Search Algorithms:
Heuristics,
Backward A*, Weighted A* Search
Maxim Likhachev
Robotics Institute
Carnegie Mellon University
A* Search

• Computes optimal g-values for relevant states

at any point of time:

one popular heuristic function – Euclidean distance
Heuristics

• Heuristic function must be:
  – admissible: for every state \( s \), \( h(s) \leq c^*(s, s_{\text{goal}}) \)
  – consistent (satisfy triangle inequality):
    \[ h(s_{\text{goal}}, s_{\text{goal}}) = 0 \text{ and for every } s \neq s_{\text{goal}} \text{, } h(s) \leq c(s, \text{succ}(s)) + h(\text{succ}(s)) \]
  – admissibility provably follows from consistency and often (not always) consistency follows from admissibility
Heuristics

• For X-connected grids:
  – Euclidean distance
  – Manhattan distance: $h(x,y) = abs(x-x_{goal}) + abs(y-y_{goal})$
  – Diagonal distance: $h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal}))$
  – More informed distances???

Which heuristics are admissible for 4-connected grid? 8-connected grid?
Heuristics

- For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Non-circular robot

![Diagram of a non-circular robot](image-url)
Heuristics

• For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Non-circular robot

Grid-based representation for planning:
x,y,θ for some reference point on the robot
x,y are on 8-connected grid
θ – discretized into 8 angles
Heuristics

- For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Grid-based representation for planning:
\(x, y, \theta\) for some reference point on the robot
\(x, y\) are on 8-connected grid
\(\theta\) – discretized into 8 angles

Non-circular robot

How many states?
Heuristics

What heuristic we can use?

• For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Non-circular robot

Grid-based representation for planning:
\(x, y, \Theta\) for some reference point on the robot
\(x, y\) are on 8-connected grid
\(\Theta\) – discretized into 8 angles
Heuristics

- For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Non-circular robot

Grid-based representation for planning:
\(x, y, \Theta\) for some reference point on the robot
\(x, y\) are on 8-connected grid
\(\Theta\) – discretized into 8 angles

Any ideas for heuristics that estimate cost-to-goal better?

How about cost-to-goal distances for the reference point in 2D (accounting for obstacles)?
Heuristics

• For planning problems higher than 2D

Example:
consider planning for a non-circular robot that can move in any direction (omnidirectional)

Non-circular robot

Grid-based representation for planning: 
\(x, y, \Theta\) for some reference point on the robot
\(x, y\) are on 8-connected grid
\(\Theta\) – discretized into 8 angles

G

Carnegie Mellon University
Backward A* Search

• Searching from the goal towards the start state

• g-values are cost-to-goals

Main function
\( g(s_{start}) = 0 \); all other g-values are infinite; \( OPEN = \{s_{start}\} \);
ComputePath();
publish solution;

ComputePath function
while (\( s_{goal} \) is not expanded and \( OPEN \neq 0 \))
  remove \( s \) with the smallest \( f(s) = g(s) + h(s) \) from \( OPEN \);
  expand \( s \);

What needs to be changed?
Backward A* Search

- Searching from the goal towards the start state
- g-values are cost-to-goals

Main function
\[ g(s_{\text{goal}}) = 0; \text{ all other } g\text{-values are infinite}; \, OPEN = \{s_{\text{goal}}\} ; \]
ComputePath();
publish solution;

ComputePath function
while \((s_{\text{start}} \) is not expanded and \( OPEN \neq 0 \))
  remove \( s \) with the smallest \([f(s) = g(s) + h(s)]\) from \( OPEN \);
  expand \( s \);

What needs to be changed?
Backward A* Search

- Searching from the goal towards the start state
- **g-values are cost-to-goals**

ComputePath function

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

remove $s$ with the smallest $[f(s) = g(s)+h(s)]$ 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 $OPEN$;

What needs to be changed in here?
Backward A* Search

- Searching from the goal towards the start state
- g-values are cost-to-goals

ComputePath function
while($s_{start}$ is not expanded and OPEN ≠ 0)
remove $s$ with the smallest $[f(s) = g(s)+h(s)]$ from OPEN;
insert $s$ into CLOSED;
for every predecessor $s'$ of $s$ such that $s'$ not in CLOSED
if $g(s') > c(s',s) + g(s)$
$g(s') = c(s',s) + g(s)$;
insert $s'$ into OPEN;

What needs to be changed in here?
Backward A* Search that computes ALL g-values

• Searching from the goal towards the start state

• g-values are cost-to-goals

ComputePath function

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

  remove $s$ with the smallest $[f(s) = g(s) + h(s)]$ from $OPEN$;

  insert $s$ into $CLOSED$;

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

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

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

      insert $s'$ into $OPEN$;

How do we make it compute ALL g-values?
Backward A* Search that computes ALL g-values

- Searching from the goal towards the start state
- **g-values are cost-to-goals**

ComputePath function

while($\text{OPEN} \neq 0$)

remove $s$ with the smallest $[f(s) = g(s) + h(s)]$ from $\text{OPEN}$;

insert $s$ into $\text{CLOSED}$;

for every predecessor $s'$ of $s$ such that $s'$ not in $\text{CLOSED}$

\[
g(s') = \begin{cases} 
    c(s', s) + g(s), & \text{if } g(s') > c(s', s) + g(s) \\
    g(s'), & \text{otherwise}
\end{cases}
\]

insert $s'$ into $\text{OPEN}$;

Run until all states get expanded!
Backward A* Search that computes ALL $g$-values

- Searching from the goal towards the start state
- **g-values are cost-to-goals**

ComputePath function

while($OPEN \neq 0$)

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

\begin{itemize}
  \item \textbf{Does it make sense to have heuristics if we are computing ALL $g$-values?}
\end{itemize}
Backward A* Search that computes ALL g-values

- Searching from the goal towards the start state
- g-values are cost-to-goals

ComputePath function
while (OPEN ≠ 0)
    remove s with the smallest \([f(s) = g(s)]\) from OPEN;
    insert s into CLOSED;
    for every predecessor s’ of s such that s’ not in CLOSED
        if \(g(s’) > c(s’,s) + g(s)\)
            \(g(s’) = c(s’,s) + g(s)\);
            insert s’ into OPEN;
**Backward A* Search that computes ALL g-values**

- Searching from the goal towards the start state.
- **g-values are cost-to-goals**
  
  ComputePath function
  
  while($OPEN \neq 0$)

  remove $s$ with the smallest $[f(s) = g(s)]$ from $OPEN$;

  insert $s$ into $CLOSED$;

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

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

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

  insert $s'$ into $OPEN$;

---

At termination, $g$-values of all states will be equal to optimal cost-to-goal values.
Backward A* Search that computes ALL g-values

- Searching from the goal towards the start state
- g-values are cost-to-goals

**ComputePath function**

while ($OPEN \neq 0$)

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

At termination, g-values of all states will be equal to optimal cost-to-goal values

Can be run on low-D problems (e.g., 2D) to compute heuristics for higher-D problems (e.g., 3+D)

---

Diagram:

- $S_{start}$ to $S_2$: $g=5$, $S_2$ to $S_1$: $g=4$, $S_1$ to $S_{goal}$: $g=2$, $S_{goal}$ to $S_3$: $g=0$
- $S_4$ to $S_3$: $g=4$, $S_3$ to $S_{goal}$: $g=1$
Examples: Heuristics via Low-D Search

- Planning in \((x,y,z,\Theta,v)\) with heuristics = 3D \((x,y,z)\) distances accounting for obstacles

[MacAllister et al., ICRA’13]

- Planning for 7DOF arm with heuristics = 3D \((x,y,z)\) distances for end-effector

[Cohen et al., IROS’13]
Weighted A*

- **Uninformed A***: expands states in the order of $g$ values
- **A***: expands states in the order of $f = g + h$ values
- **Weighted A***: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1 = \text{bias towards states that are closer to goal}$

---

**Diagram:**

- $S_{\text{start}}$ to $S_{\text{goal}}$
- $g(s)$: the cost of a shortest path from $s_{\text{start}}$ to $s$ found so far
- $h(s)$: an (under) estimate of the cost of a shortest path from $s$ to $s_{\text{goal}}$
Weighted A*

- **Uninformed A**: expands states in the order of g values

What are the states expanded?
Weighted A*

- A*: expands states in the order of $f = g + h$ values

What are the states expanded?
Weighted A*

• A*: expands states in the order of $f = g+h$ values

for large problems this results in A* quickly running out of memory (memory: $O(n)$)
Weighted A*

- Weighted A*: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

**What states are expanded?**

**Key to finding solution fast:**

*shallow minima for $h(s) - h^*(s)$ function*
Weighted A*

- **Weighted A***: expands states in the order of \( f = g + \epsilon h \) values, \( \epsilon > 1 \) = bias towards states that are closer to goal

**What states are expanded?**

*No one knows. Topic for research.*

**Key to finding solution fast:** shallow minima for \( h(s) - h^*(s) \) function
Weighted A*

- **Weighted A* Search:**
  - trades off optimality for speed
  - $\varepsilon$-suboptimal:
    \[
    cost(solution) \leq \varepsilon \cdot cost(optimal \ solution)
    \]
  - in many domains, it has been shown to be orders of magnitude faster than A*
  - research becomes to develop a heuristic function that has shallow local minima
Few Properties of Heuristic Functions

• Useful properties to know:

- \( h_1(s), h_2(s) \) – consistent, then:
  \[ h(s) = \max(h_1(s), h_2(s)) \] – consistent

- if A* uses \( \varepsilon \)-consistent heuristics:
  \[ h(s_{\text{goal}}) = 0 \text{ and } h(s) \leq \varepsilon \ c(s, \text{succ}(s)) + h(\text{succ}(s)) \text{ for all } s \neq s_{\text{goal}}, \]
  then A* is \( \varepsilon \)-suboptimal:
  \[ \text{cost(solution)} \leq \varepsilon \ \text{cost(optimal solution)} \]

- weighted A* is A* with \( \varepsilon \)-consistent heuristics

- \( h_1(s), h_2(s) \) – consistent, then:
  \[ h(s) = h_1(s) + h_2(s) \] – \( \varepsilon \)-consistent

\[ \text{Proof?} \]
What You Should Know…

• Common heuristic functions for X-connected grids
  – Euclidean distance, Manhattan distance, Diagonal distance, etc.

• Be able to design and implement heuristics for high-D planning (e.g., heuristics computed by low-d search)

• Weighted A* and its properties

• Backward A*

• How to combine heuristics, properties, $\varepsilon$-consistent heuristics