# *16-350*
# *Planning Techniques for Robotics*

# *Search Algorithms: A* Search*

*Maxim Likhachev*

*Robotics Institute*

*Carnegie Mellon University*

# Uninformed A* Search

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

**Main function**

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

ComputePath();

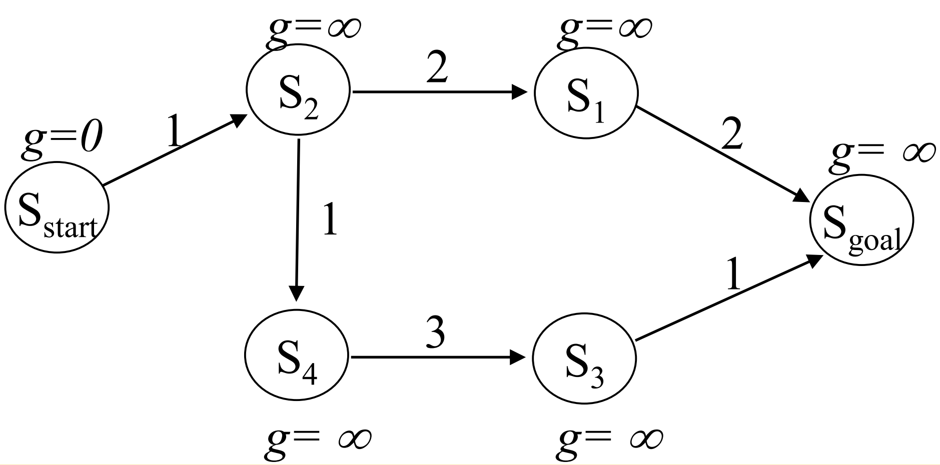
publish solution; //compute least-cost path using $g$-values

**ComputePath function**

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

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

  expand $s$;

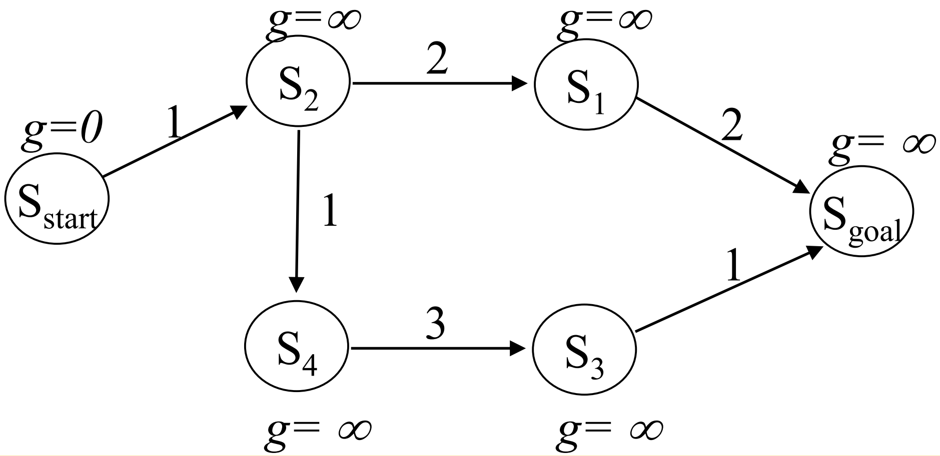


# Uninformed A* Search

- 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*;



# 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*;

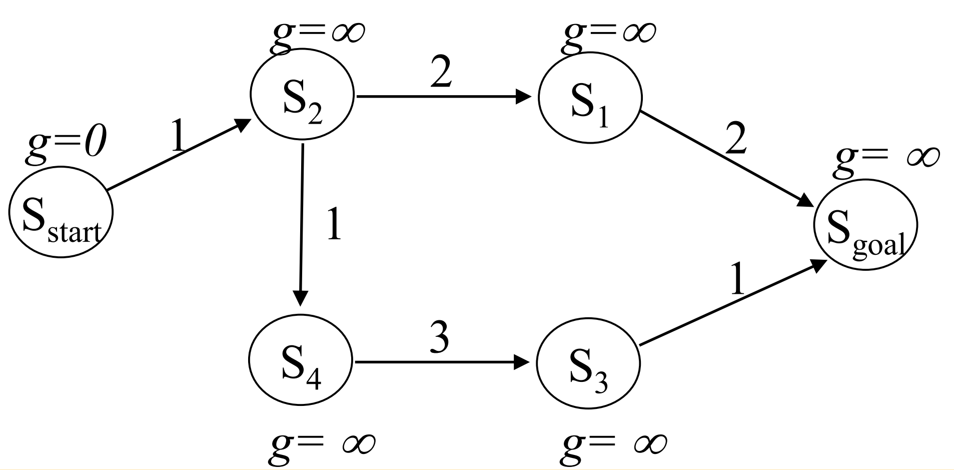
for every successor $s'$ of $s$ such that $s'$ not in *CLOSED*

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

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

insert $s'$ into *OPEN*;

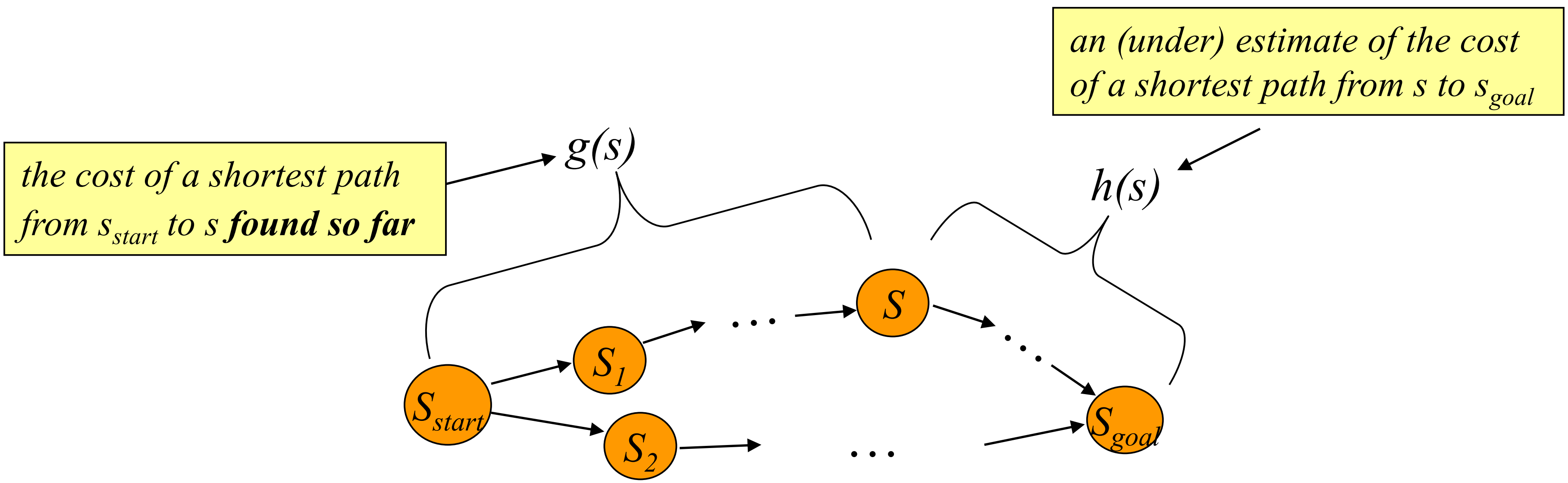
*clarification: updates g(s') if s' is already in OPEN*



# A* Search [Hart, Nillson, Raphael, '68]

- Computes optimal g-values for relevant states

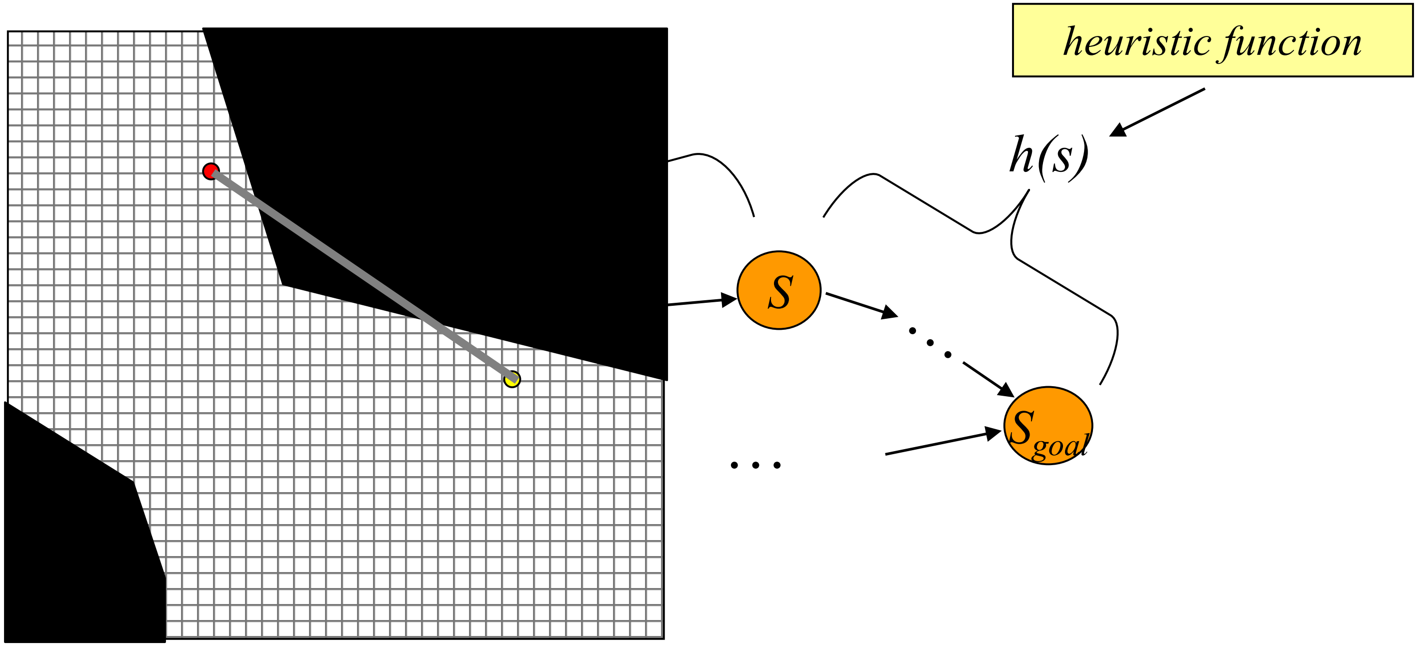
at any point of time:



# A* Search [Hart, Nillson, Raphael, '68]

- Computes optimal g-values for relevant states

at any point of time:



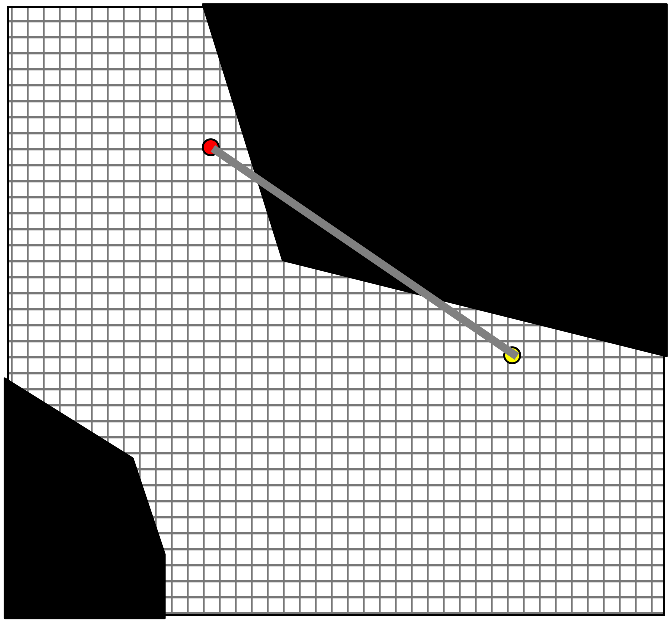

*one popular heuristic function – Euclidean distance*

# A* Search [Hart, Nillson, Raphael, ‘68]

*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



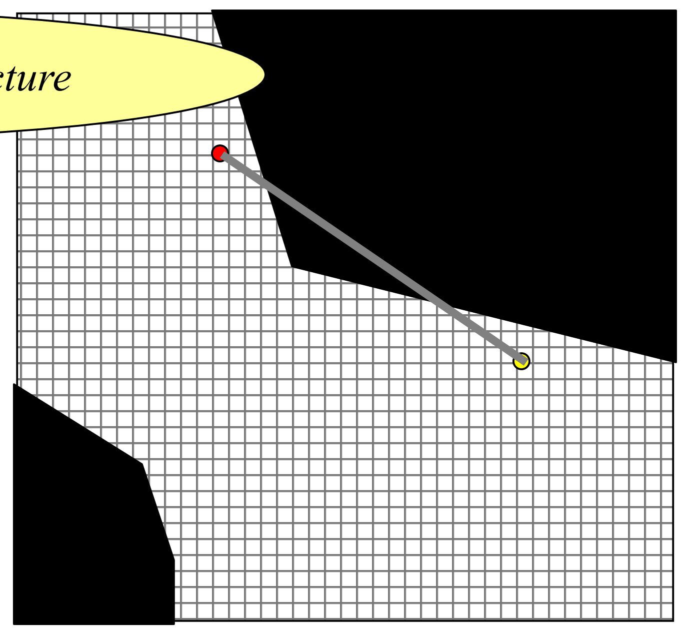
# A* Search [Hart, Nillson, Raphael, '68]

*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). *Why 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

*More on this in the later lecture*



# A*: Uninformed vs. Informed Search

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

# A* Search

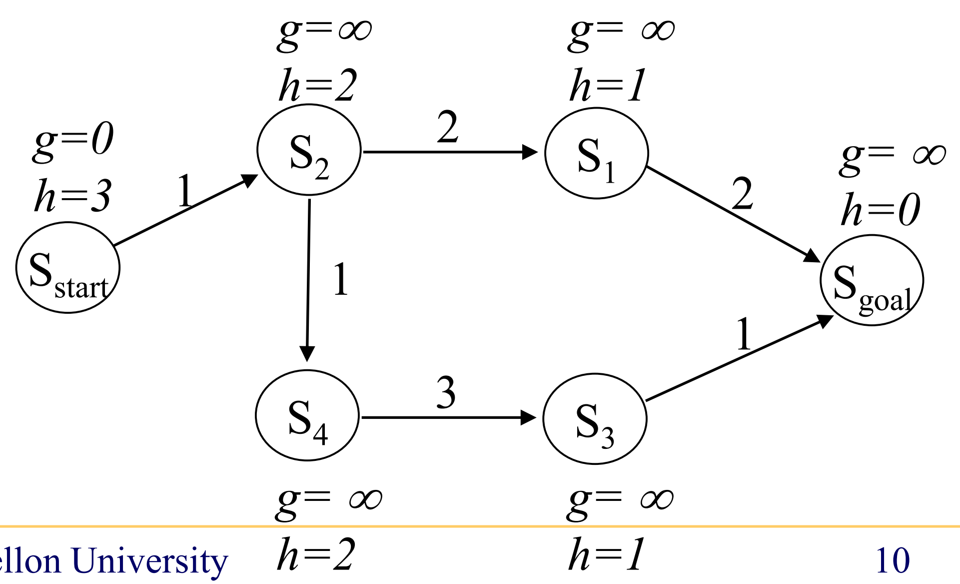
- Computes optimal g-values for relevant states

**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 $s$;



# A* Search

- Computes optimal g-values for relevant states

**ComputePath function**

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

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

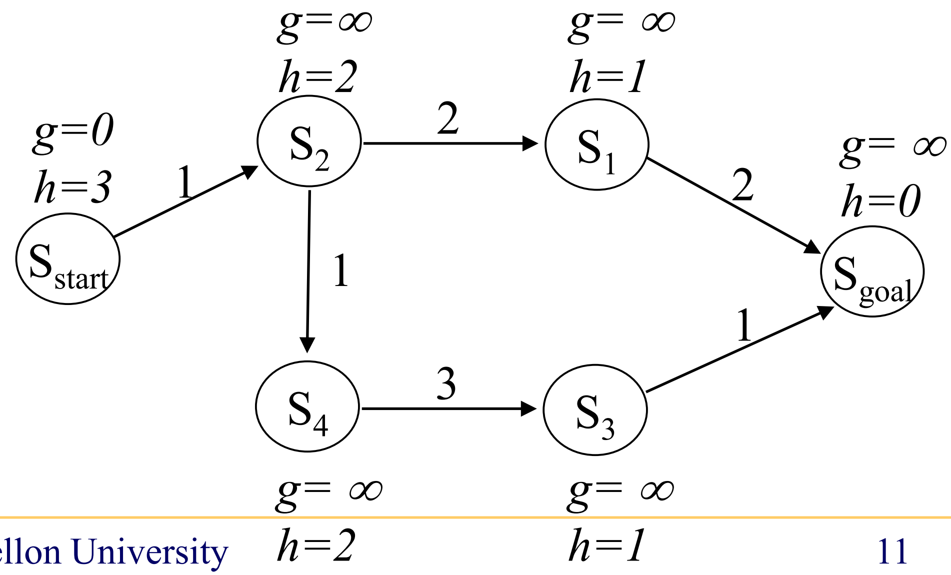
insert $s$ into $CLOSED$;

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 $OPEN$;



# A* Search

- Computes optimal g-values for relevant states

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

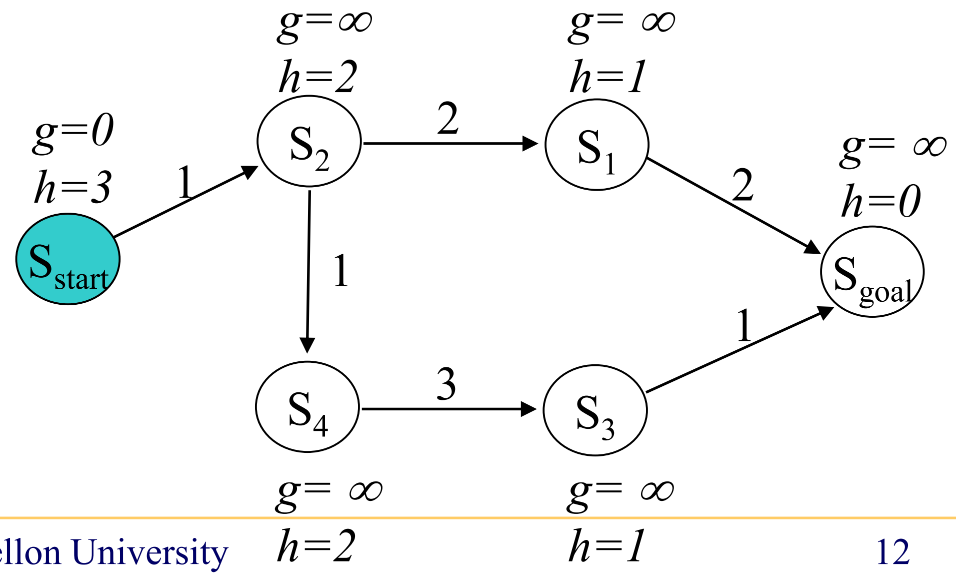
$g(s') = g(s) + c(s,s')$;

insert $s'$ into $OPEN$;

$CLOSED = \{\}$

$OPEN = \{s_{start}\}$

*next state to expand:* $s_{start}$



# A* Search

- Computes optimal g-values for relevant states

**ComputePath function**

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

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

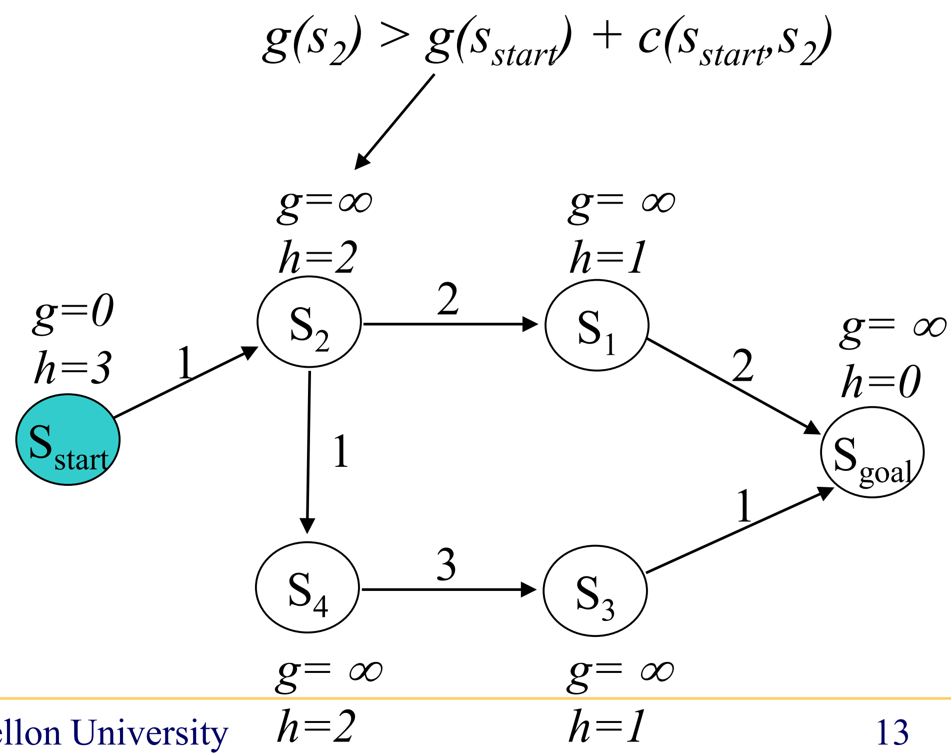
insert $s$ into $CLOSED$;

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 $OPEN$;




$CLOSED = \{\}$

$OPEN = \{s_{start}\}$

*next state to expand:* $s_{start}$

# A* Search

- Computes optimal g-values for relevant states

**ComputePath function**

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

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

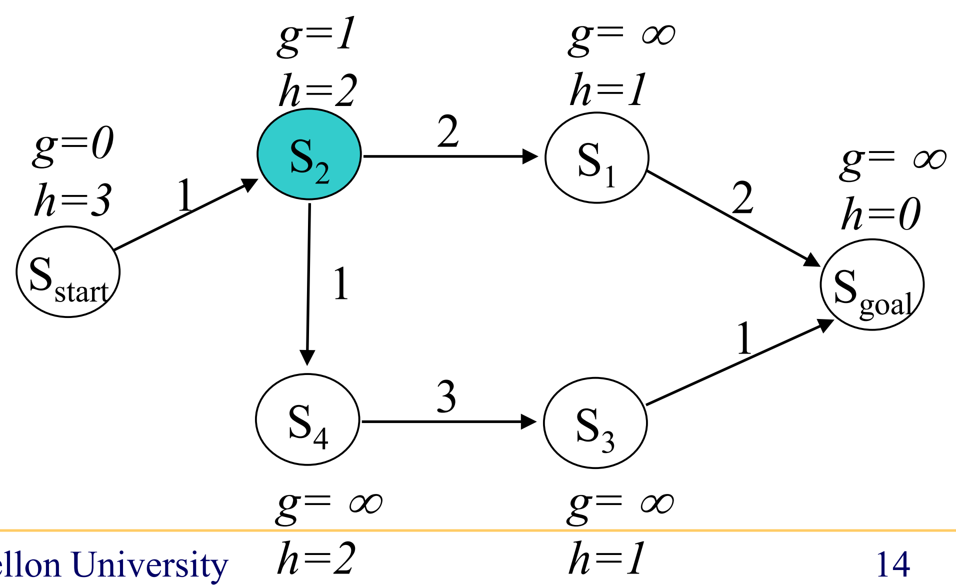
insert $s$ into $CLOSED$;

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 $OPEN$;



# A* Search

- Computes optimal g-values for relevant states

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

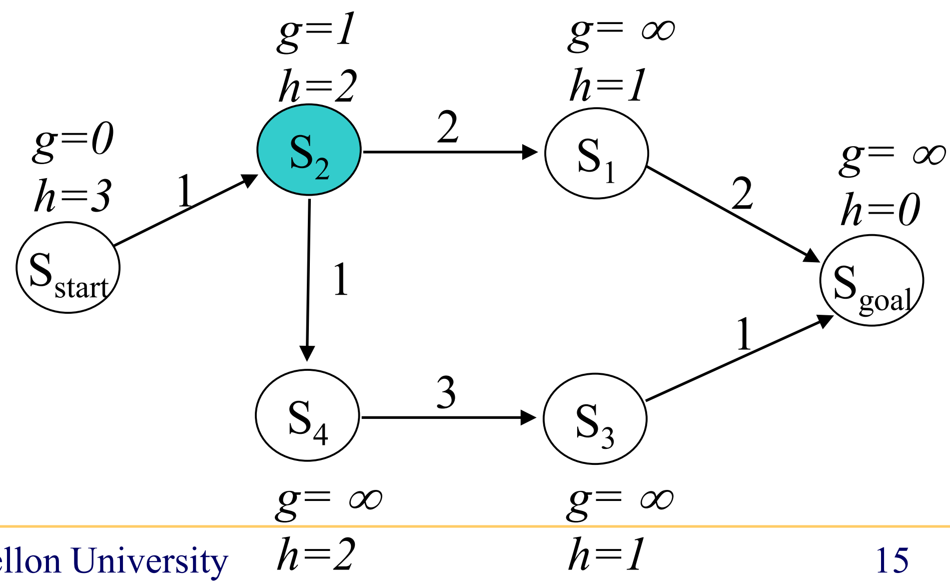
$g(s') = g(s) + c(s,s');$

insert $s'$ into $OPEN$;

$CLOSED = \{s_{start}\}$

$OPEN = \{s_2\}$

*next state to expand:* $s_2$



# A* Search

- Computes optimal g-values for relevant states

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

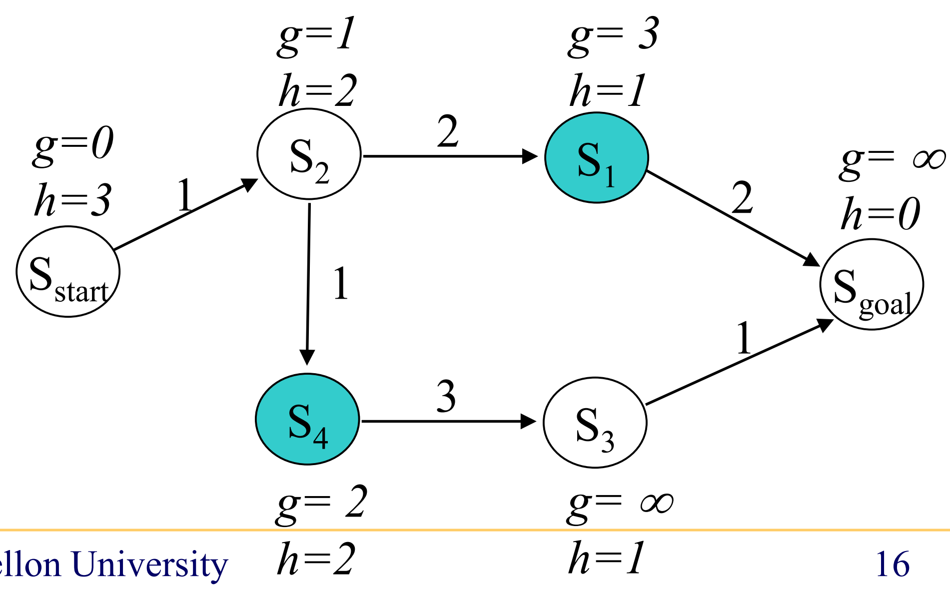
$g(s') = g(s) + c(s,s')$;

insert $s'$ into $OPEN$;

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

$OPEN = \{s_1, s_4\}$

*next state to expand:* $s_1$



# A* Search

- Computes optimal g-values for relevant states

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

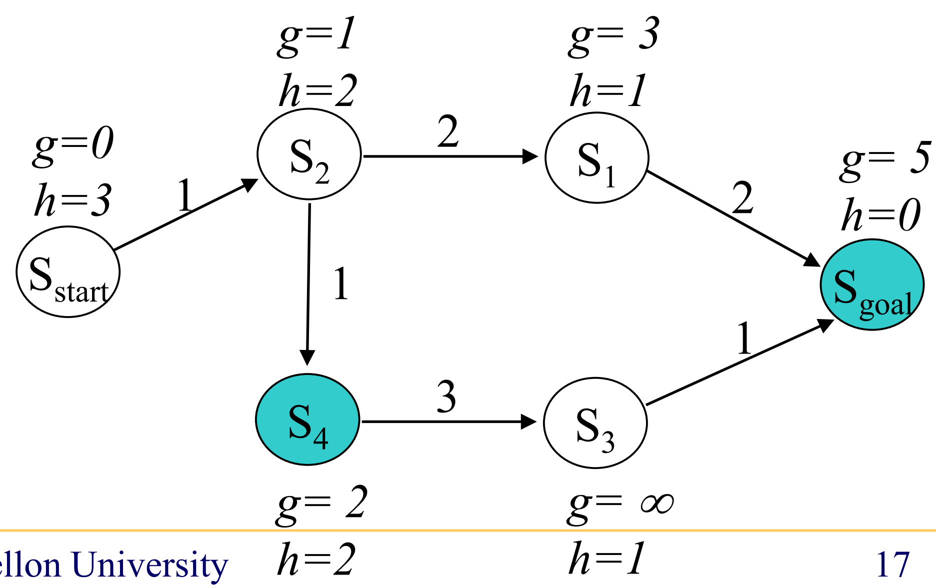
      $g(s') = g(s) + c(s,s')$;

      insert $s'$ into $OPEN$;

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

$OPEN = \{s_4, s_{goal}\}$

*next state to expand:* $s_4$



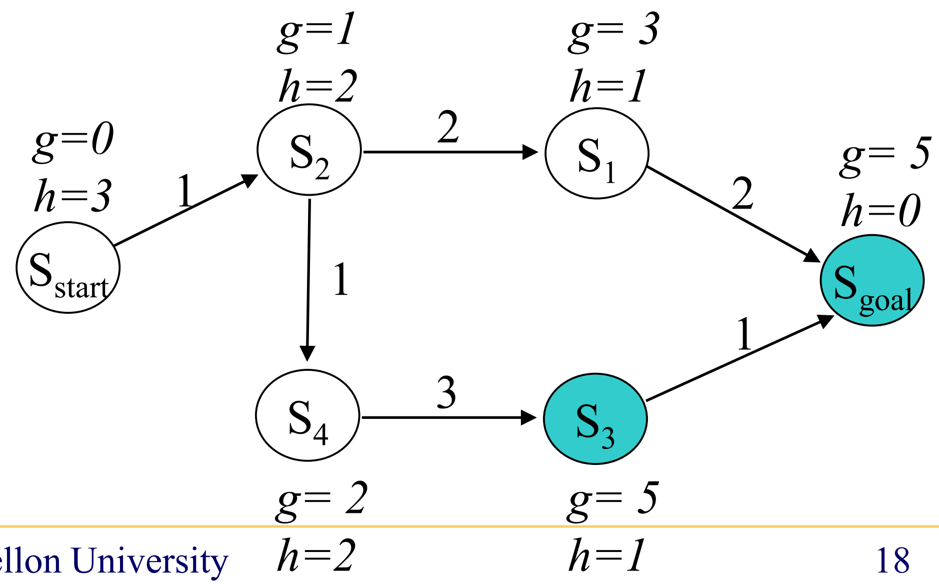
# A* Search

- Computes optimal g-values for relevant states

**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 $OPEN$;

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



- Computes optimal g-values for relevant states

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

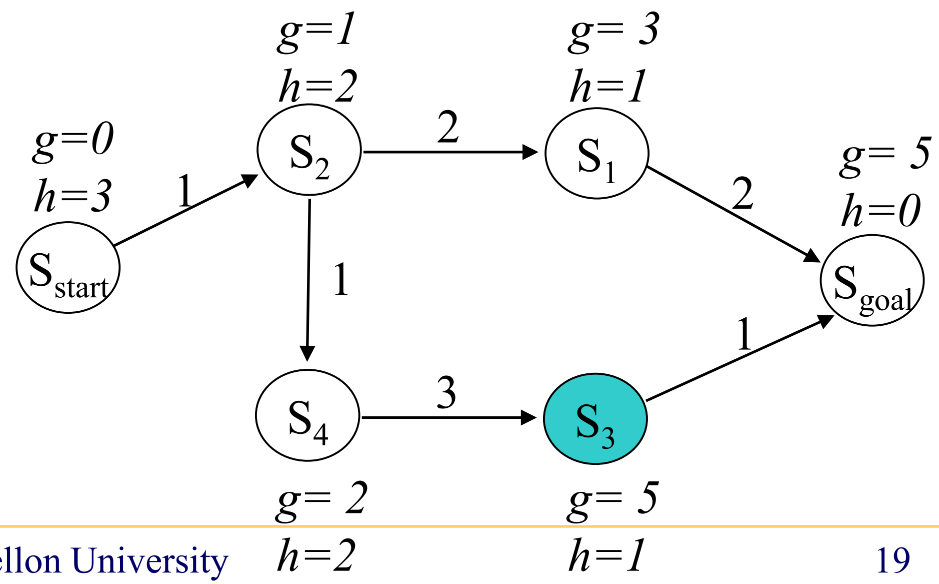
$g(s') = g(s) + c(s,s')$;

insert $s'$ into $OPEN$;

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

$OPEN = \{s_3\}$

*done*



# A* Search

- Computes optimal g-values for relevant states

**ComputePath function**

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

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

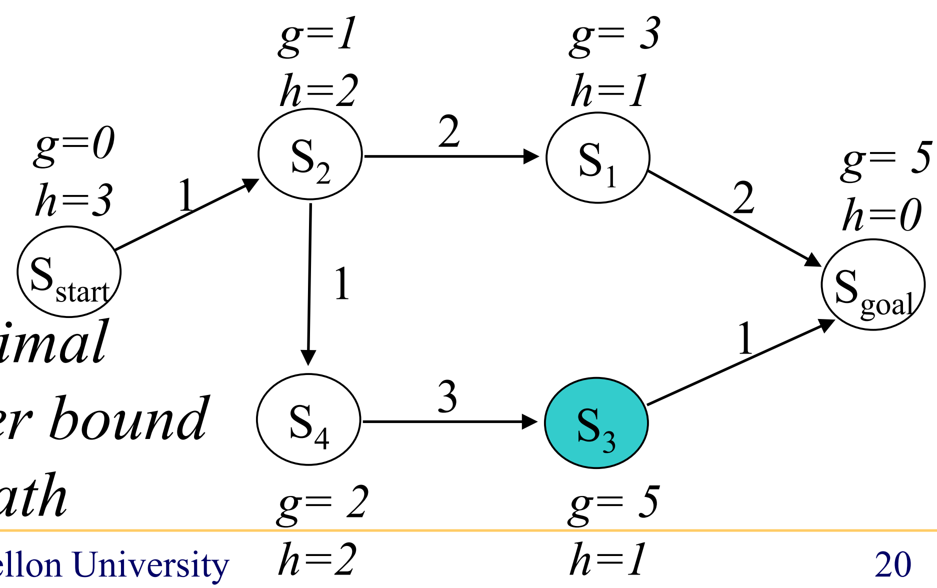
insert $s$ into $CLOSED$;

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 $OPEN$;




*for every expanded state g(s) is optimal*

*for every other state g(s) is an upper bound*

*we can now compute a least-cost path*

# A* Search

- Computes optimal g-values for relevant states

**ComputePath function**

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

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

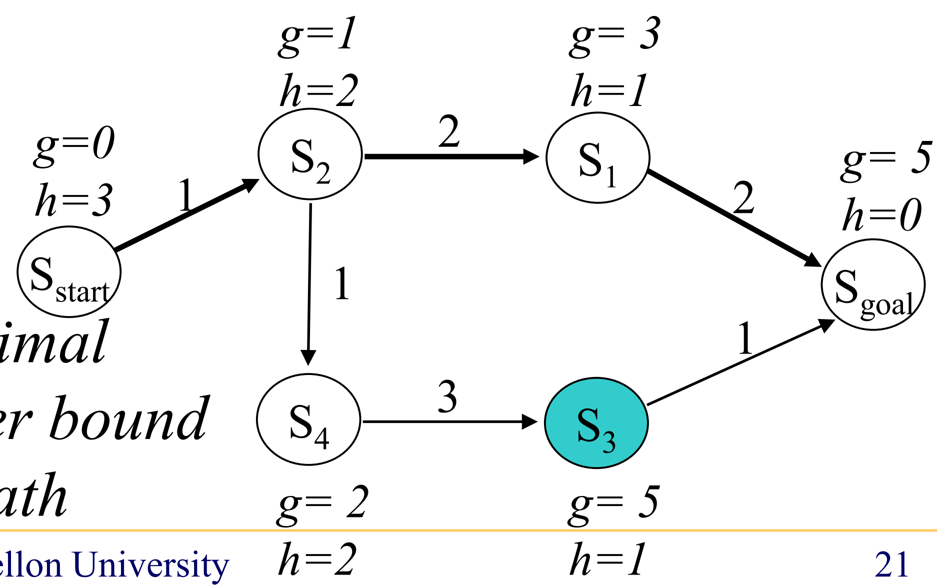
insert $s$ into $CLOSED$;

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 $OPEN$;




*for every expanded state g(s) is optimal*

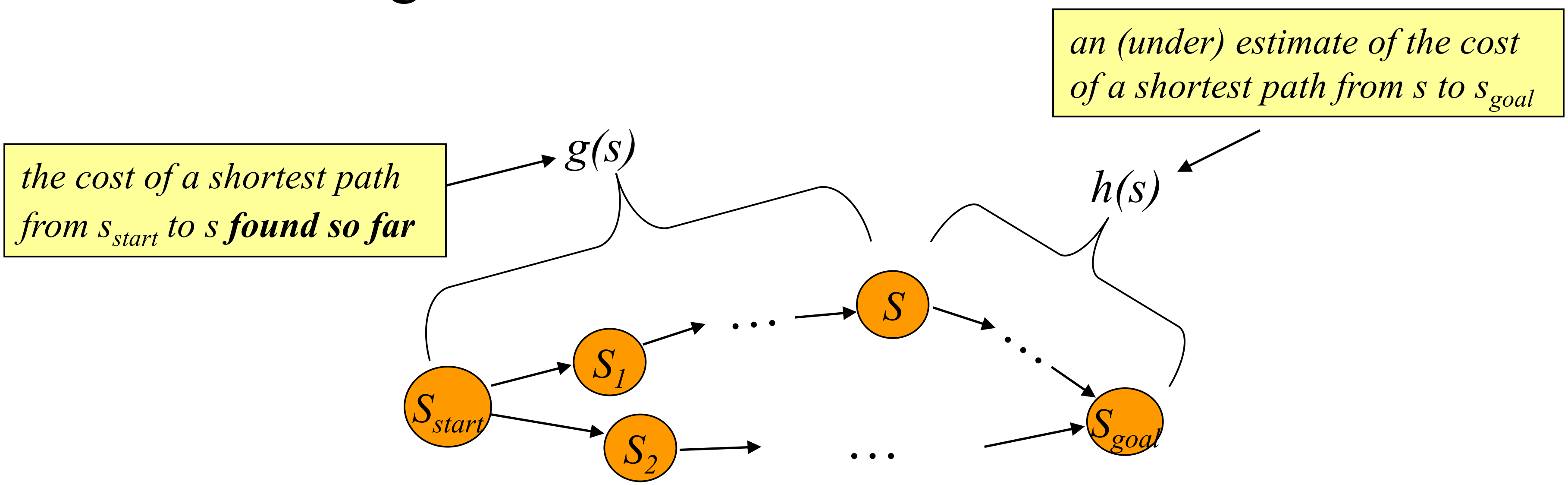
*for every other state g(s) is an upper bound*

*we can now compute a least-cost path*

# A*: Uninformed vs. Informed Search

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

- Intuitively: *f(s)* – estimate of the cost of a least cost path from start to goal via state *s*



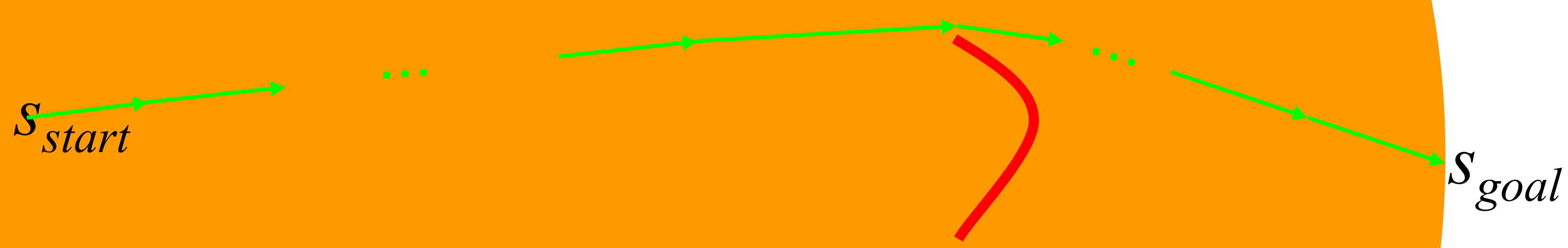
# Informed Search

+*h* values

r of *g* values

ost path

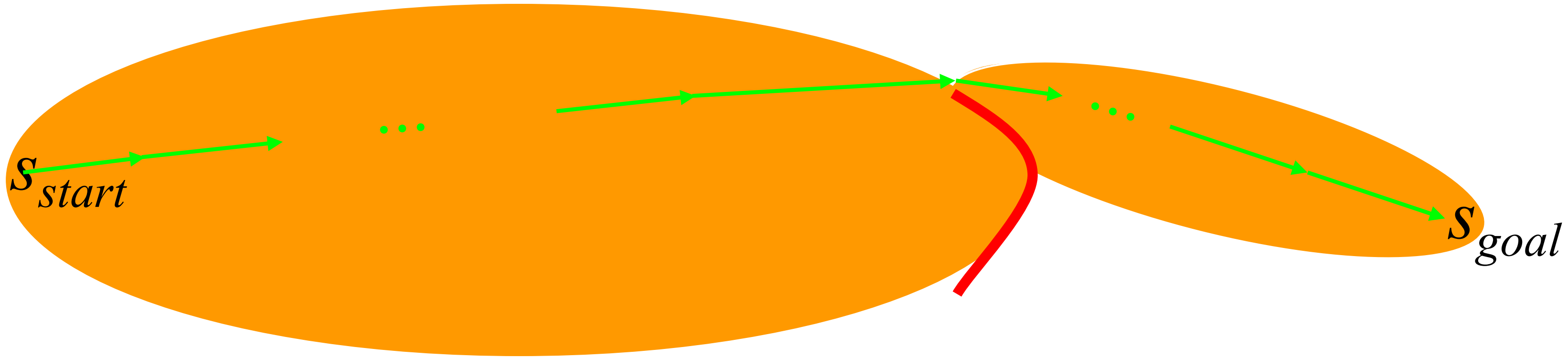
*Uninformed A**



# A*: Uninformed vs. Informed Search

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

- Intuitively: $f(s)$ – estimate of the cost of a least cost path from start to goal via state $s$



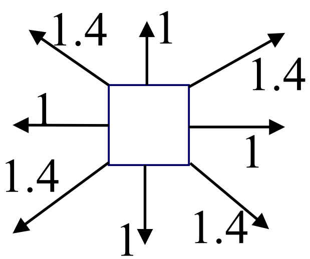

*A* with Heuristics=Euclidean Distance*

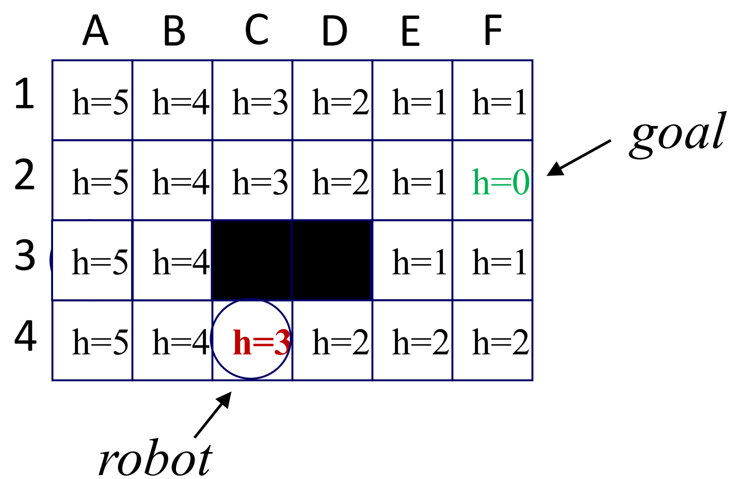
# A* Search

- Example on a Grid-based Graph:

$$h(cell <x,y>) = max(|x-x_{goal}|, |y-y_{goal}|)$$

*8-connected grid*



# A* Search: Proofs

**Theorem 1.** For every expanded state *s*, it is guaranteed that $g(s)=g^*(s)$

*Sketch of proof by induction:*

- *assume all previously expanded states have optimal g-values*
- *next state to expand is s: $f(s) = g(s)+h(s)$ – min among states in OPEN*
- *assume g(s) is suboptimal (we will prove that it is impossible by contradiction)*
- *then there must be at least one state s' on an optimal path from start to s such that it is in OPEN but wasn't expanded*
- $g(s') + h(s') \geq g(s)+h(s)$
- *but* $g(s') + c^*(s',s) < g(s)$ =>
- $g(s') + c^*(s',s) + h(s) < g(s) + h(s)$ => *(from consistency of h-values)*
- $g(s') + h(s') < g(s) + h(s)$ => *CONTRADICTION*
- *thus it must be the case that g(s) is optimal*

# A* Search: Proofs

**Theorem 2.** Once the search terminates, it is guaranteed that $g(s_{goal})=g^*(s_{goal})$

*Sketch of proof:*

*Proof?*

# A* Search: Proofs

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

## *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)+cost(u,s) that are strictly larger than g(s')+cost(s',s))*

# A* Search: Proofs

**Theorem 4 (complexity).** No state is expanded more than once by A*

*Sketch of proof:*

*Proof?*

# A* Search: Proofs

**Theorem 5.** Given a graph and a heuristic function, **A* performs a minimal number of expansions to find a <u>provably</u> optimal path** *(provided goal state is always expanded first among the states with the same f-values in OPEN)*

# Implementation Details of A* Search

- Computes optimal g-values for relevant states

**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*;

*How to implement OPEN?*

*How to implement CLOSED?*

# Implementation Details of A* Search

- Computes optimal g-values for relevant states

**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*;

*How to implement OPEN?*

*Typically, a priority queue built using a binary heap*

*How to implement CLOSED?*

*Typically, each state has a Boolean flag indicating if it was already closed*

# A* Search with Backpointers

- After search terminates, least-cost path is given by backtracking backpointers from $s_{goal}$ to $s_{start}$

**Main function**

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

***set all backpointers bp to NULL;***

ComputePath();

publish solution; //**backtrack least-cost path using backpointers *bp***

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

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

      insert $s'$ into $OPEN$;

# What You Should Know…

- Operation of A*
- Understand why A* returns an optimal solution (e.g., understand the sketch of proof)
- Theoretical properties of A*
- Properties of heuristics (e.g., admissibility, consistency)