12 \\
  9 & 11 & 1 & 10 \\
  \bullet & 5 & 4 & 14 \\
  15 & 8 & 7 & 6 \\
  \end{array}
  \]

  Goal state
  \[
  \begin{array}{cccc}
  1 & 2 & 3 & 4 \\
  5 & 6 & 7 & 8 \\
  9 & 10 & 11 & 12 \\
  13 & 14 & 15 & \bullet \\
  \end{array}
  \]

  Sum of Manhattan distances between each number’s position in start state and its position in goal state.

- Can we make do with inadmissible heuristics?
  Yes—example coming up in next section. But try to avoid.

- Is A* search used widely in practice?
  Yes. Along with variants such as IDA*.
This Lecture

- Informed search (a.k.a. heuristic search)

- Application in game-playing
Chess
Checkers/Draughts

Can winning at chess/checkers be posed as a search problem?

There's another player!


Can winning at chess/checkers be posed as a search problem?

What's the main difference from our previous examples?

There's another player!
Can winning at chess/checkers be posed as a search problem?


Can winning at chess/checkers be posed as a search problem?

What’s the main difference from our previous examples?


Can winning at chess/checkers be posed as a search problem?

What’s the main difference from our previous examples? There’s another player!


Game Tree

- Assume turn-taking zero sum game with two players, Max and Min.
- Action costs usually taken as 0, but leaves have value
  
  $-1$ (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.

\[
\begin{array}{c|c|c|c}
\text{Max} & \text{Leaf} & \text{Min} \\
\hline
1 & 1 & 0 \\
1 & 0 & -1 \\
\end{array}
\]

In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).

The Checkers game tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$. 
Game Tree

- Assume turn-taking zero sum game with two players, Max and Min.
- Action costs usually taken as 0, but leaves have value
  \[-1\] (Max loses), \(0\) (draw), \(1\) (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?

In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is \(0\) (draw).

The Checkers game tree has \(\approx 10^{40}\) nodes; Chess has \(\approx 10^{120}\).
Game Tree

- Assume turn-taking zero sum game with two players, Max and Min.
- Action costs usually taken as 0, but leaves have value
  - 1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?

In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).

The Checkers game tree has \( \approx 10^{40} \) nodes; Chess has \( \approx 10^{120} \).
Game Tree

- Assume turn-taking zero sum game with two players, Max and Min.
- Action costs usually taken as 0, but leaves have value $-1$ (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children. Value of Min node is minimum of values of children.
- What is the value of the root node?
Assume turn-taking zero sum game with two players, Max and Min.

Action costs usually taken as 0, but leaves have value $-1$ (Max loses), 0 (draw), 1 (Max wins).

Value of Max node is maximum of values of children.
Value of Min node is minimum of values of children.

What is the value of the root node?

In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).
Game Tree

- Assume turn-taking zero sum game with two players, Max and Min.
- Action costs usually taken as 0, but leaves have value $-1$ (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
- Value of Min node is minimum of values of children.
- What is the value of the root node?

In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).

The Checkers game tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$.
Evaluation Function

- What if game tree depth/size makes it infeasible to solve?

```
<table>
<thead>
<tr>
<th>Max</th>
<th>Leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>
```

Cannot explore beyond this depth!
Evaluation Function

- What if game tree depth/size makes it infeasible to solve?

- At some depth $d$ from current node, estimate node value using features.
  - For example, in Chess, set evaluation as
    \[ w_1 \times \text{Material difference} + w_2 \times \text{King safety} + w_3 \times \text{pawn strength} + \ldots. \]
  - Weights $w_1, w_2, w_3, \ldots$ are tuned or learned from experience.