Reminder: Technology Policy

No technology except for first four rows of left and right sides
Reminder: Due Feb 4

• Special accommodations letter:
  Scan documentation letter and email to FAI-L@cornell.edu

• Prelim and Final Exam conflicts (3420 – larger class rule)
Reminder: First Karma Lecture

The Principal-Agent Value Alignment Problem in Artificial Intelligence

Dylan Hadfield-Menell
UC Berkeley

Jan 28 11:40am-12:40pm
Gates G01
Formalizing a Problem as State Space Search

Define:

- States
  - Initial State
  - Goal State/Condition
- Actions (aka Operators):
Missionaries and Cannibals
Missionaries and Cannibals

1. States and operators define a graph
2. The graph is generated dynamically, not given up front
Search

• A search algorithm methodically explores the space of possible states

• At any point during the search there will be:
  • States that have already been seen – “seen” means
    (1) the state has been checked if it’s a goal, and
    (2) if not, we’ve generated a list of its successors
  • Terminology: These states have been “expanded” or “closed”
Search

• A search algorithm methodically explores the space of possible states

• At any point during the search there will be:
  • States that have already been seen — “seen” means
    (1) the state has been checked if it’s a goal, and
    (2) if not, we’ve generated a list of its successors
    • Terminology: These states have been “expanded” or “closed”
  • States that have been generated but not seen
    • Terminology: These states are “open”
Example: Snapshot of the search from last time for Missionaries and Cannibals
Search

• A search algorithm methodically explores the space of possible states

• At any point during the search there will be:
  • States that have already been seen – “seen” means
    (1) the state has been checked if it’s a goal, and
    (2) if not, we’ve generated a list of its successors
    • Terminology: These states have been “expanded” or “closed”
  • States that have been generated but not seen
    • Terminology: These states are “open”
Search

• A search algorithm methodically explores the space of possible states

• At any point during the search there will be:
  • States that have already been seen – “seen” means
    (1) the state has been checked if it’s a goal, and
    (2) if not, we’ve generated a list of its successors
      • Terminology: These states have been “expanded” or “closed”
  • States that have been generated but not seen
    • Terminology: These states are “open”
  • States that have not been seen (yet)
Example: Snapshot of the search from last time for Missionaries and Cannibals

Closed states

(3,3,L)

(2,3,R)

(1,3,R)

(3,2,R)

(3,1,R)

Open states

(3,2,L)

(2,2,R)

Unseen states (one of which is hopefully a goal state)
Two Basic Search Methods

Depth-First Search

Breadth-First Search
Two Basic Search Methods

Depth-First Search

Always generate successors of one of the newest open nodes

Breadth-First Search
Two Basic Search Methods

Depth-First Search

Always generate successors of one of the newest open nodes

Breadth-First Search

Used on Missionaries and Cannibals earlier
Sample Search Space (Tree)
Depth-First Search

Leaf node
Depth-First Search
Depth-First Search

Leaf nodes
Depth-First Search

These are tied – all equally new
Depth-First Search
Depth-First Search

These are tied – all equally new
In the absence of information to the contrary
use convention of left-to-right in case of ties
This would also be a depth-first search
Depth-First Search

Leaf nodes
Depth-First Search

RED NODES: Closed
YELLOW NODES: Open
BLUE NODES: Unknown to algorithm at this point
Depth-First Search

Order of search if this is the entire state space
## Two Basic Search Methods

<table>
<thead>
<tr>
<th>Depth-First Search</th>
<th>Breadth-First Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always generate successors of one of the newest open nodes</td>
<td>Always generate successors of one of the oldest open nodes</td>
</tr>
</tbody>
</table>
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search
Breadth-First Search

Order of search if this is the entire state space
A search algorithm methodically explores the space of possible states. At any point during the search there will be:

- States that have already been seen – “seen” means
  1. the state has been checked if it’s a goal, and
  2. if not, we’ve generated a list of its successors
- Terminology: These states have been “expanded” or “closed”
- States that have been generated but not seen
- Terminology: These states are “open”
- States that have not been seen (yet)
Example: Snapshot of Breadth-First Search

About to become closed – it will be expanded next

Unseen, but about to become open
Search Algorithm Template

Initial call: \texttt{Search(initialstate,ops,\{}{},{}\})

\texttt{Search(s,ops,open,closed) =}
\hspace{1em} If goal(s) Then return(s);
\hspace{1em} Else If not(s \in closed)
\hspace{2em} Then
\hspace{3em} successors \leftarrow \{\}; add(s,\text{closed});
\hspace{3em} For each \( o \in \text{ops} \) that applies to \( s \)
\hspace{4em} add apply(o,s) to successors
\hspace{3em} open \leftarrow add successors to open;
\hspace{1em} If not(empty(open))
\hspace{2em} s' \leftarrow \text{select}(open);
\hspace{2em} open \leftarrow remove(s',open);
\hspace{2em} \texttt{search}(s',\text{ops},\text{open},\text{closed})
\hspace{1em}` Else return(FAIL)
Search Algorithm Template

Initial call: \textbf{Search}((initialstate,ops,\{\},\{\})

\textbf{Search}(s,ops,open,closed) =
  \textbf{If} \textbf{goal}(s) \textbf{Then return}(s);
  \\textbf{Else If not}(s \in \text{closed})
  \\textbf{Then}
  \text{successors} \leftarrow \{\}; \text{add}(s,\text{closed});
  \text{For each } o \in \text{ops} \text{ that applies to } s
  \text{ add } \text{apply}(o,s) \text{ to successors}
  \text{open} \leftarrow \text{add successors to open};
  \textbf{If not}(\text{empty}(\text{open}))
  s' \leftarrow \text{select}(\text{open});
  \text{open} \leftarrow \text{remove}(s',\text{open});
  \textbf{search}(s',\text{ops,open,closed})
  \\textbf{Else return}(\text{FAIL})

\textbf{Goal}(s): \text{Returns true if } s \text{ is a goal, false otherwise}
Search Algorithm Template

Initial call: \textbf{Search}((initialstate,ops,\{\},\{\}))

\textbf{Search}(s,ops,open,closed) =

\begin{align*}
\text{If goal}(s) \text{ Then return}(s) ; \\
\text{Else If not}(s \in \text{closed}) \\
\text{Then} \\
\quad \text{successors} \leftarrow \{\} ; \text{add}(s,\text{closed}) ; \\
\quad \text{For each } o \in \text{ops that applies to } s \\
\quad \quad \text{add apply}(o,s) \text{ to successors} \\
\quad \text{open} \leftarrow \text{add successors to open} ; \\
\quad \text{If not}(\text{empty}(\text{open})) \\
\quad \quad s' \leftarrow \text{select}(\text{open}) ; \\
\quad \quad \text{open} \leftarrow \text{remove}(s',\text{open}) ; \\
\quad \quad \text{\textbf{search}}(s',\text{ops,open,closed}) \\
\quad \text{Else return}(\text{FAIL})
\end{align*}

apply(o,s): Returns the state you get from applying o to s
Search Algorithm Template

Initial call: Search(initialstate,ops,{},{})

Search(s,ops,open,closed) =
  If goal(s) Then return(s);
  Else If not(s ∈ closed)
    Then
      successors ← {}; add(s,closed);
      For each o ∈ ops that applies to s
        add apply(o,s) to successors
      open ← add successors to open;
      If not(empty(open))
        s’ ← select(open);  \textbf{select: defines the search method}
        open ← remove(s’,open);
        search(s’,ops,open,closed)
    \ Else return(FAIL)
Initial call: \texttt{Search(initialstate,ops,{},{})}

\texttt{Search(s,ops,open,closed) =}
  If goal(s) Then return(s);
  Else If not(s \in closed)
    Then
      successors \leftarrow \{\}; add(s,\text{closed});
      For each o \in ops that applies to s
        add apply(o,s) to successors
      open \leftarrow add successors to open;
    If not(\text{empty}(\text{open}))
      s' \leftarrow \text{select}(\text{open});
      open \leftarrow remove(s',\text{open});
      \text{search}(s',\text{ops,open,closed})
    Else return(FAIL)

\texttt{select}: defines the search method
What if \texttt{select} = newest?
Search Algorithm Template

Initial call: \texttt{Search}(initialstate,ops,\{\},\{\})

\texttt{Search}(s,ops,open,closed) =
\begin{enumerate}
  \item If goal(s) Then return(s);
  \item Else If not(s \in closed) Then
    \begin{enumerate}
      \item successors $\leftarrow \{\};$ add(s,closed);
      \item For each o $\in$ ops that applies to s
        \begin{enumerate}
          \item add apply(o,s) to successors
        \end{enumerate}
      \end{enumerate}
    \end{enumerate}
    \begin{enumerate}
      \item open $\leftarrow$ add successors to open;
      \item If not(empty(open))
        \begin{enumerate}
          \item s' $\leftarrow$ \texttt{select}(open);
          \item open $\leftarrow$ remove(s',open);
          \item \texttt{search}(s',ops,open,closed)
        \end{enumerate}
        \item Else return(FAIL)
    \end{enumerate}
\end{enumerate}

\texttt{select}: defines the search method
\begin{itemize}
  \item What if select = newest? DFS
\end{itemize}
Search Algorithm Template

Initial call: \texttt{Search(initialstate,ops,\{}\{}\}\},{}\})

\texttt{Search(s,ops,open,closed) =}
\begin{itemize}
  \item If goal(s) Then return(s); \\
  \item Else If not(s \in closed) Then \\
    \begin{itemize}
      \item successors \assign \{}\}; add(s,closed); \\
      \item For each o \in ops that applies to s add apply(o,s) to successors \\
      \item open \assign add successors to open;
    \end{itemize}
  \item Else return(FAIL)
\end{itemize}

select: defines the search method
What if select = oldest?
Search Algorithm Template

Initial call: \texttt{Search(initialstate,ops,{},{}))}

\texttt{Search(s,ops,open,closed) = }
\hspace{1cm} \texttt{If goal(s) Then return(s);}
\hspace{1cm} \texttt{Else If not(s } \in \texttt{ closed) Then}
\hspace{2cm} \texttt{successors } \leftarrow \{\}; \texttt{add(s,closed);}
\hspace{2cm} \texttt{For each o } \in \texttt{ ops that applies to s}
\hspace{3cm} \texttt{add apply(o,s) to successors}
\hspace{2cm} \texttt{open } \leftarrow \texttt{add successors to open;}
\hspace{1cm} \texttt{If not(empty(open))}
\hspace{2cm} \texttt{s' } \leftarrow \texttt{select(open); select: defines the search method}
\hspace{2cm} \texttt{open } \leftarrow \texttt{remove(s’,open);}
\hspace{2cm} \texttt{search(s’,ops,open,closed)}
\hspace{1cm} \texttt{\` Else return(FAIL) What if select = oldest? BFS}
Search Algorithm Template vs Implementation

If you are implementing a search system you would likely use data structures for Open and Closed so that Select is fast

Examples: Stack for DFS, Queue for BFS
Search Algorithm Template vs Implementation

How does the algorithm return the sequence of actions?
Search Algorithm Template

Initial call: \textbf{Search}(\text{initialstate}, \text{ops}, \{\}, \{\})

\textbf{Search}(s, \text{ops}, \text{open}, \text{closed}) =

\hspace{20pt}\text{If goal}(s) \text{ Then return}(s); \\
\hspace{20pt}\text{Else If not}(s \in \text{closed}) \\
\hspace{40pt}\text{Then} \\
\hspace{60pt}\text{successors} \gets \{\}; \text{add}(s, \text{closed}); \\
\hspace{60pt}\text{For each } o \in \text{ops that applies to } s \\
\hspace{80pt}\text{add apply}(o, s) \text{ to successors} \\
\hspace{60pt}\text{open} \gets \text{add successors to open}; \\
\hspace{20pt}\text{If not}(\text{empty}(\text{open})) \\
\hspace{40pt}s' \gets \text{select}(\text{open}); \\
\hspace{40pt}\text{open} \gets \text{remove}(s', \text{open}); \\
\hspace{40pt}\text{search}(s', \text{ops}, \text{open}, \text{closed}) \\
\hspace{20pt}\text{` Else return}(\text{FAIL})
Search Algorithm Template vs Implementation

How does the algorithm return the sequence of actions?

Template simplified some of the details:

s would be a data structure containing (at least)
the state and the sequence of actions to get here

“open ← add successors to open”

would need to update the data structure will all pertinent information
Evaluating a Search Method

• Complete?: Does it find a solution when one exists?
• Optimal?: Will it always find the optimal solution?
  (for now, “optimal” means fewest number of steps)
• Time: How long does it run?
• Space: How much space does it require?
# DFS vs BFS

<table>
<thead>
<tr>
<th>Method</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimal?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>$O()$</td>
<td>$O()$</td>
</tr>
<tr>
<td>Space</td>
<td>$O()$</td>
<td>$O()$</td>
</tr>
</tbody>
</table>
DFS vs BFS

• b: branching factor (finite)
• d: depth of solution (finite)

• Assume a tree of depth m
### DFS vs BFS: Handwavy Analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Optimal?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^m)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(bm)$</td>
<td>$O(b^d)$</td>
</tr>
</tbody>
</table>