16-350
Planning Techniques for Robotics

Search Algorithms:
Uninformed A* Search

Maxim Likhachev
Robotics Institute
Carnegie Mellon University
Searching Graphs for a Least-cost Path

• Once a graph is constructed (from skeletonization or cell decomposition or whatever else), we need to search it for a least-cost path.
Searching Graphs for a Least-cost Path

• Once a graph is constructed (from skeletonization or cell decomposition or whatever else), we need to search it for a least-cost path
Many searches (including A*) work by computing $g^*$ values for graph vertices (states)

- $g^*(s)$ – the cost of a least-cost path from $s_{\text{start}}$ to $s$
Many searches (including A*) work by computing $g^*$ values for graph vertices (states)

- $g^*(s)$ – the cost of a least-cost path from $s_{start}$ to $s$
Many searches (including A*) work by computing $g^*$ values for graph vertices (states)

- $g^*(s)$ – the cost of a least-cost path from $s_{start}$ to $s$

- $g^*$ values satisfy: $g^*(s) = \min_{s'' \in \text{pred}(s)} g^*(s'') + c(s'', s)$
Many searches (including A*) work by computing $g^*$ values for graph vertices (states)

- $g^*(s)$ – the cost of a least-cost path from $s_{\text{start}}$ to $s$
- $g^*$ values satisfy: $g^*(s) = \min_{s'' \in \text{pred}(s)} g^*(s'') + c(s'', s)$

Once $g^*$-values are computed, a least-cost path from $s_{\text{start}}$ to $s_{\text{goal}}$ can be easily computed!
• Least-cost path is a greedy path computed by backtracking:

- start with \( s_{goal} \) and from any state \( s \) backtrack to the predecessor state \( s' \) such that

\[
\text{arg min}_{s'' \in \text{pred}(s)} (g^*(s'') + c(s'', s))
\]
Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

How can we compute $g^*$-values?
Searching Graphs for a Least-cost Path

• Example on a Grid-based Graph:

Intuition behind uninformed A*:
Starting with the start state (marked R), always compute next the state with smallest g* value!

8-connected grid

How can we compute g* -values?
Searching Graphs for a Least-cost Path

• Example on a Grid-based Graph:

8-connected grid

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>3.4</td>
<td>3.8</td>
<td>4.2</td>
</tr>
<tr>
<td>2.8</td>
<td>2.4</td>
<td>2.8</td>
<td>3.8</td>
</tr>
<tr>
<td>2.4</td>
<td>1.4</td>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Carnegie Mellon University
Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

Use \( g^* \) to compute the least-cost path by back-tracking.
Searching Graphs for a Least-cost Path

• Example on a Grid-based Graph:

Use $g^*$ to compute the least-cost path by back-tracking
Computes $g^*$-values for relevant (not all) states at any point of time:

\[ g(s) \]

the cost of a shortest path from $s_{\text{start}}$ to $s$ found so far
Uninformed A* Search

- Computes $g^*$-values for relevant (not all) states

**Main function**

$g(s_{\text{start}}) = 0$; all other $g$-values are infinite; $OPEN = \{s_{\text{start}}\}$; ComputePath(); publish solution; //compute least-cost path using $g$-values

**ComputePath function**

while ($s_{\text{goal}}$ is not expanded and $OPEN \neq 0$)

- remove $s$ with the smallest $g(s)$ from $OPEN$;
- expand $s$;

_for every expanded state $g(s)$ is optimal ($g(s) = g^*(s)$)_

---

Carnegie Mellon University
Uninformed A* Search

• Computes g*-values for relevant (not all) states

ComputePath function
while($s_{\text{goal}}$ is not expanded and OPEN ≠ 0)
    remove $s$ with the smallest $g(s)$ from OPEN;
    expand $s$;
Uninformed A* Search

- Computes $g^*$-values for relevant (not all) states

ComputePath function

while($s_{goal}$ is not expanded and $OPEN \neq 0$)
  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into $OPEN$;

set of states that have already been expanded

tries to decrease $g(s')$ using the found path from $s_{start}$ to $s$
• Computes $g^*$-values for relevant (not all) states

**ComputePath function**
while($s_{goal}$ is not expanded and $OPEN \neq 0$)
  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
    insert $s'$ into $OPEN$;

$CLOSED = \{\}$
$OPEN = \{s_{start}\}$
next state to expand: $s_{start}$
• Computes g*-values for **relevant** (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and OPEN ≠ 0)

remove $s$ with the smallest $g(s)$ from OPEN;

insert $s$ into CLOSED;

for every successor $s'$ of $s$ such that $s'$ not in CLOSED

if $g(s') > g(s) + c(s,s')$

$g(s') = g(s) + c(s,s')$;

insert $s'$ into OPEN;

CLOSED = {}  
OPEN = {$s_{start}$}  
next state to expand: $s_{start}$
• Computes g*-values for **relevant** (not all) states

**ComputePath function**
while($s_{goal}$ is not expanded and OPEN \(\neq 0\))
  remove $s$ with the smallest $g(s)$ from OPEN;
  insert $s$ into CLOSED;
  for every successor $s'$ of $s$ such that $s'$ not in CLOSED
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into OPEN;
Computes g*-values for **relevant** (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and $OPEN \neq 0$)

  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into $OPEN$;

$CLOSED = \{s_{start}\}$
$OPEN = \{s_2\}$
next state to expand: $s_2$
Uninformed A* Search

• Computes $g^*$-values for relevant (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and $OPEN \neq 0$)
  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$  
      $g(s') = g(s) + c(s,s')$;
    insert $s'$ into $OPEN$;

$CLOSED = \{s_{start}, s_2\}$
$OPEN = \{s_1, s_4\}$
next state to expand: ?
- Computes g*-values for **relevant** (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and $OPEN \neq 0$)
remove $s$ with the smallest $g(s)$ from $OPEN$;
insert $s$ into $CLOSED$;
for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
  if $g(s') > g(s) + c(s,s')$
  $g(s') = g(s) + c(s,s')$;
  insert $s'$ into $OPEN$;

$CLOSED = \{s_{start}, s_2\}$
$OPEN = \{s_1, s_4\}$
next state to expand: $s_4$

Uninformed A* Search
Uninformed A* Search

• Computes \( g^* \)-values for relevant (not all) states

ComputePath function
while \( s_{goal} \) is not expanded and \( OPEN \neq 0 \)
  remove \( s \) with the smallest \( g(s) \) from \( OPEN \);
  insert \( s \) into \( CLOSED \);
  for every successor \( s' \) of \( s \) such that \( s' \) not in \( CLOSED \)
    if \( g(s') > g(s) + c(s,s') \)
      \( g(s') = g(s) + c(s,s') \);
      insert \( s' \) into \( OPEN \);

\( CLOSED = \{ s_{start}, s_2, s_4 \} \)
\( OPEN = \{ s_1, s_3 \} \)
next state to expand: ?
Uninformed A* Search

- Computes $g^*$-values for relevant (not all) states

\textbf{ComputePath function}

while($s_{goal}$ is not expanded and $OPEN \neq 0$)

\begin{itemize}
  \item remove $s$ with the smallest $g(s)$ from $OPEN$;
  \item insert $s$ into $CLOSED$;
  \item for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    \begin{itemize}
      \item if $g(s') > g(s) + c(s,s')$
      \item $g(s') = g(s) + c(s,s')$;
      \item insert $s'$ into $OPEN$;
    \end{itemize}
\end{itemize}

$CLOSED = \{s_{\text{start}}, s_2, s_4\}$

$OPEN = \{s_1, s_3\}$

next state to expand: $s_1$
Uninformed A* Search

• Computes $g^*$-values for relevant (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and $OPEN \neq 0$)
  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into $OPEN$;

$CLOSED = \{ s_{start}, s_2, s_4, s_1 \}$
$OPEN = \{ s_3, s_{goal} \}$
next state to expand: ?

Carnegie Mellon University
Uninformed A* Search

- Computes $g^*$-values for relevant (not all) states

**ComputePath function**

while ($s_{goal}$ is not expanded and OPEN ≠ 0)
  remove $s$ with the smallest $g(s)$ from OPEN;
  insert $s$ into CLOSED;
  for every successor $s'$ of $s$ such that $s'$ not in CLOSED
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into OPEN;

$CLOSED = \{s_{start}, s_2, s_4, s_1\}$

$OPEN = \{s_3, s_{goal}\}$

next state to expand: $s_{goal}$
Uninformed A* Search

- Computes g*-values for relevant (not all) states

ComputePath function
while($s_{goal}$ is not expanded and $OPEN \neq 0$)
    remove $s$ with the smallest $g(s)$ from $OPEN$;
    insert $s$ into $CLOSED$;
    for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
        if $g(s') > g(s) + c(s,s')$
            $g(s') = g(s) + c(s,s')$;
            insert $s'$ into $OPEN$;

$CLOSED = \{S_{start}, S_2, S_4, S_1, S_{goal}\}$
$OPEN = \{S_3\}$
done
Uninformed A* Search

- Computes $g^*$-values for relevant (not all) states

**ComputePath function**

while($s_{goal}$ is not expanded and $OPEN \neq 0$)
remove $s$ with the smallest $g(s)$ from $OPEN$;
insert $s$ into $CLOSED$;
for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
  if $g(s') > g(s) + c(s,s')$
    $g(s') = g(s) + c(s,s')$;
  insert $s'$ into $OPEN$;

for every expanded state $g(s) = g^*(s)$
for every other state $g(s) \geq g^*(s)$
we can now compute a least-cost path
Uninformed A* Search

• Computes g*-values for relevant (not all) states

\begin{verbatim}
ComputePath function
while(s_{goal} is not expanded and OPEN \neq 0)
    remove s with the smallest g(s) from OPEN;
    insert s into CLOSED;
    for every successor s' of s such that s' not in CLOSED
        if g(s') > g(s) + c(s,s')
            g(s') = g(s) + c(s,s');
            insert s' into OPEN;
\end{verbatim}

for every expanded state g(s)=g*(s)
for every other state g(s) \geq g*(s)
we can now compute a least-cost path
Uninformed A* Search

- Computes $g^*$-values for **relevant** (not all) states

ComputePath function
while($s_{goal}$ is not expanded and $OPEN \neq 0$)
  remove $s$ with the smallest $g(s)$ from $OPEN$;
  insert $s$ into $CLOSED$;
  for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
    insert $s'$ into $OPEN$;

for every expanded state $g(s) = g^*(s)$
for every other state $g(s) \geq g^*(s)$ **why?**
we can now compute a least-cost path
Theorem 1. For every expanded state \( s \), it is guaranteed that \( g(s) = g^*(s) \)

**Sketch of proof by induction:**
- consider state \( s \) getting selected for expansion and assume that all previously expanded states had their g-values equal to g*-values
- since \( s \) was selected for expansion, then \( g(s) \) – min among states in OPEN
- OPEN is a frontier of states that separates previously expanded states from the states that have never been seen by the search
- thus, the cost of the path from \( s_{start} \) to \( s \) via any state in OPEN or any state not previously seen will be worse than \( g(s) \) (assuming positive costs)
- therefore, \( g(s) \) (the cost of the best path found so far) is already optimal
Theorem 2. Once the search terminates, it is guaranteed that
\( g(s_{\text{goal}}) = g^*(s_{\text{goal}}) \)

**Sketch of proof:**
Theorem 3. Once the search terminates, the least-cost path from \( s_{\text{start}} \) to \( s_{\text{goal}} \) can be re-constructed by backtracking (start with \( s_{\text{goal}} \) and from any state \( s \) backtrack to the predecessor state \( s' \) such that \( s' = \arg\min_{s'' \in \text{pred}(s)} (g(s'') + c(s'', s)) \)).

Sketch of proof:
- every backtracking step from state \( s \) moves to a predecessor state \( s' \) that continues to be on a least-cost path (because all predecessors \( u \) not on a least-cost path will have have \( g(u) + \text{cost}(u, s) \) that are strictly larger than \( g(s') + \text{cost}(s', s) \)).
What You Should Know…

- Given $g^*$-values, how to re-construct a least-cost path

- Operation of Uninformed A*

- Properties of uninformed A* search
  - $g$-values of expanded states are optimal ($g=g^*$)
  - for every expanded state, one can re-construct a least-cost path to it via back-tracking

- Sketch of proof for why uninformed A* returns a least-cost path