16-350
Planning Techniques for Robotics

Search Algorithms: Heuristics, Weighted A Search*

Maxim Likhachev

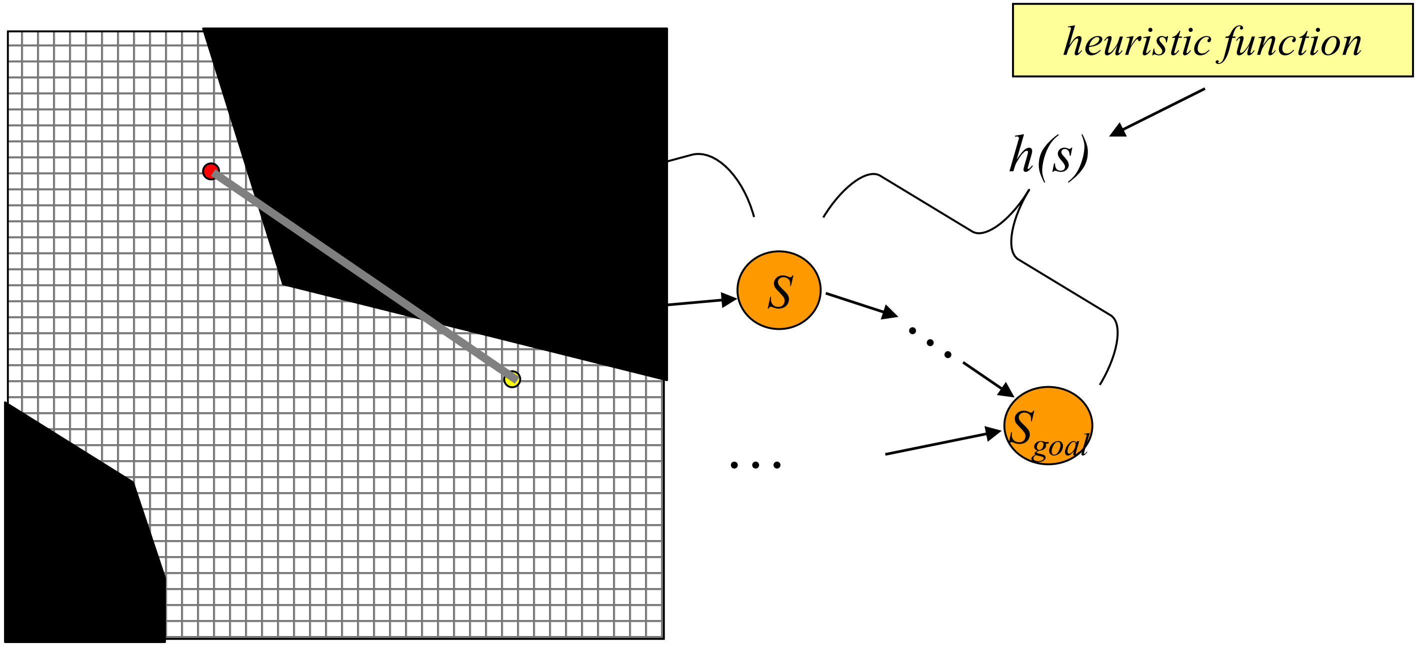
Robotics Institute

Carnegie Mellon University

A* Search

- Computes optimal g-values for relevant states

at any point of time:



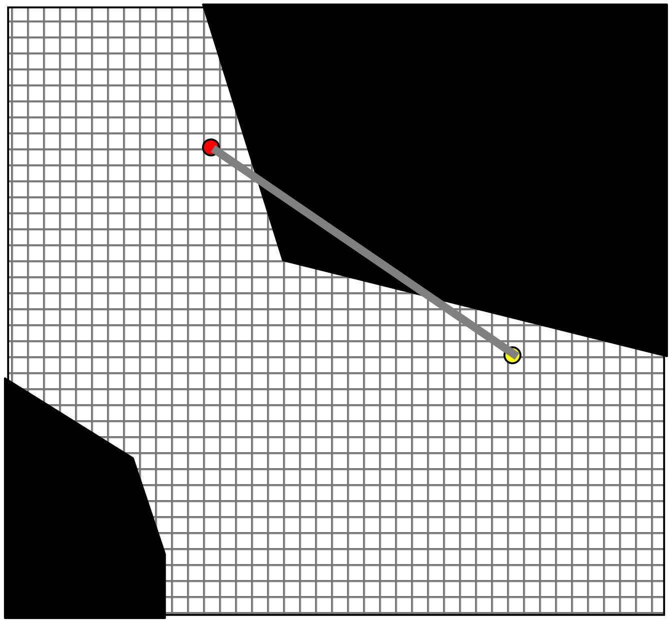
one popular heuristic function – Euclidean distance

Heuristics

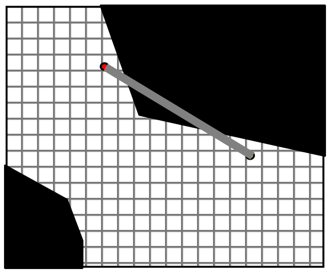
minimal cost from s to s_{goal}

- Heuristic function must be:
  - admissible: for every state s, $h(s) \leq c^*(s,s_{goal})$
  - consistent (satisfy triangle inequality):

    $h(s_{goal},s_{goal}) = 0$ *and for every* $s \neq s_{goal}$, $h(s) \leq c(s,succ(s)) + h(succ(s))$
  - 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???

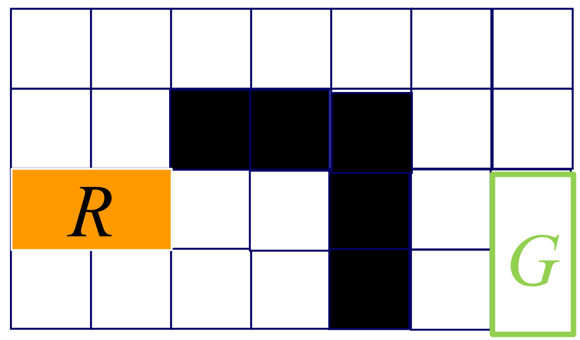
Which heuristics are admissible for
4-connected grid?
8-connected grid?

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

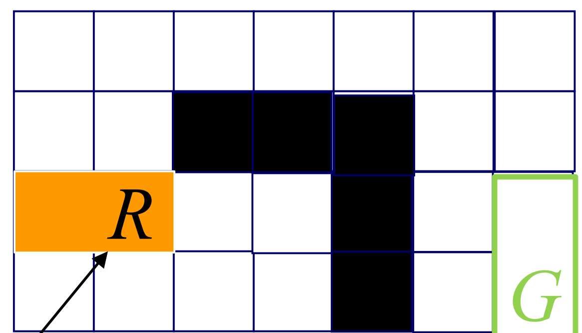


Heuristics

- For planning problems higher than 2D

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

Non-circular robot



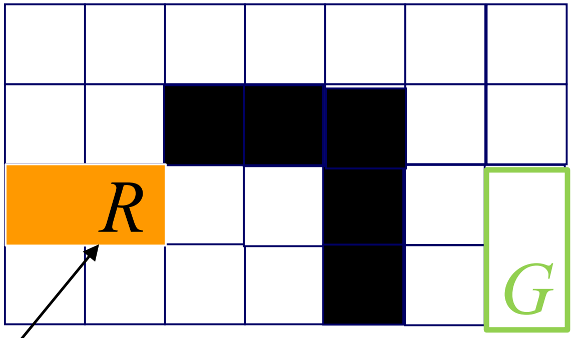
Grid-based representation for planning:
x,y,Θ for some reference point on the robot
x,y are on 8-connected grid
Θ – discretized into 8 angles

Heuristics

- For planning problems higher than 2D

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

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

How many states?

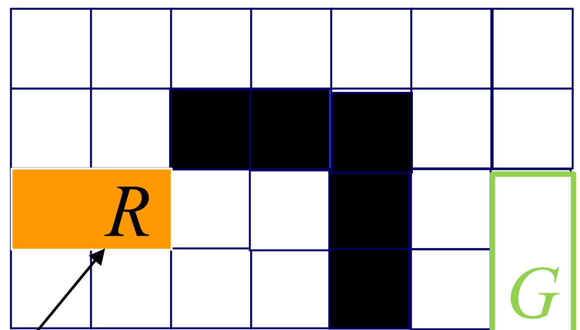
Heuristics

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,Θ for some reference point on the robot
x,y are on 8-connected grid
Θ – discretized into 8 angles

Heuristics

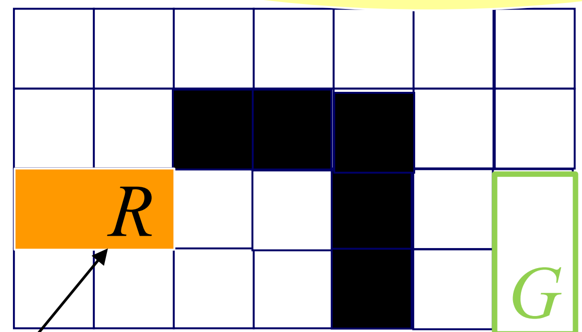
Any ideas for heuristics that estimate cost-to-goal better ?

- For planning problems higher

Example:
consider planning for a non-c
direction (omnidirectional)

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

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

Heuristi

How can we compute them?

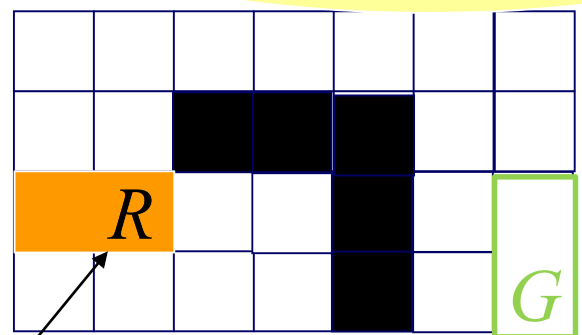
Are these admissible?

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

- For planning problems hig

Example:
consider planning for a non-c
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

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;

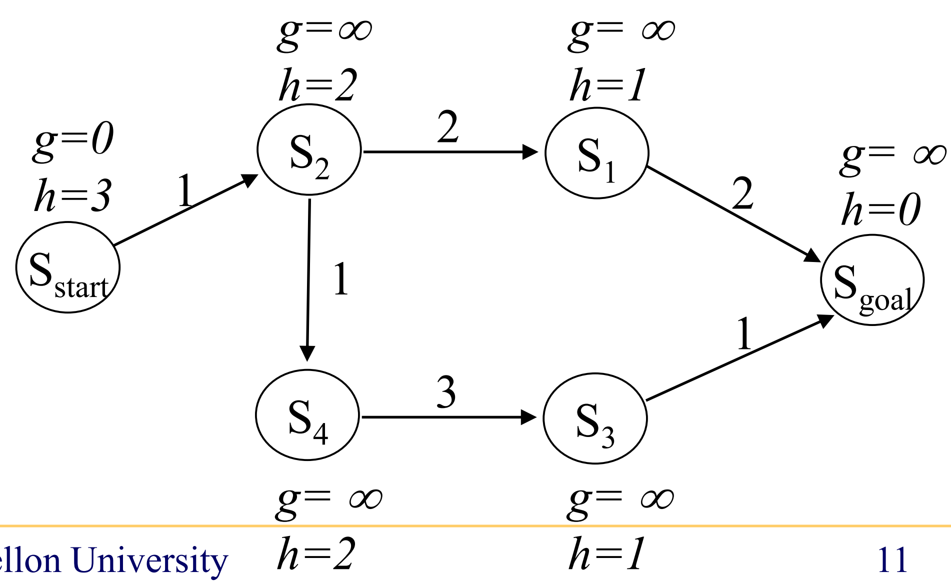
What needs to be changed?

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;



Backward A* Search

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

Main function

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

ComputePath();

publish solution;

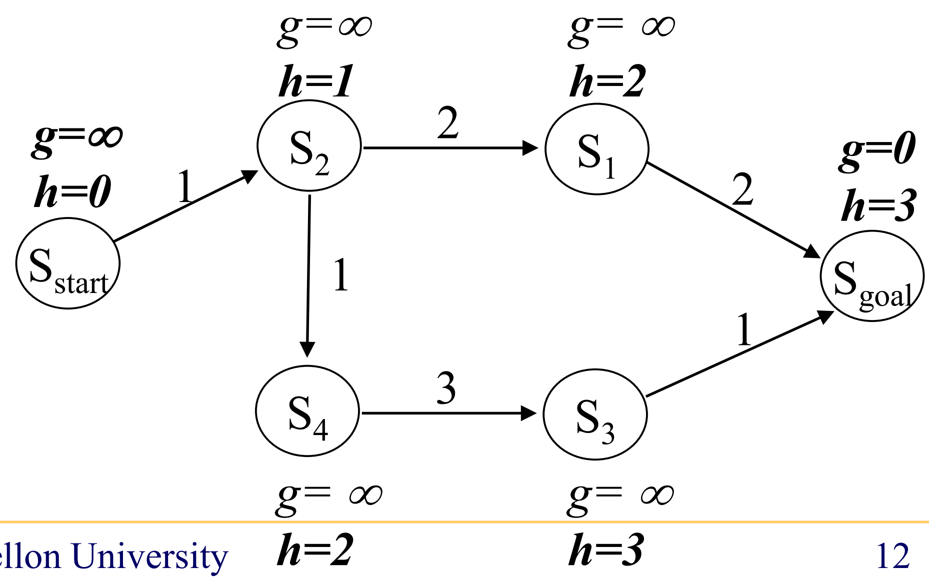
What needs to be changed?

ComputePath function

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

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

  expand s;



Backward A* Search

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

What needs to be changed in here?

ComputePath function

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

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

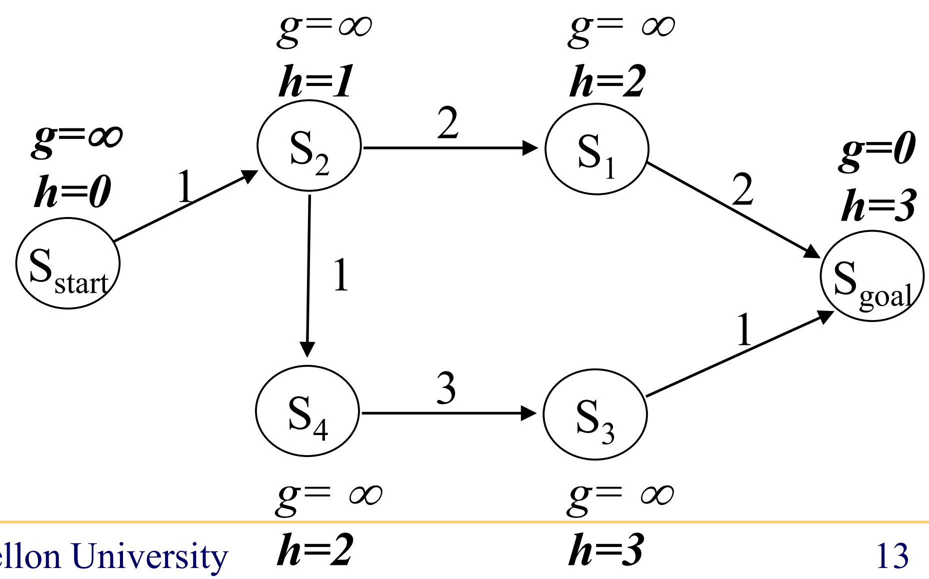
  insert s into $CLOSED$;

  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 $OPEN$;



Backward A* Search

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

What needs to be changed in here?

ComputePath function

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

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

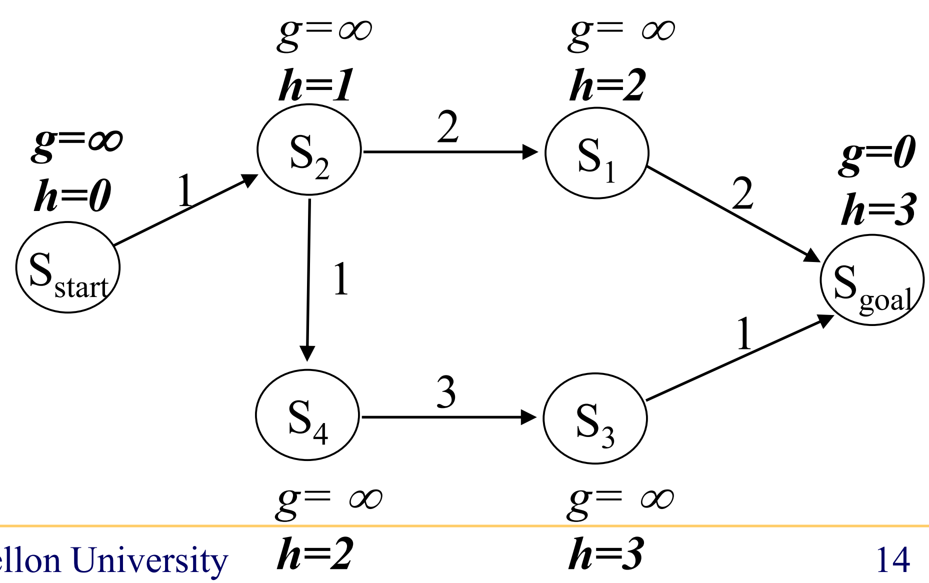
  insert s into $CLOSED$;

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

    if $g(s') > \mathbf{c(s',s) + g(s)}$

      $g(s') = \mathbf{c(s',s) + g(s)}$;

      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**

*How do we make it compute **ALL** g-values?*

ComputePath function

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

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

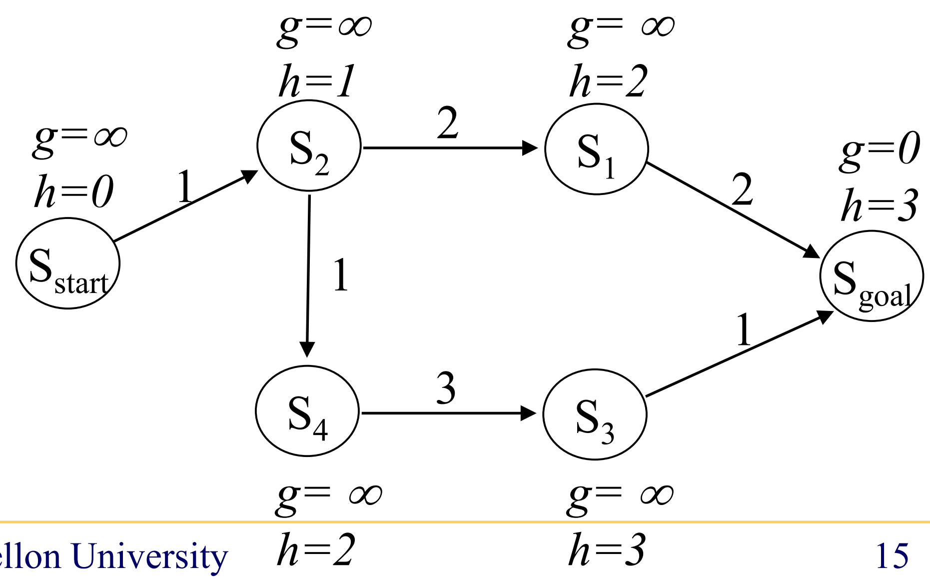
insert s into $CLOSED$;

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 $OPEN$;



Backward A* Search that computes **ALL** g-values

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

Run until all states get expanded!

ComputePath function

while(***OPEN* ≠ 0**)

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

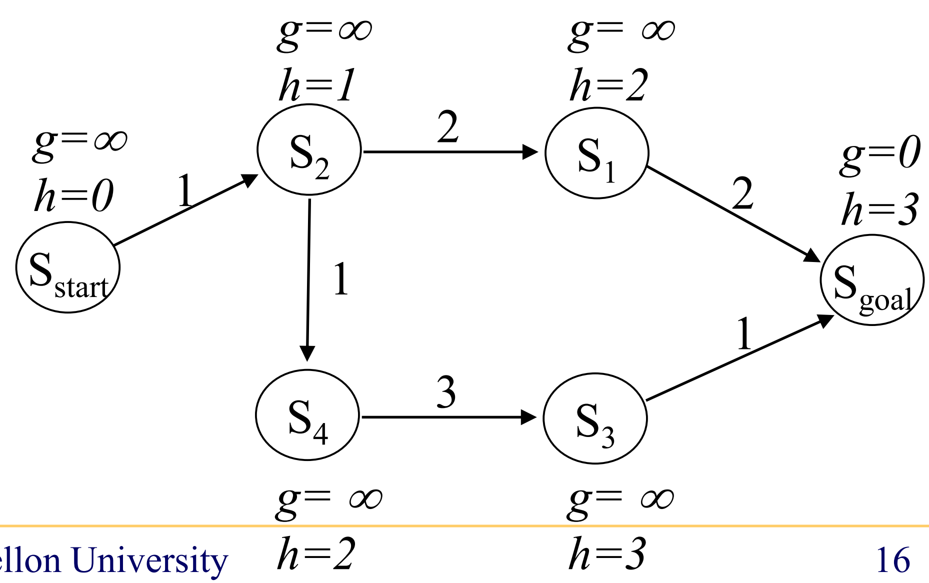
  insert *s* into *CLOSED*;

  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*;



Backward A* Search that computes **ALL** g-values

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

Does it make sense to have heuristics if we are computing ALL g-values?

ComputePath function

while($OPEN \neq 0$)

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

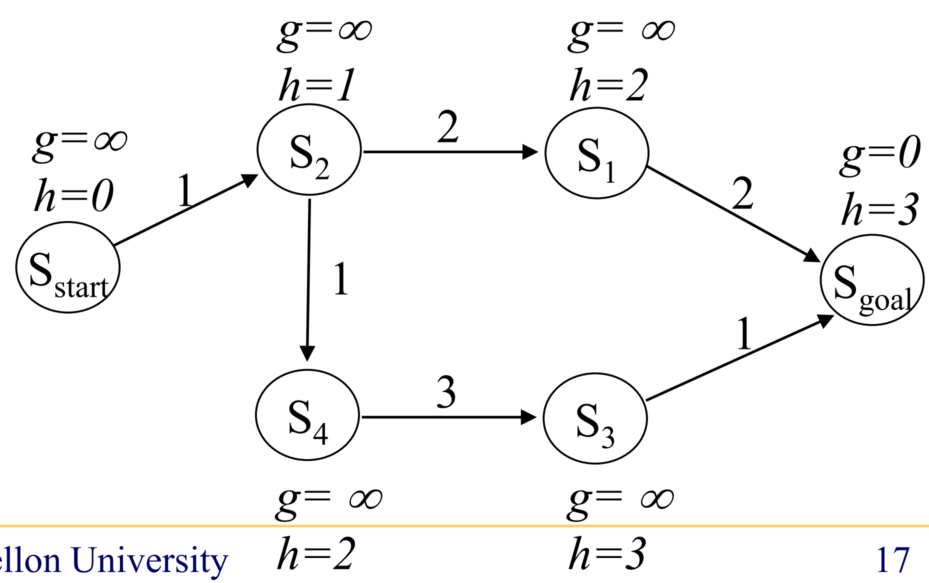
insert s into $CLOSED$;

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 $OPEN$;



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*;

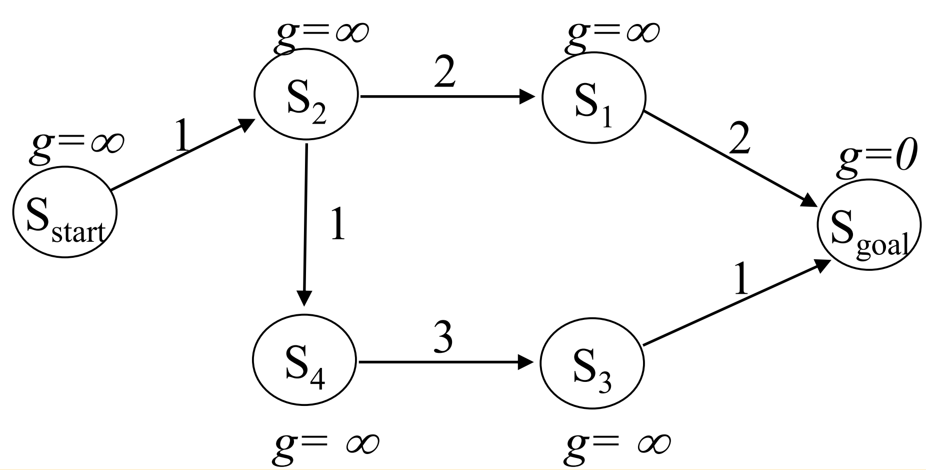
insert s into *CLOSED*;

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*;



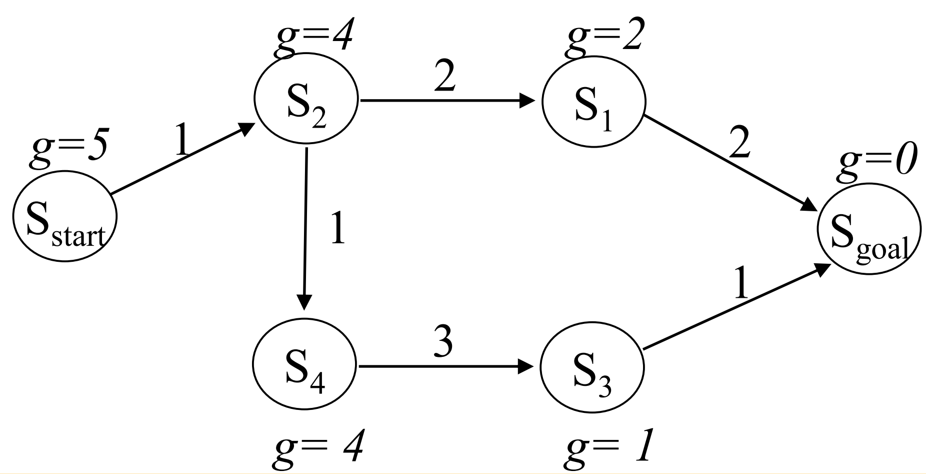
Backward A* Search that computes **ALL** g-values

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

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

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 $OPEN$;



Backward A* Search that computes **ALL** g-values

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

ComputePath function

while($OPEN \neq 0$)

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

insert s into $CLOSED$;

for every predecessor s' of s

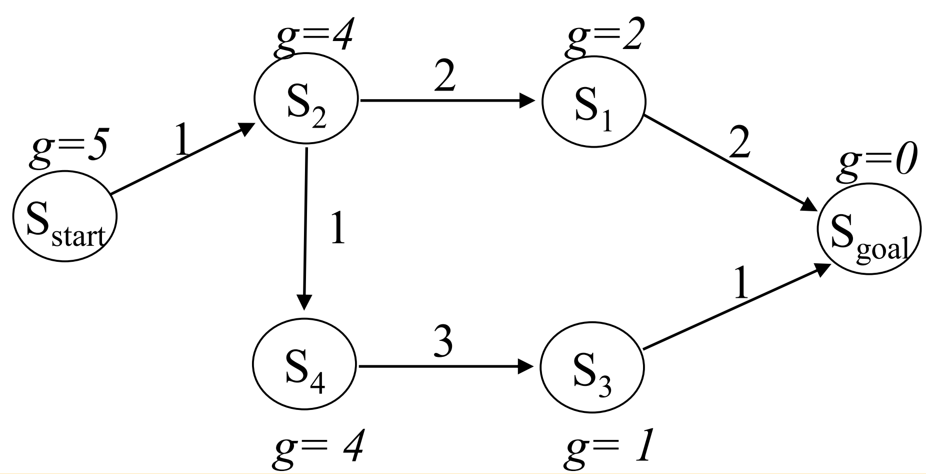
if $g(s') > c(s',s) + g(s)$

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

insert s' into $OPEN$;

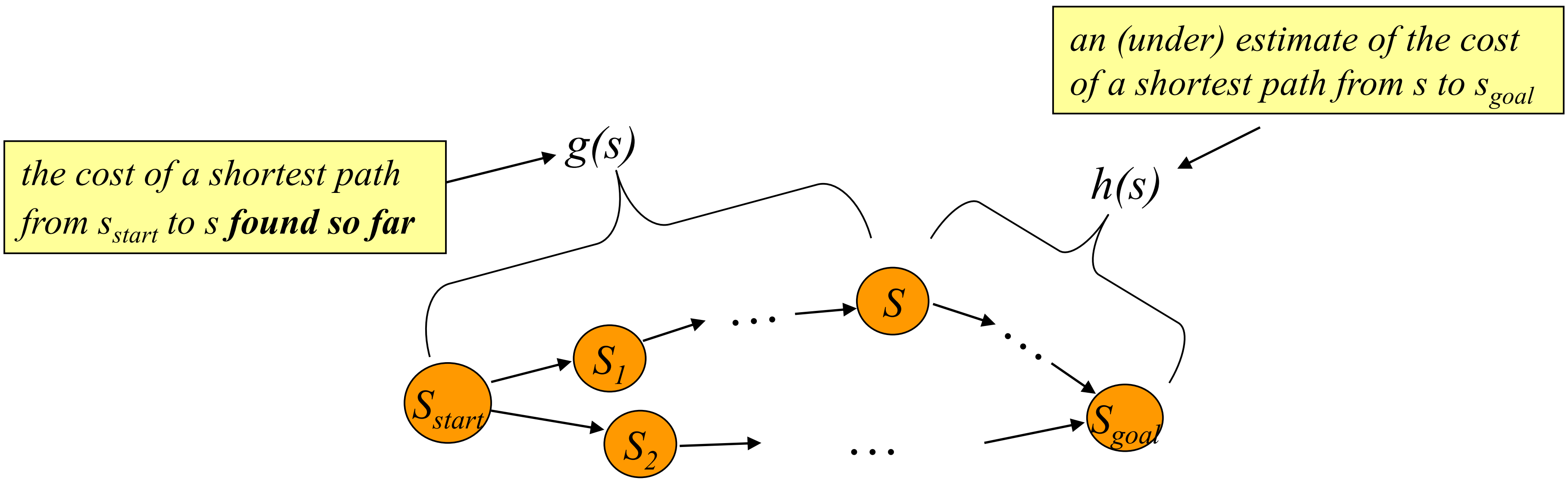
At termination,
g-values of all states
will be equal to
optimal cost-to-goal values

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



Weighted A*

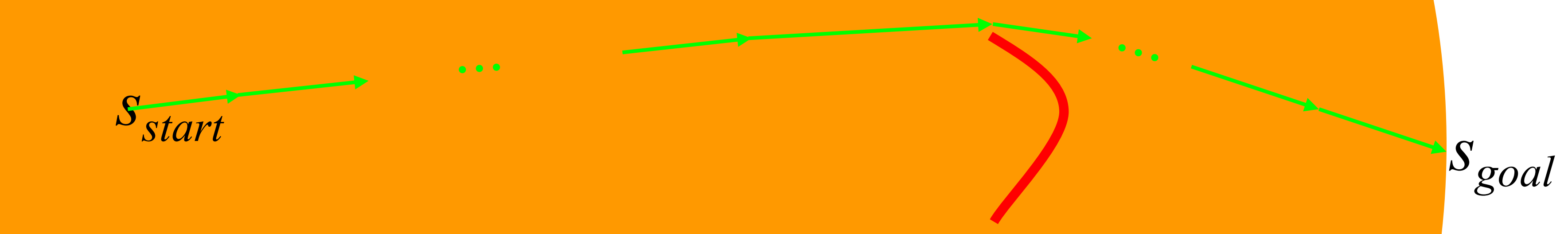
- **Uninformed A*:** expands states in the order of $\boldsymbol{g}$ values
- **A*:** expands states in the order of $\boldsymbol{f = g+h}$ values
- **Weighted A*:** expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal



Weighted A*

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

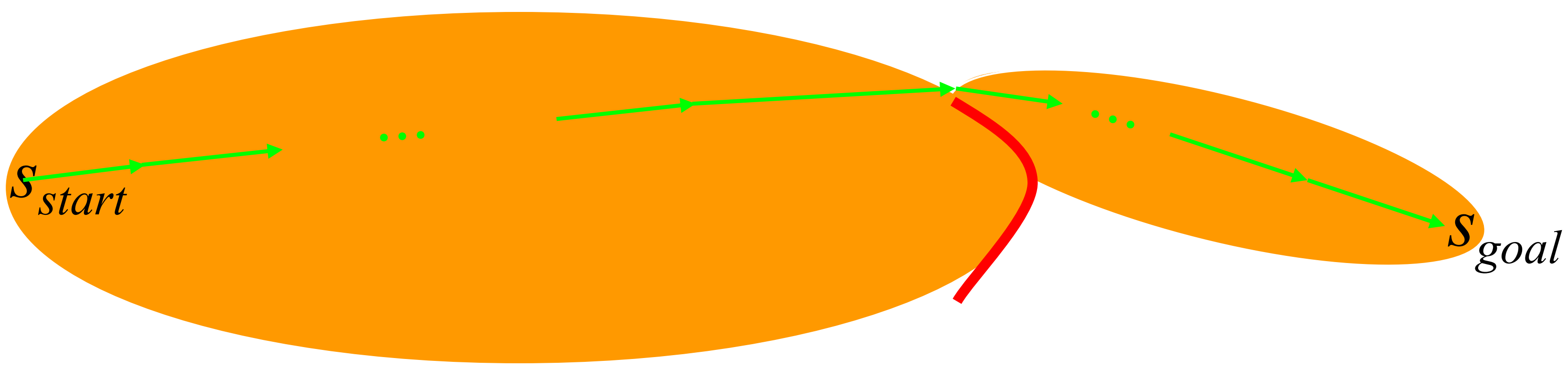
What are the states expanded?



Weighted A*

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

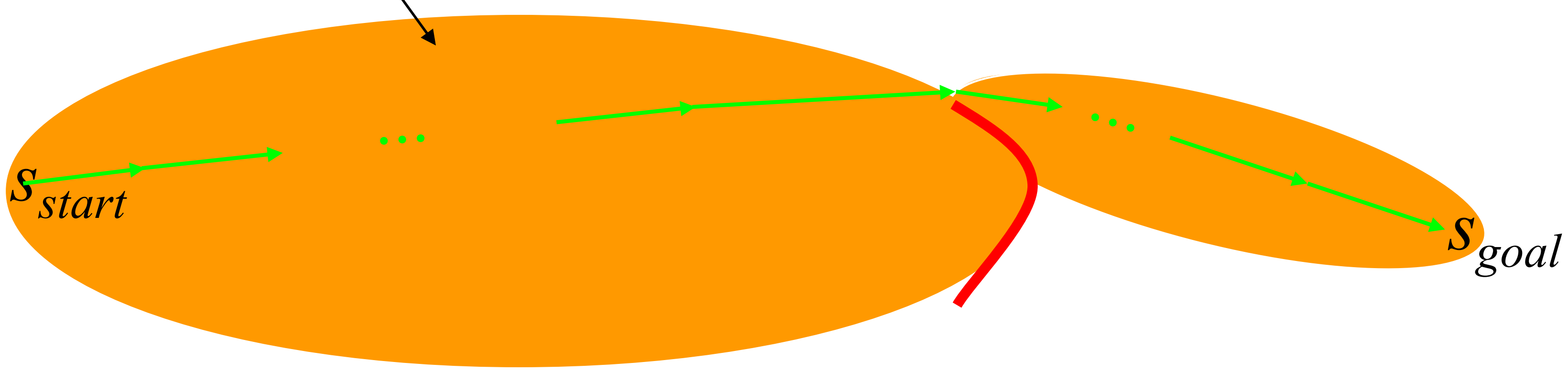
What are the states expanded?



Weighted A*

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

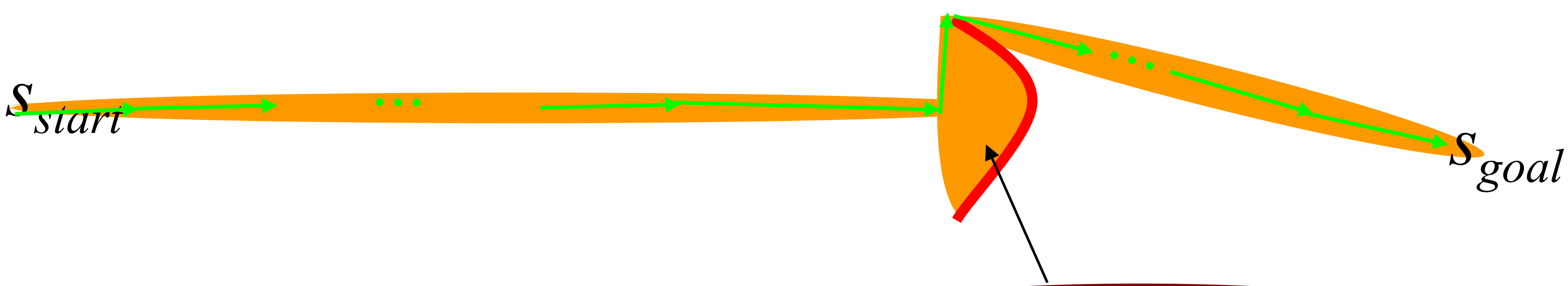
for large problems this results in A quickly running out of memory (memory: O(n))*



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?



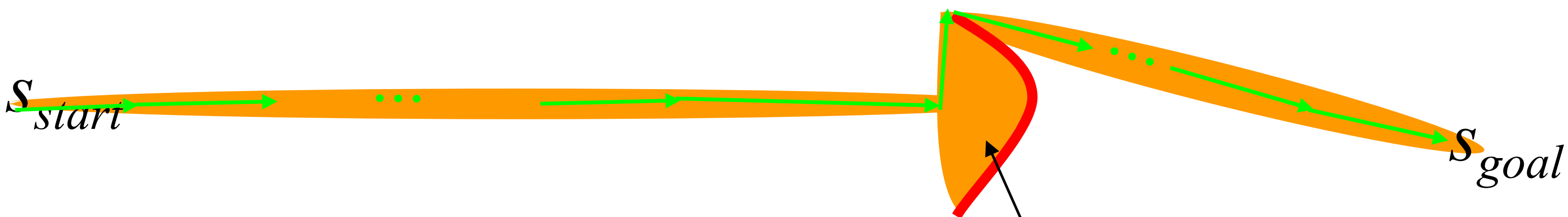
key to finding solution fast:
shallow minima for h(s)-h(s) function*

Weighted A*

- **Weighted A*:** expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = 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
  - ε-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 ε-consistent heuristics:

    $h(s_{goal}) = 0$ *and* $h(s) \leq \varepsilon\, c(s,succ(s)) + h(succ(s)$ *for all* $s \neq s_{goal}$,

    then A* is ε-suboptimal:

    $cost(solution) \leq \varepsilon\, cost(optimal\ solution)$

  - weighted A* is A* with ε-consistent heuristics

    Proof?

  - $h_1(s)$, $h_2(s)$ – consistent, then:

    $h(s) = h_1(s)+h_2(s)$ – ε-consistent

    What is ε? 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, ε-consistent heuristics