

16-782

Planning & Decision-making in Robotics

Search Algorithms:

A, Weighted A*, Backward A**

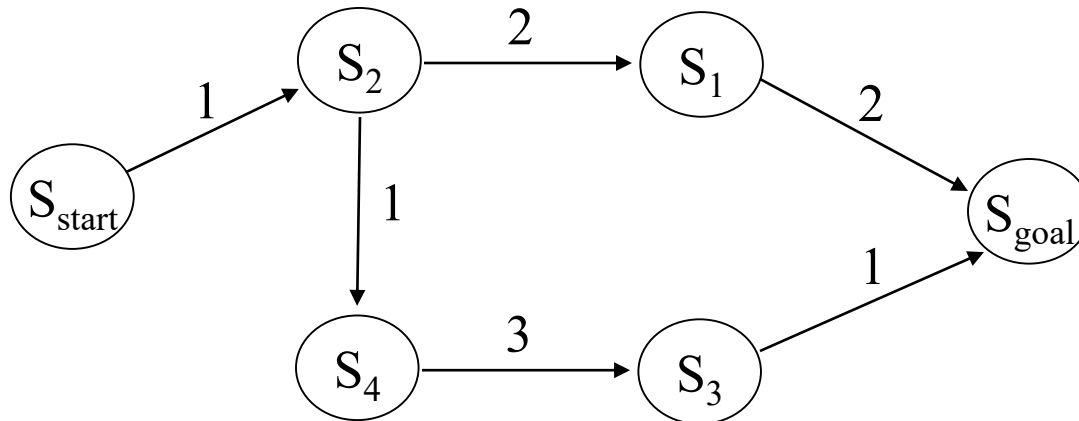
Maxim Likhachev

Robotics Institute

Carnegie Mellon University

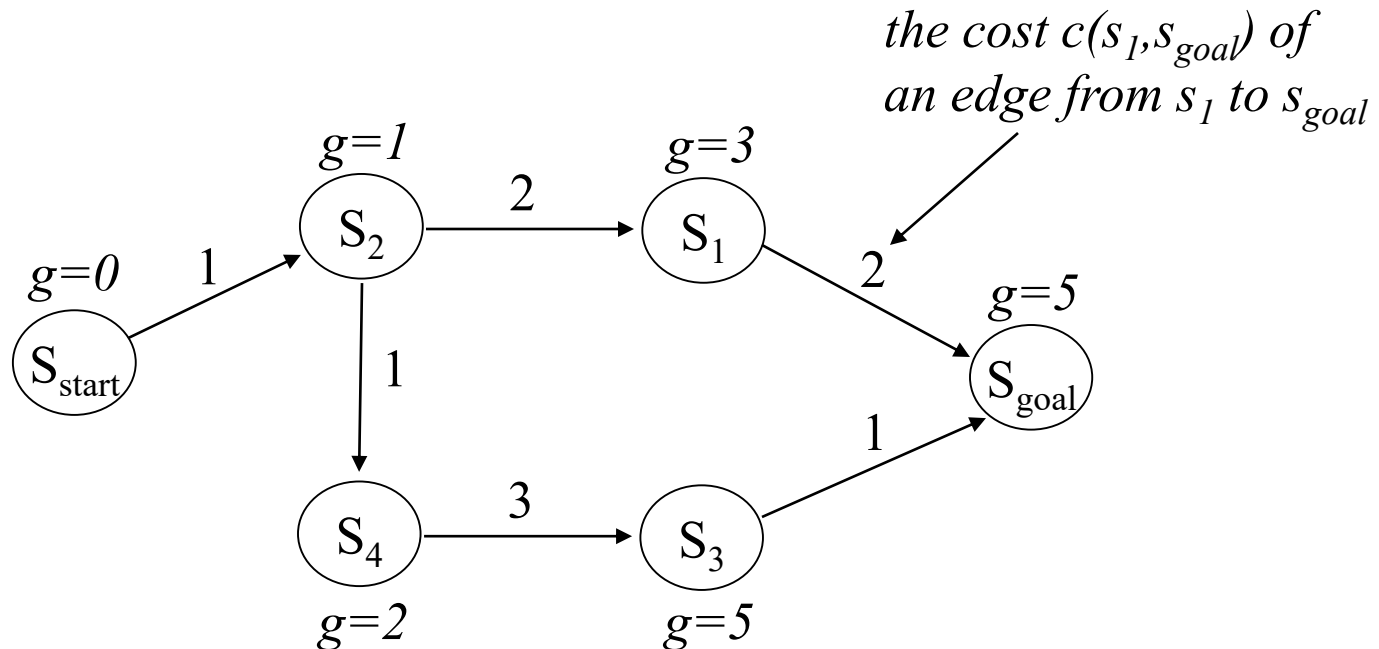
Searching Graphs for a Least-cost Path

- Once a graph is constructed (from skeletonization or uniform cell decomposition or adaptive cell decomposition or lattice or whatever else), we need to search it for a least-cost path



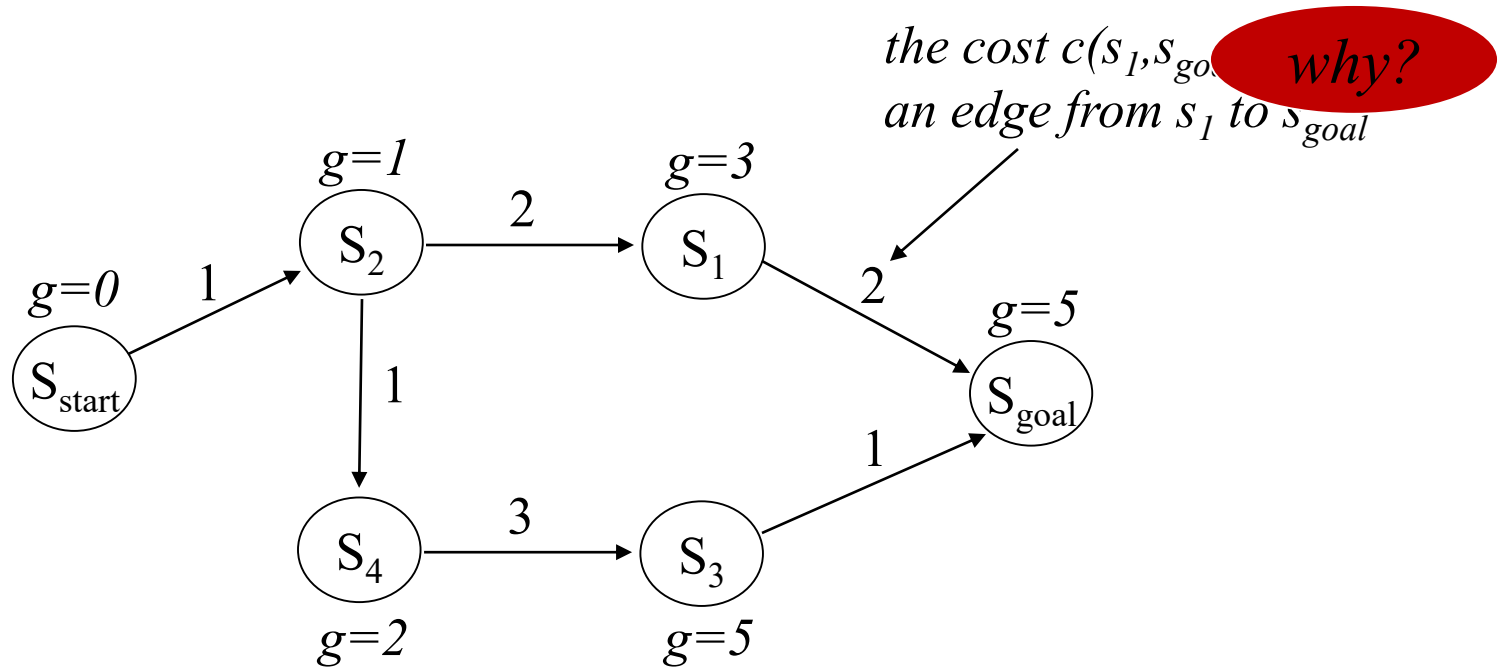
Searching Graphs for a Least-cost Path

- Many searches work by computing optimal g-values for relevant states
 - $g(s)$ – an estimate of the cost of a least-cost path from s_{start} to s
 - optimal values satisfy: $g(s) = \min_{s'' \in pred(s)} g(s'') + c(s'', s)$



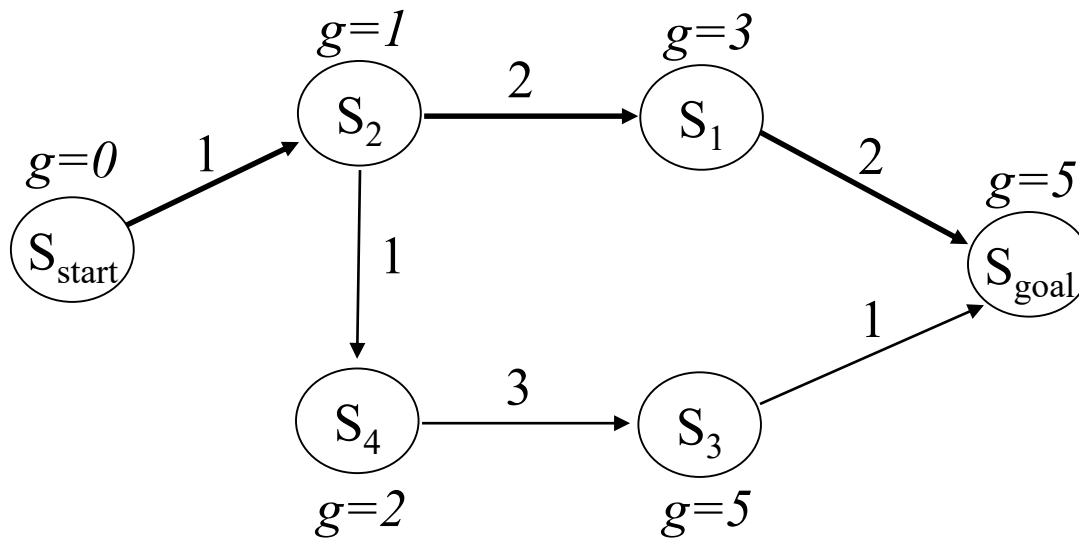
Searching Graphs for a Least-cost Path

- Many searches work by computing optimal g-values for relevant states
 - $g(s)$ – an estimate of the cost of a least-cost path from s_{start} to s
 - optimal values satisfy: $g(s) = \min_{s'' \in pred(s)} g(s'') + c(s'', s)$



Searching Graphs for a Least-cost Path

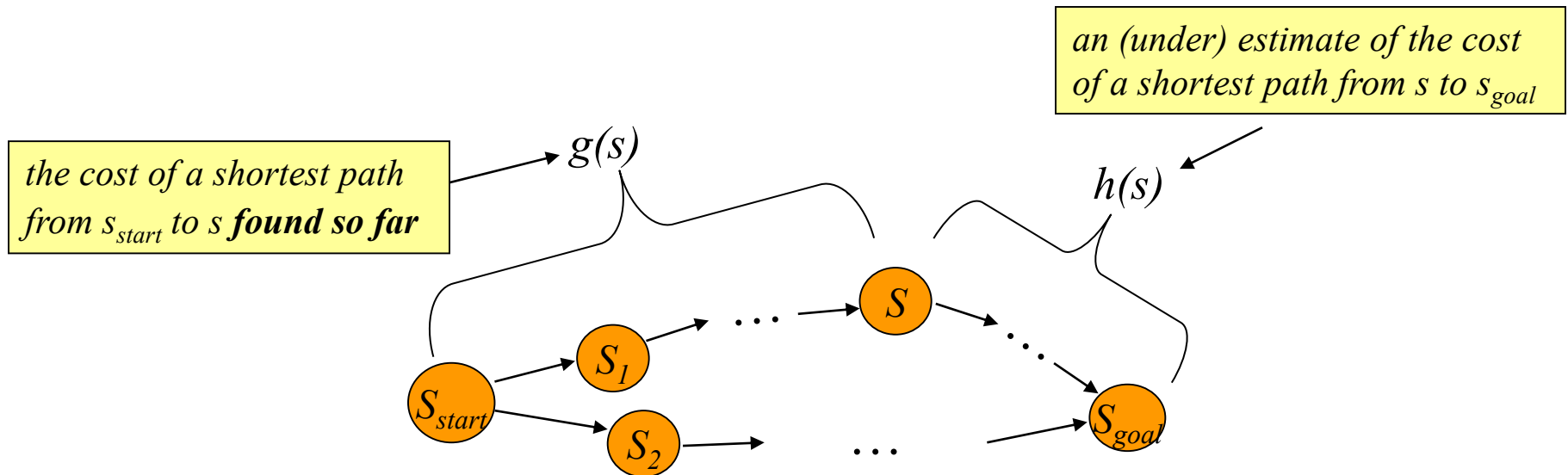
- Least-cost path is a greedy path computed by backtracking:
 - start with s_{goal} and from any state s move to the predecessor state s' such that
$$s' = \arg \min_{s'' \in pred(s)} (g(s'') + c(s'', s))$$



A* Search [Hart, Nilsson, Raphael, '68]

- Computes optimal g-values for relevant states

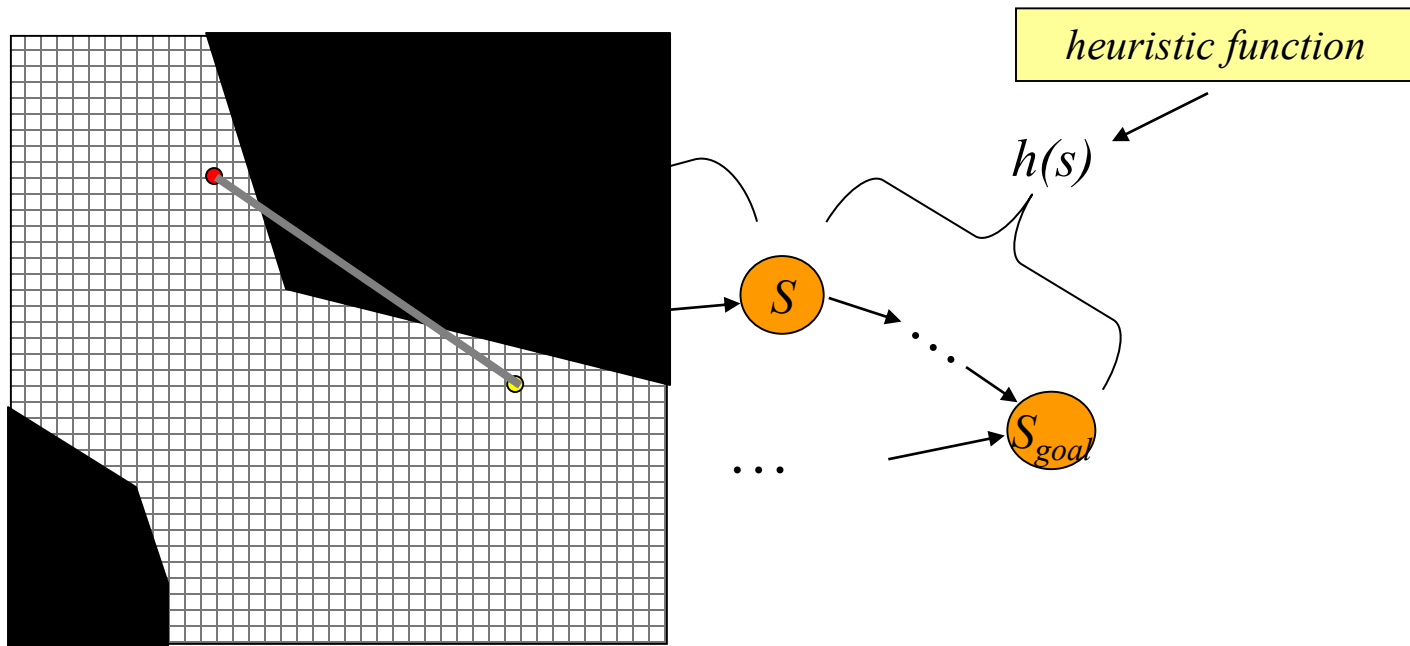
at any point of time:



A* Search

- Computes optimal g-values for relevant states

at any point of time:

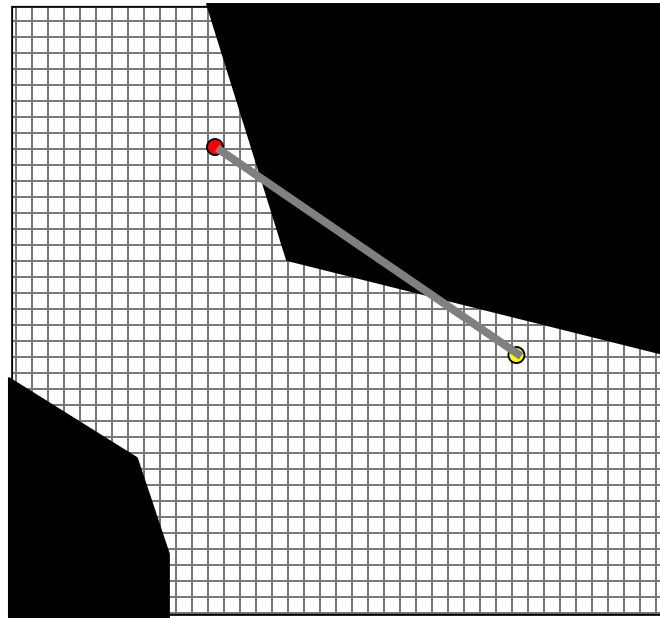


one popular heuristic function – Euclidean distance

A* Search

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

- 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

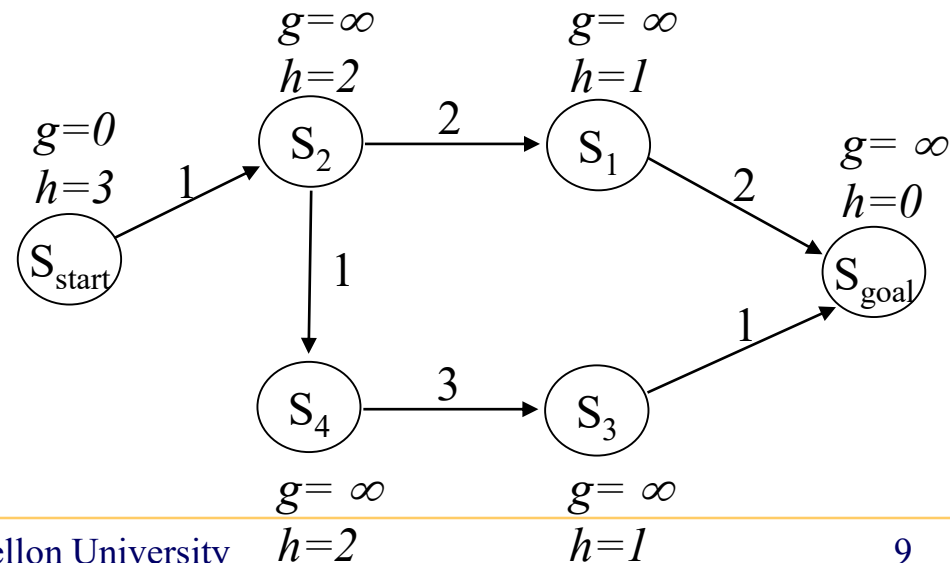
while(s_{goal} is not expanded and $OPEN \neq \emptyset$)

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

expand s ;

set of candidates for expansion

*for every expanded state
g(s) is optimal
(if heuristics are consistent)*



A* Search

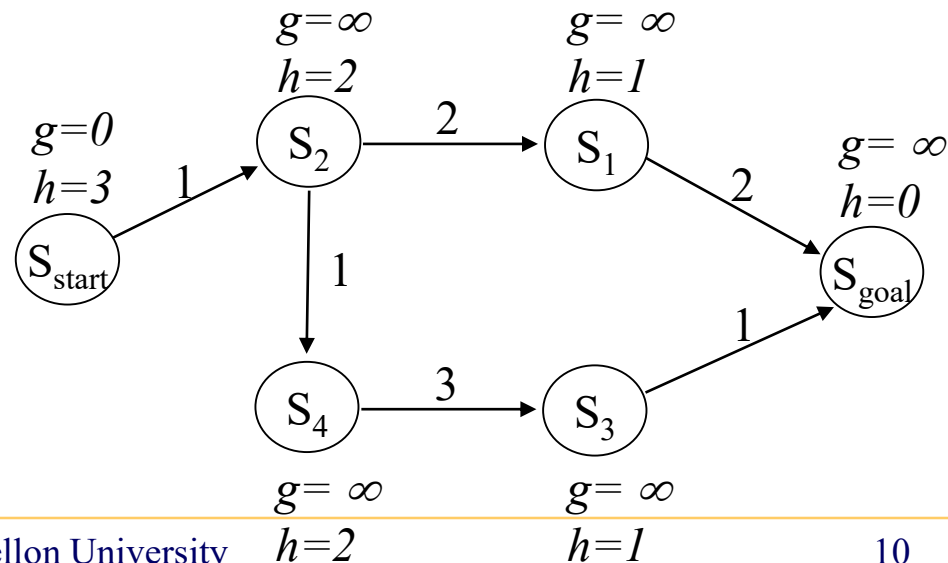
- Computes optimal g-values for relevant states

ComputePath function

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

 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 $CLOSED$

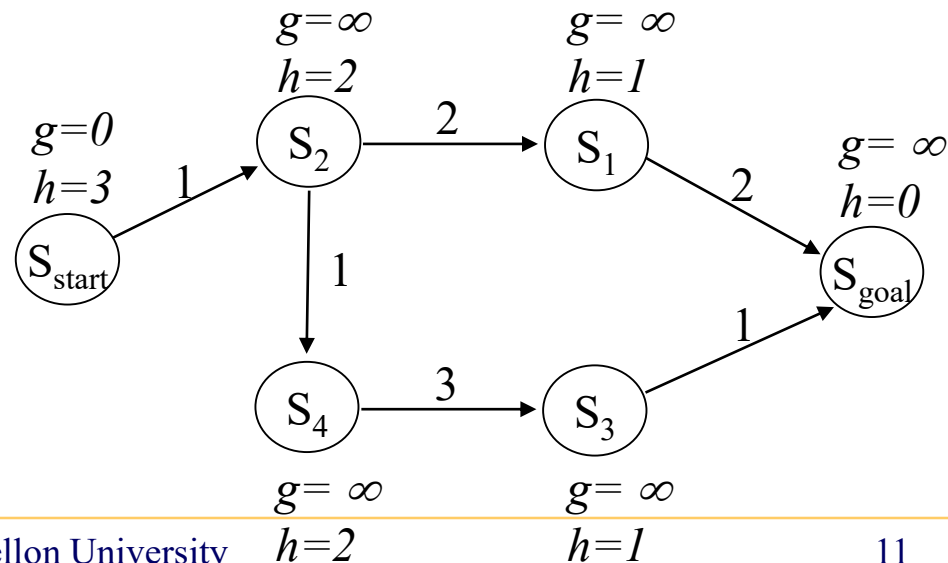
if $g(s') > g(s) + c(s, s')$

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

insert s' into $OPEN$;

tries to decrease $g(s')$ using the found path from s_{start} to s

set of states that have already been expanded



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

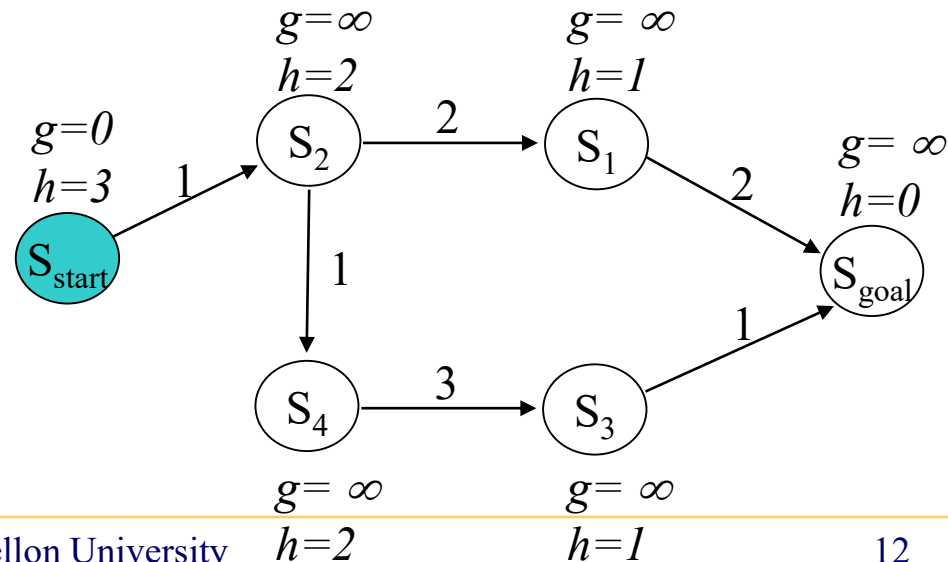
$g(s') = g(s) + c(s, s')$;

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

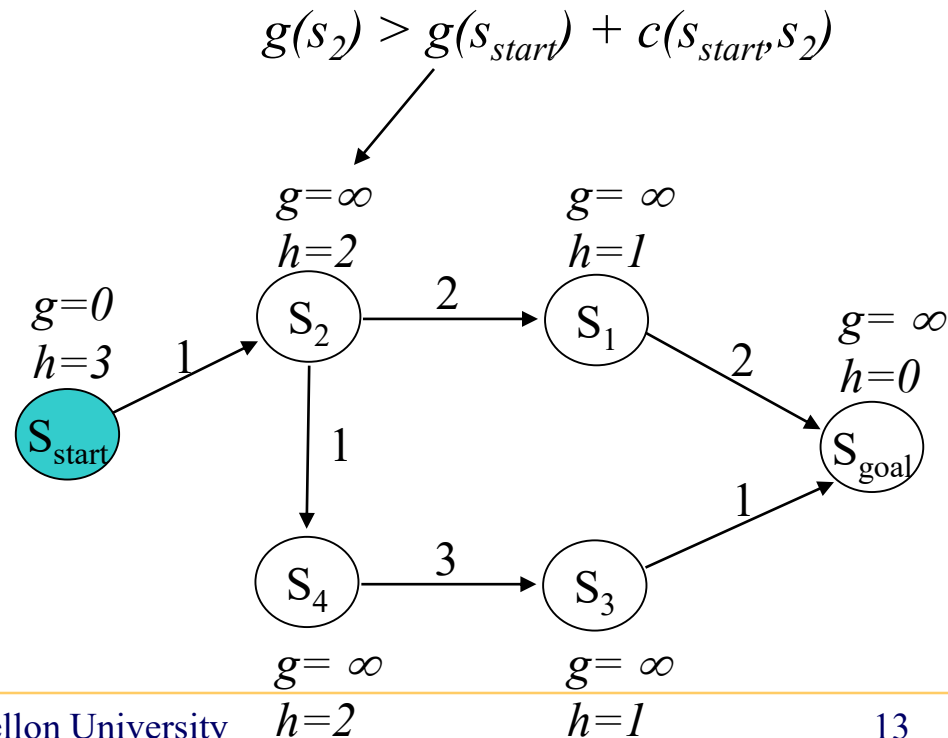
$g(s') = g(s) + c(s, s')$;

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 $OPEN$;

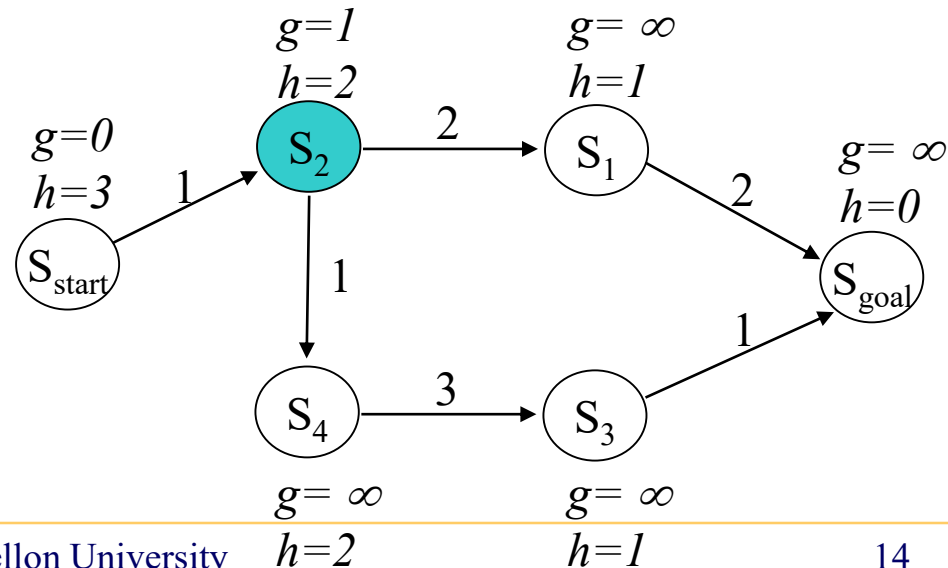
insert s into $CLOSED$;

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

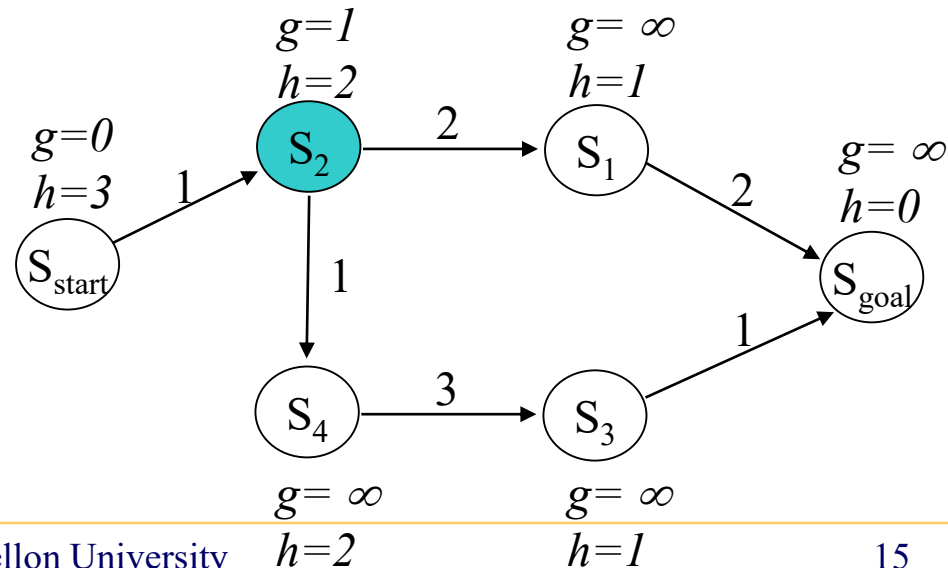
$g(s') = g(s) + c(s, s')$;

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

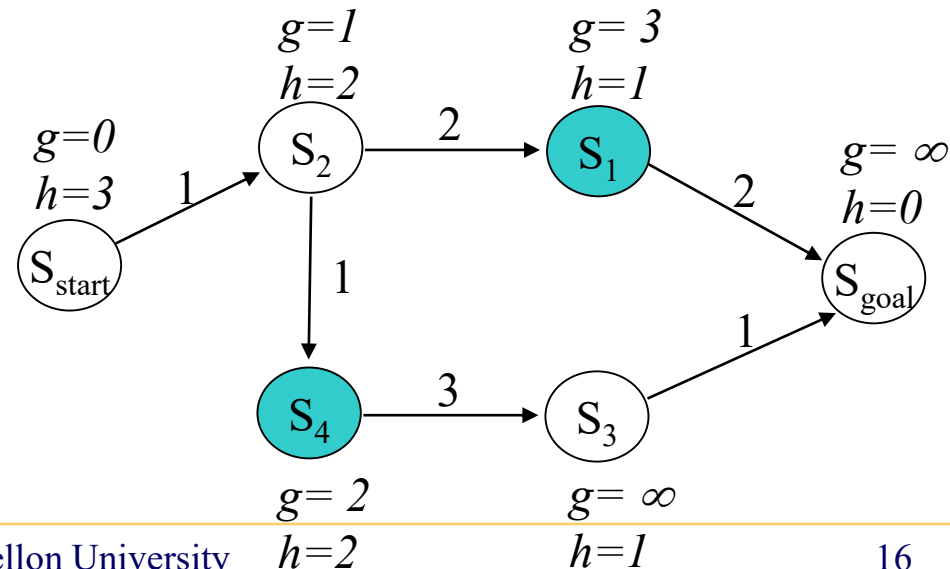
$g(s') = g(s) + c(s, s')$;

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

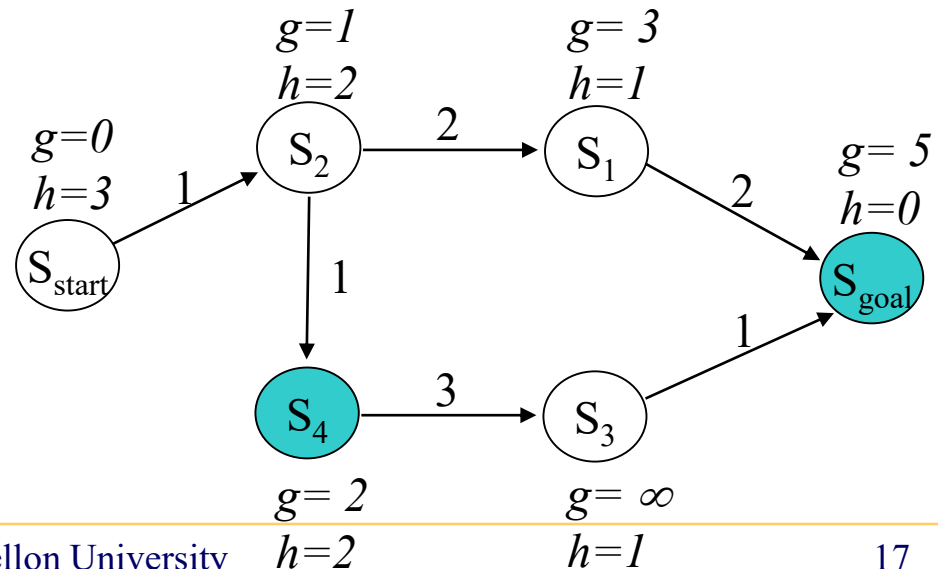
$g(s') = g(s) + c(s, s')$;

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

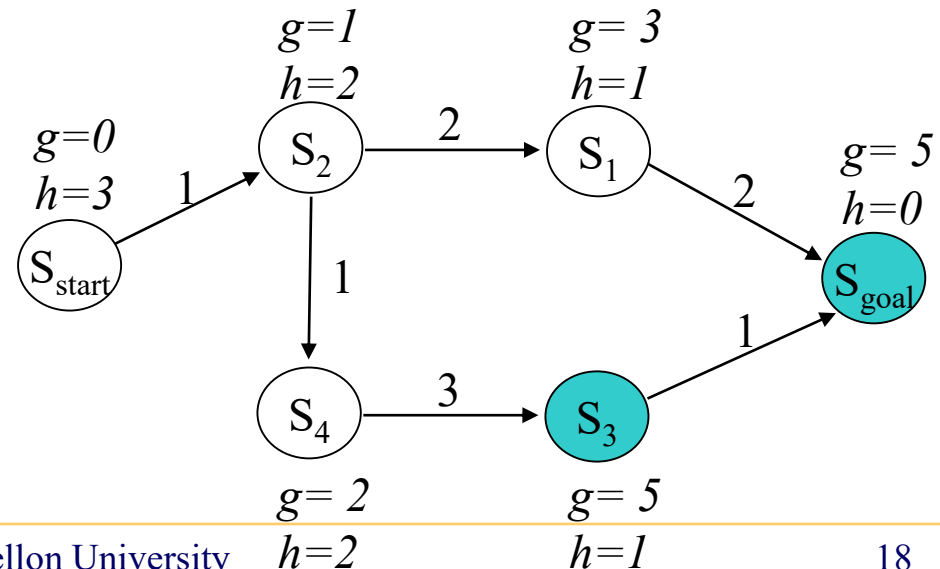
$g(s') = g(s) + c(s, s')$;

 insert s' into $OPEN$;

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

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

next state to expand: s_{goal}



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

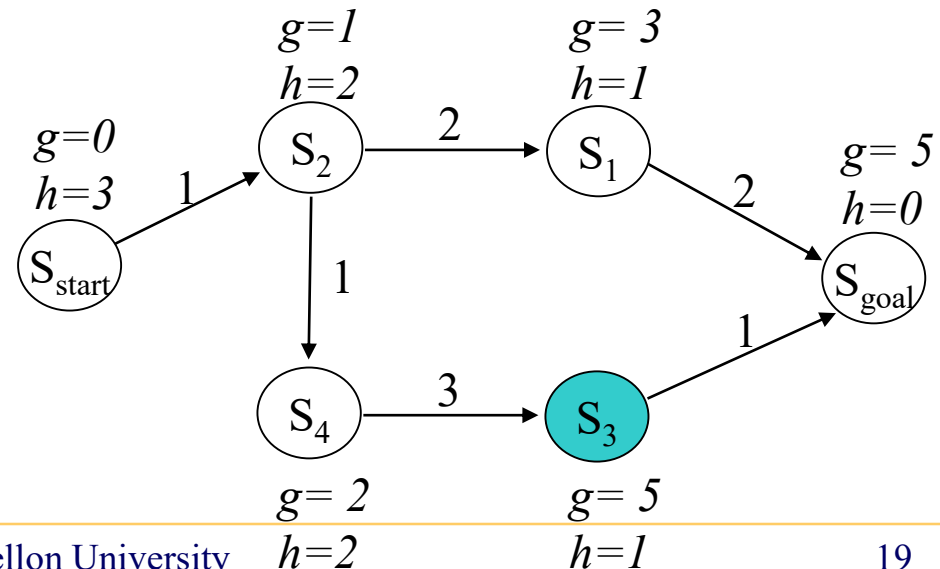
$g(s') = g(s) + c(s, s')$;

insert s' into $OPEN$;

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

$OPEN = \{s_3\}$

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 $OPEN$;

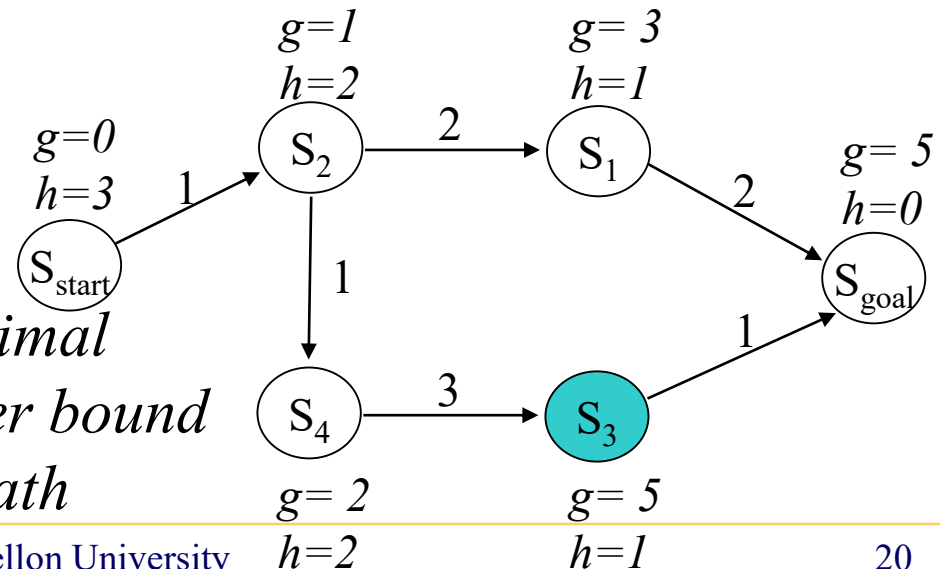
insert s into $CLOSED$;

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 $OPEN$;

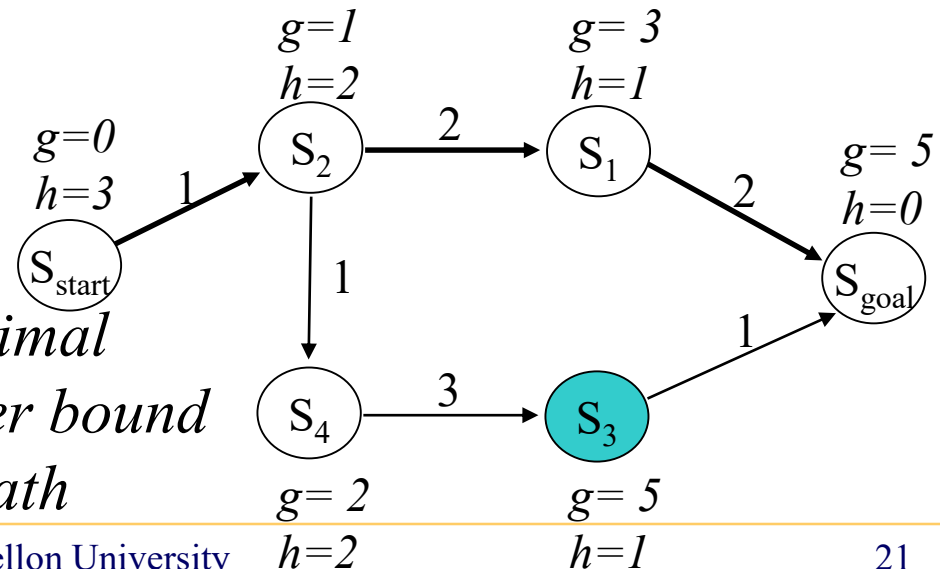
insert s into $CLOSED$;

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* 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 $OPEN$;

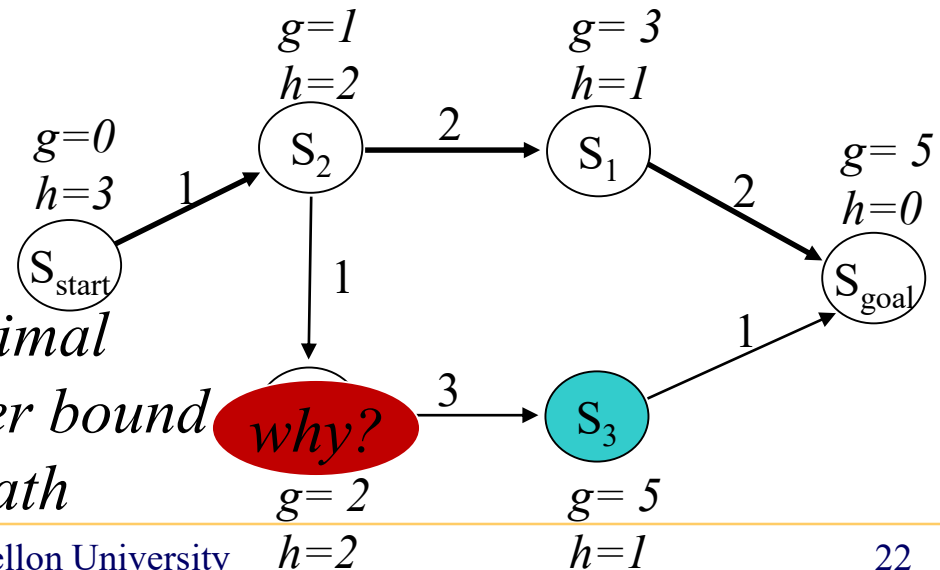
insert s into $CLOSED$;

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

- Is guaranteed to return an optimal path (in fact, for every expanded state) – optimal in terms of the solution
- Performs provably minimal number of state expansions required to guarantee optimality – optimal in terms of the computations

A* Search

- Is guaranteed to return an optimal path (in fact, for every expanded state) – optimal in terms of the solution

Sketch of proof by induction for $h = 0$:

assume all previously expanded states have optimal g-values

next state to expand is s : $f(s) = g(s) - \min$ among states in OPEN

OPEN separates expanded states from never seen states

thus, path to s via a state in OPEN or an unseen state will be worse than $g(s)$ (assuming positive costs)

Effect of the Heuristic Function

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

Effect of the Heuristic Function

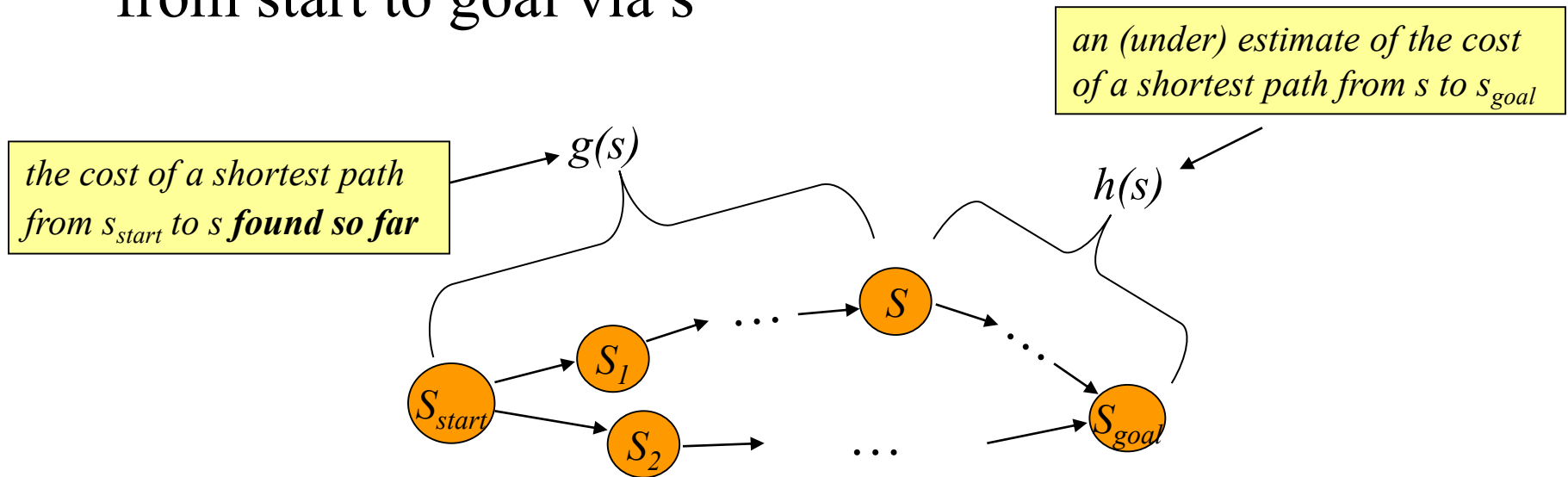
- A* Search: expands states in the order of $f = g+h$ values

Sketch of proof of optimality by induction for consistent h:

- 1. assume all previously expanded states have optimal g-values*
- 2. next state to expand is s : $f(s) = g(s)+h(s) - \min$ among states in OPEN*
- 3. assume $g(s)$ is suboptimal (i.e., proof by contradiction)*
- 4. then there must be at least one state s' on an optimal path from start to s such that it is still in OPEN*
- 5. $g(s') + h(s') \geq g(s)+h(s)$*
- 6. but $g(s') + c^*(s',s) < g(s) \Rightarrow$
 $g(s') + c^*(s',s) + h(s) < g(s) + h(s) \Rightarrow$
 $g(s') + h(s') < g(s) + h(s)$ (= contradiction)*
- 7. thus it must be the case that $g(s)$ is optimal*

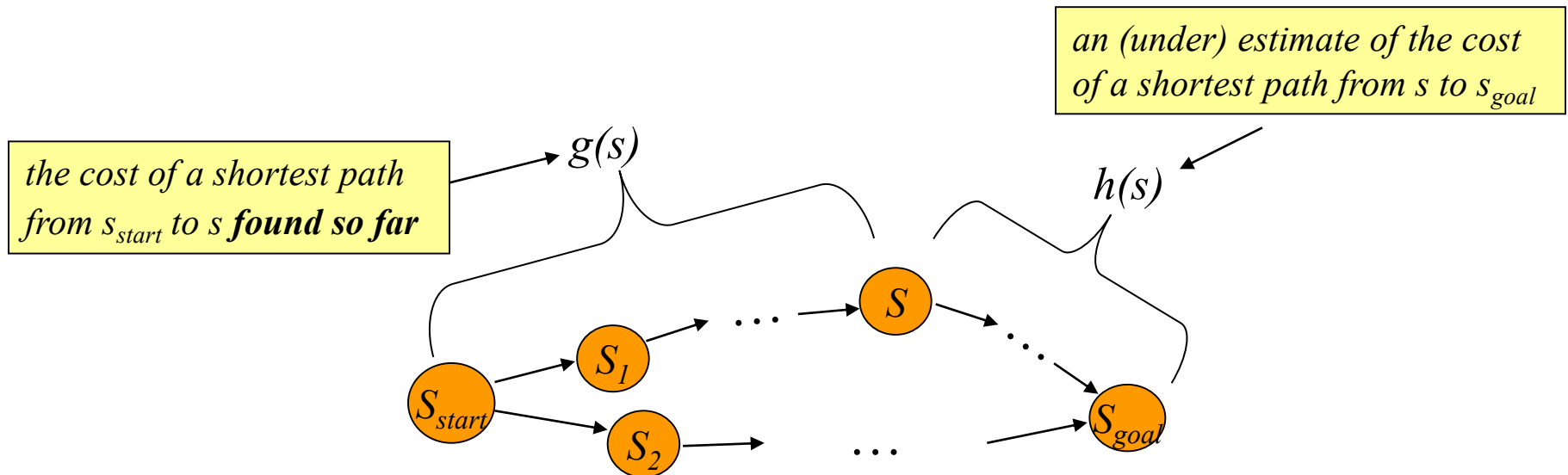
Effect of the Heuristic Function

- A* Search: expands states in the order of $f = g+h$ values
- Dijkstra's: expands states in the order of $f = g$ values (pretty much)
- Intuitively: $f(s)$ – estimate of the cost of a least cost path from start to goal via s



Effect of the Heuristic Function

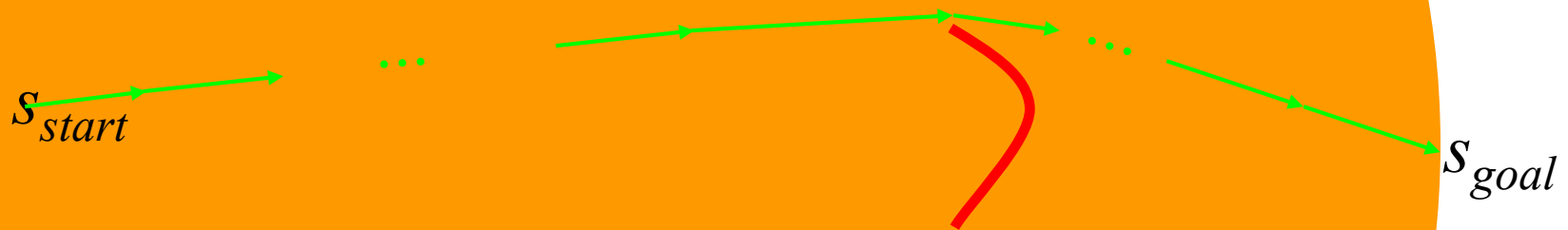
- **A*** Search: expands states in the order of $f = g+h$ values
- Dijkstra's: expands states in the order of $f = g$ values (pretty much)
- **Weighted A***: expands states in the order of $f = g+\epsilon h$ values, $\epsilon > 1$ = bias towards states that are closer to goal



Effect of the Heuristic Function

- Dijkstra's: expands states in the order of $f = g$ values

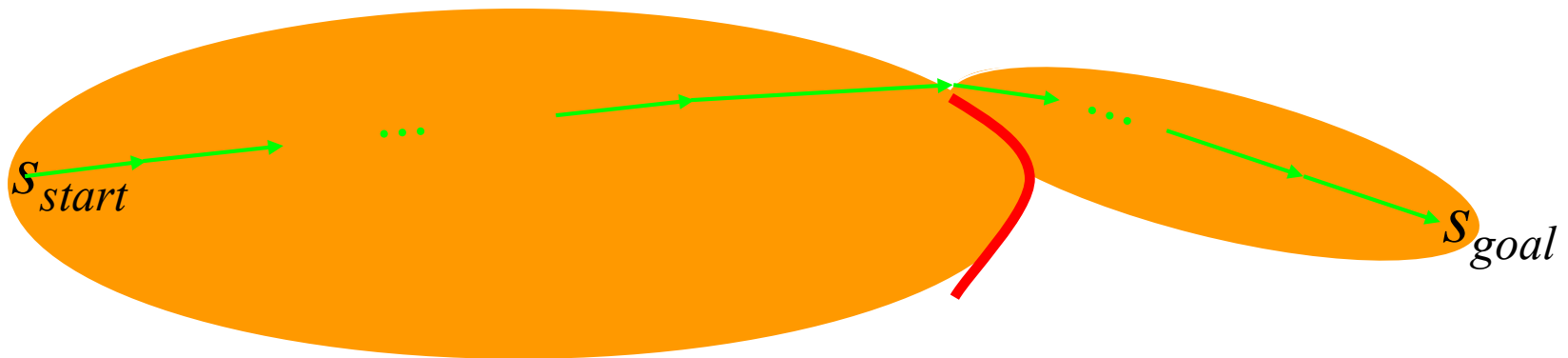
What are the states expanded?



Effect of the Heuristic Function

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

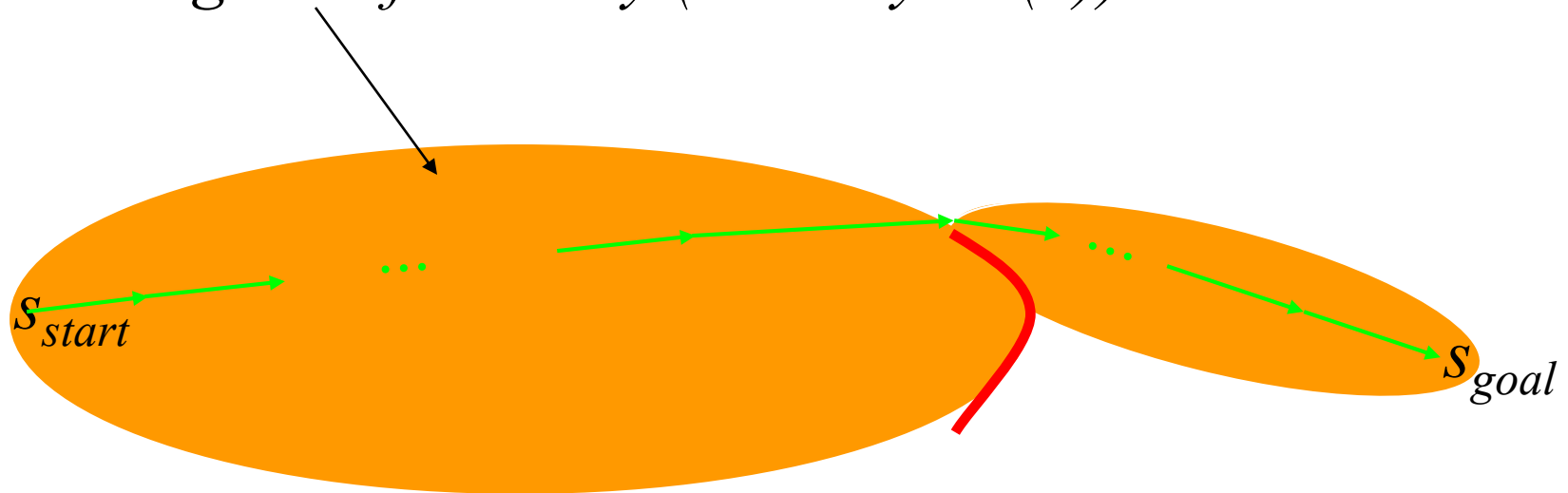
What are the states expanded?



Effect of the Heuristic Function

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

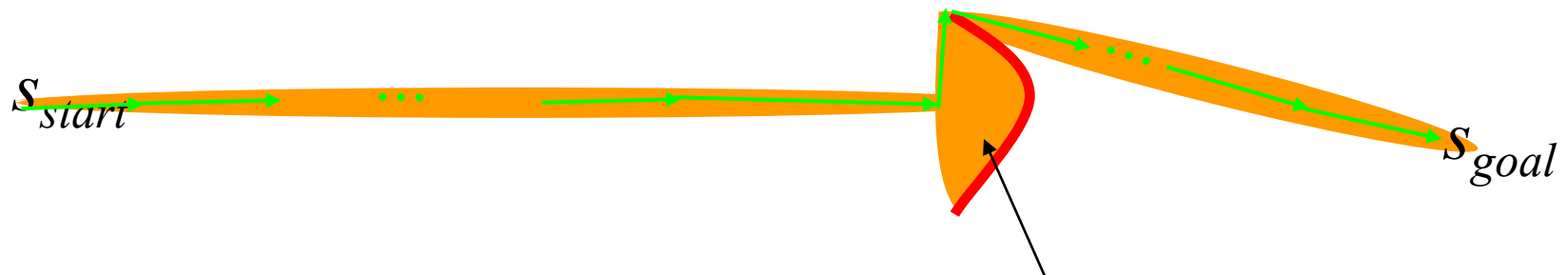
for large problems this results in A quickly running out of memory (memory: $O(n)$)*



Effect of the Heuristic Function

- Weighted A* Search: 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?



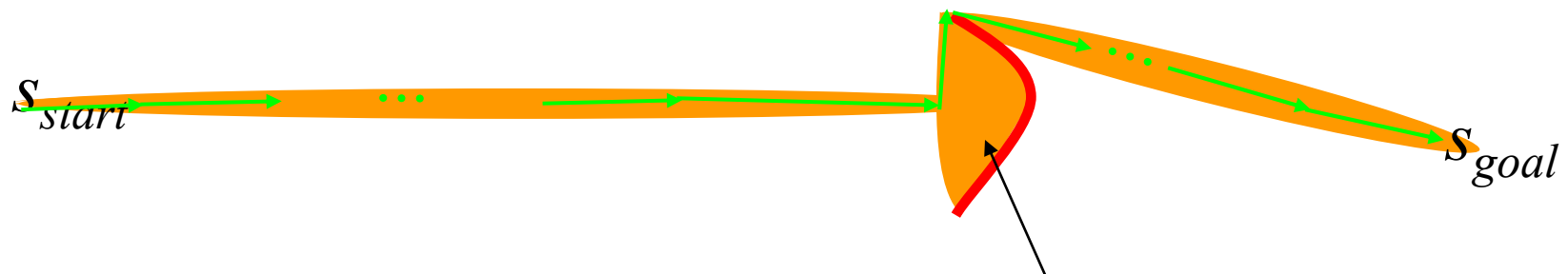
*key to finding solution fast:
shallow minima for $h(s) - h^*(s)$ function*

Effect of the Heuristic Function

- Weighted A* Search: 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*

Effect of the Heuristic Function

- Weighted A* Search:
 - trades off optimality for speed
 - ϵ -suboptimal:
$$\text{cost}(\text{solution}) \leq \epsilon \cdot \text{cost}(\text{optimal solution})$$
 - 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

Effect of the Heuristic Function

- Weighted A* Search:
 - trades off optimality for speed
 - ϵ -suboptimal:
$$\text{cost}(\text{solution}) \leq \epsilon \cdot \text{cost}(\text{optimal solution})$$
 - 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
- Weighted A* Search
 - with re-expansions (no Closed List) [Pohl, '70]
 - without re-expansions (with Closed List) [Likhachev et al., '04]
 - same sub-optimality guarantees but no more than 1 expansion per state

Effect of the Heuristic Function

- Weighted A* Search:

- trades off optimality for speed

- ϵ -suboptimal:

$$\text{cost}(\text{solution}) \leq \epsilon \cdot \text{cost}(\text{optimal solution})$$

- in many domains, it has been shown to be faster than A*

Is it guaranteed to expand no more states than A?*

- research becomes to develop a heuristic function that has shallow local minima

- Weighted A* Search

- with re-expansions (no Closed List) [Pohl, '70]

- without re-expansions (with Closed List) [Likhachev et al., '04]

- same sub-optimality guarantees but no more than 1 expansion per state

Backward A* Search

- Searches from goal towards states
- g-values are cost-to-goals

Main function

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

ComputePath();

publish solution;

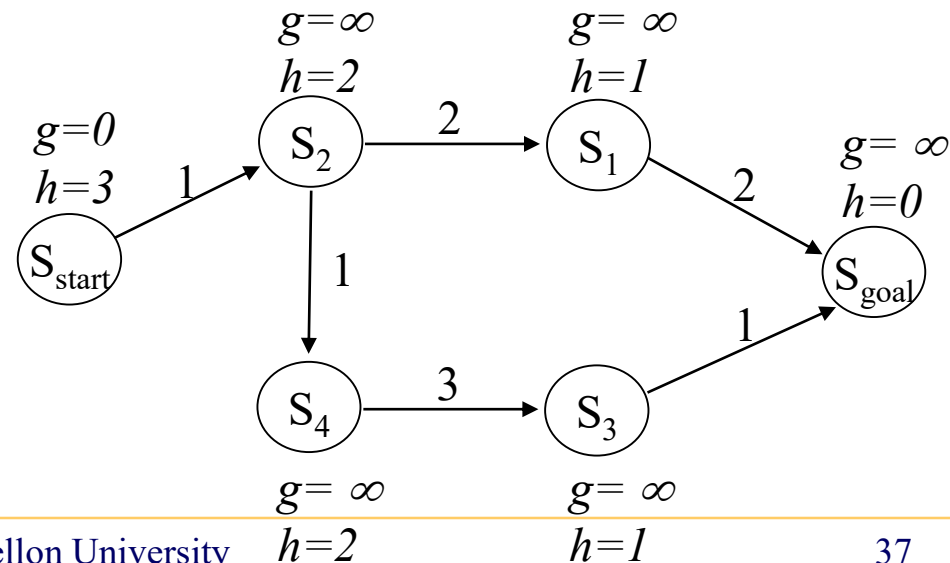
What needs to be changed?

ComputePath function

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

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

expand s ;



Backward A* Search

- Searches from goal towards states
- g-values are cost-to-goals

Main function

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

ComputePath();

publish solution;

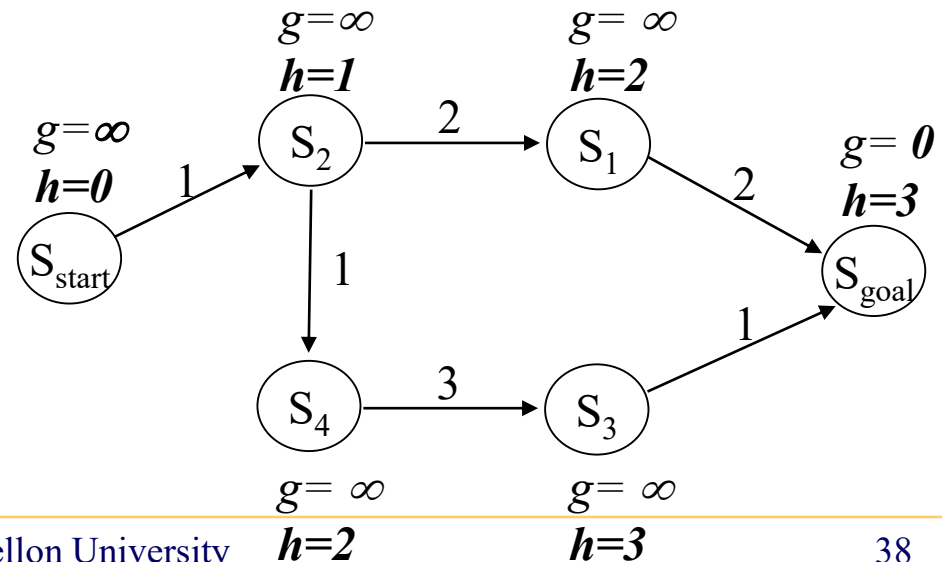
What needs to be changed?

ComputePath function

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

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

expand s ;



Backward A* Search

- Searches from goal towards states
- 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 $OPEN$;

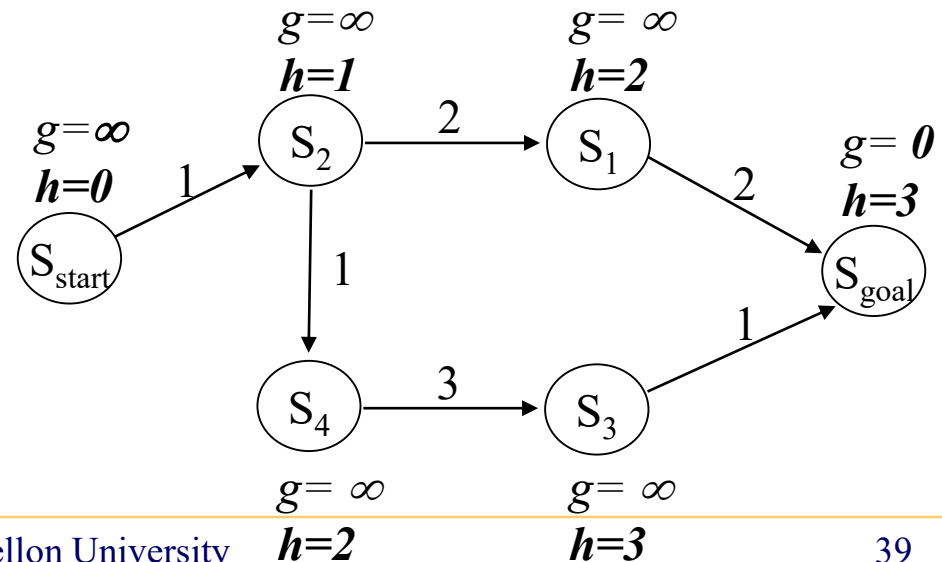
insert s into $CLOSED$;

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

- Searches from goal towards states
- 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 $OPEN$;

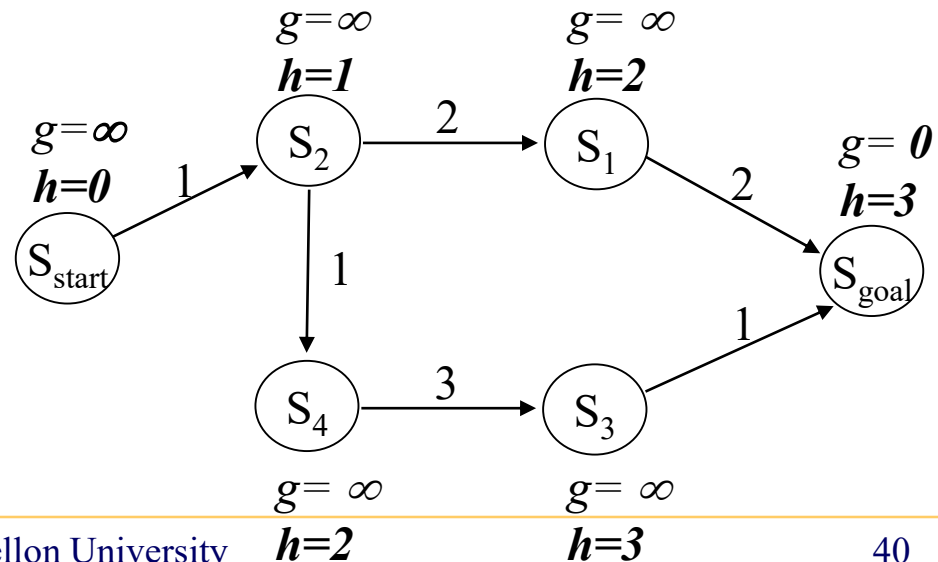
insert s into $CLOSED$;

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

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

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

insert s' into $OPEN$;



Backward A* Search

- Searches from goal towards states
- 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 $g(s') > c(s',s) + g(s)$

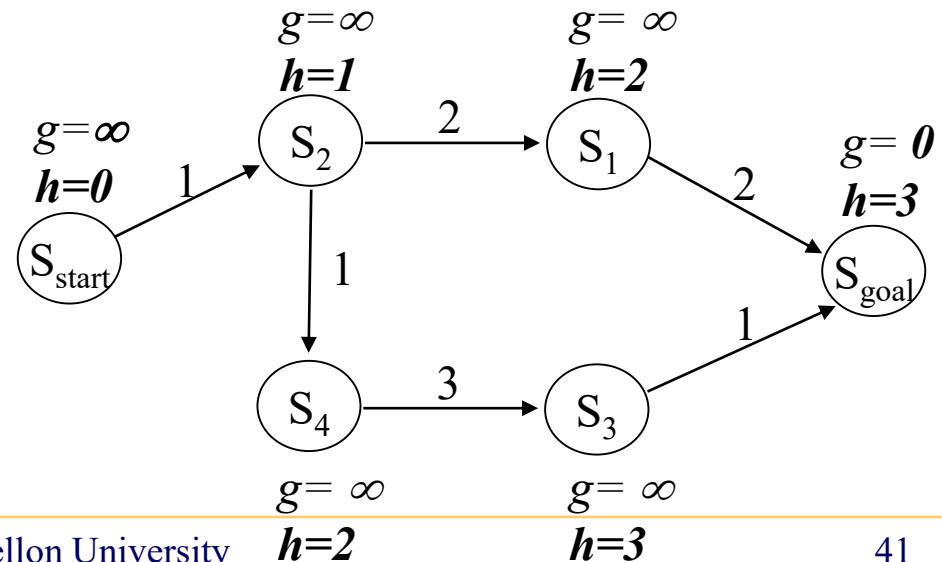
$g(s') = c(s',s) + g(s)$;

insert s' into $OPEN$;

$CLOSED = \{\}$

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

next state to expand: s_{goal}



Backward A* Search

- Searches from goal towards states
- 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 $g(s') > c(s',s) + g(s)$

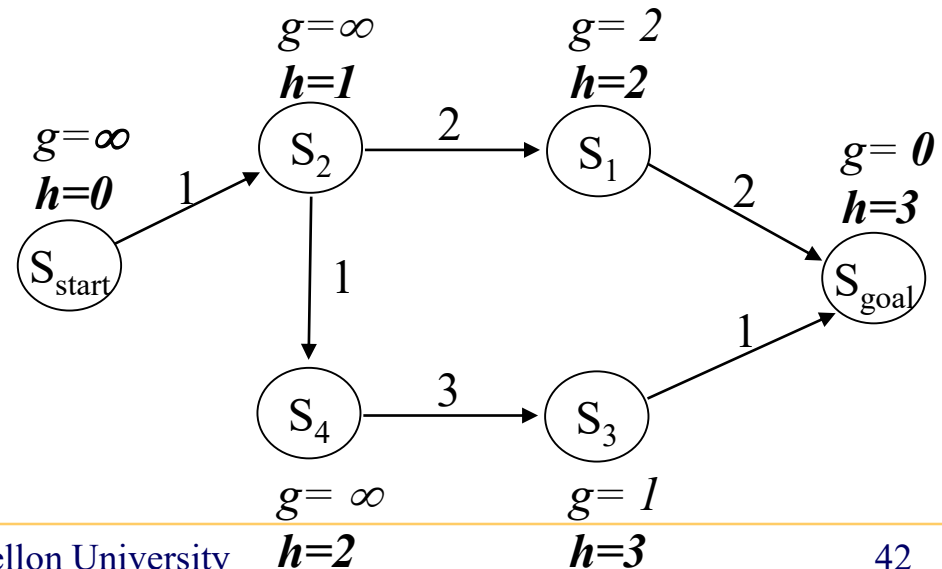
$g(s') = c(s',s) + g(s)$;

insert s' into $OPEN$;

$CLOSED = \{\}$

$OPEN = \{s_1, s_3\}$

next state to expand: s_1



Backward A* Search

- Searches from goal towards states
- 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 $g(s') > c(s',s) + g(s)$

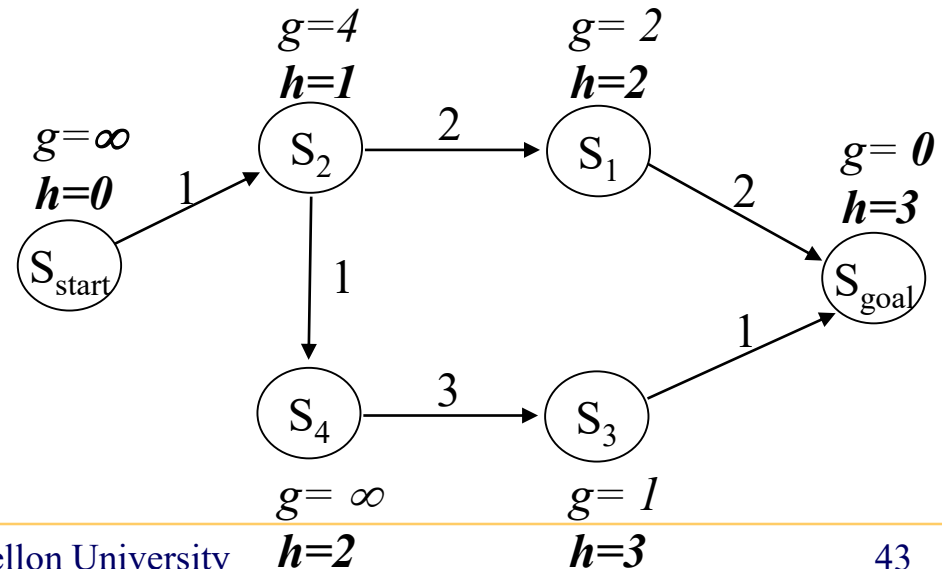
$g(s') = c(s',s) + g(s)$;

insert s' into $OPEN$;

$CLOSED = \{\}$

$OPEN = \{s_2, s_3\}$

next state to expand: s_3



Backward A* Search

- Searches from goal towards states
- 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 $g(s') > c(s',s) + g(s)$

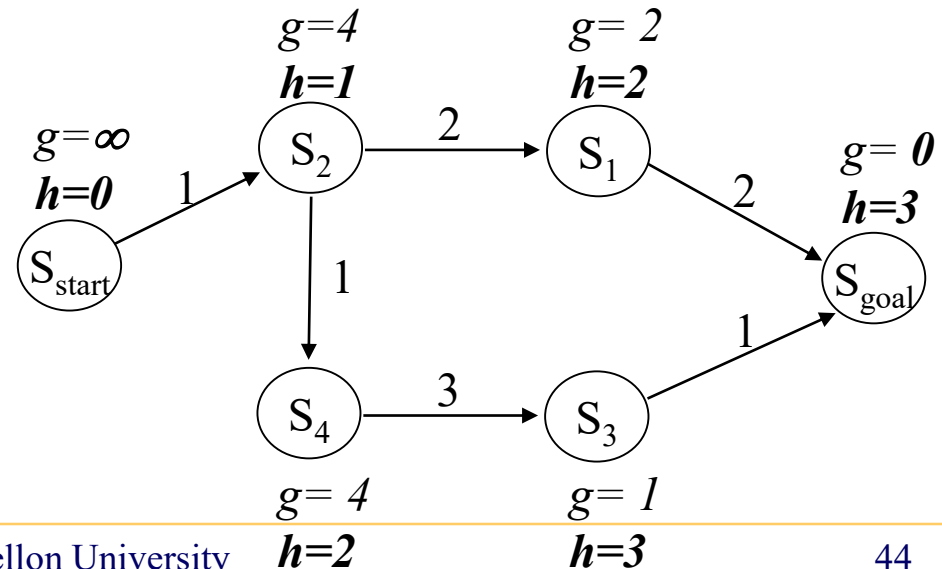
$g(s') = c(s',s) + g(s)$;

insert s' into $OPEN$;

$CLOSED = \{\}$

$OPEN = \{s_2, s_4\}$

next state to expand: s_2



Backward A* Search

- Searches from goal towards states
- 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 $g(s') > c(s',s) + g(s)$

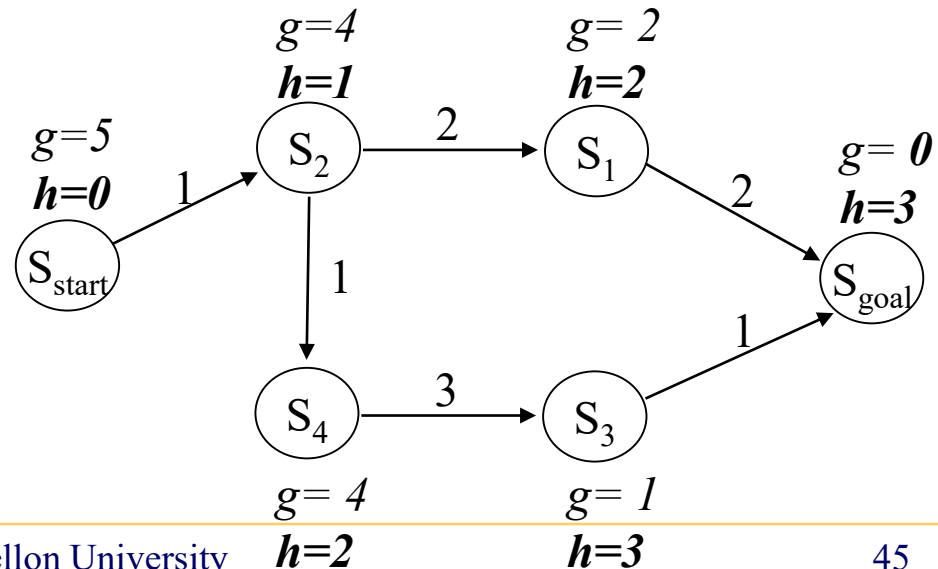
$g(s') = c(s',s) + g(s)$;

insert s' into $OPEN$;

$CLOSED = \{\}$

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

next state to expand: s_{start}



Backward A* Search

- Searches from goal towards states
- g-values are cost-to-goals

ComputePath function

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

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

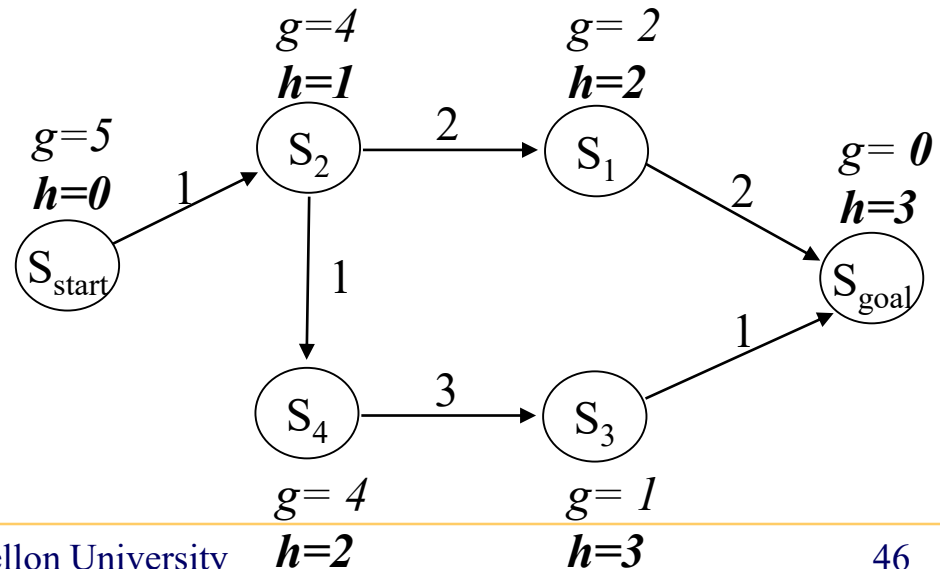
$g(s') = c(s',s) + g(s)$;

 insert s' into $OPEN$;

$CLOSED = \{\}$

$OPEN = \{s_4\}$

done



Using A^* to Compute a Policy

- Imagine planning for the agent that can easily deviate off the path



- Can A^* compute least-cost paths from **all** the states of interest?

Using A^* to Compute a Policy

- Imagine planning for the agent that can easily deviate off the path



- Can A^* compute least-cost paths from **all** the states of interest?
 - Run Backward A^* search until all states of interest have been expanded

Using A* to Compute a Policy

- Backward A* search to compute least-cost paths for all states $s \in \Phi$

ComputePath function

while(at least one state in Φ hasn't been 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 $OPEN$;

- Guaranteed to compute least-cost paths for all $s \in \Phi$ that can reach goal

What You Should Know...

- A^*
 - How it works
 - Theoretical properties
 - Proof for its optimality
- Weighted A^*
- Backwards A^*
- A^* can be used to compute a policy and not just a single path