16-350
Planning Techniques for Robotics

Search Algorithms: Uninformed A Search*

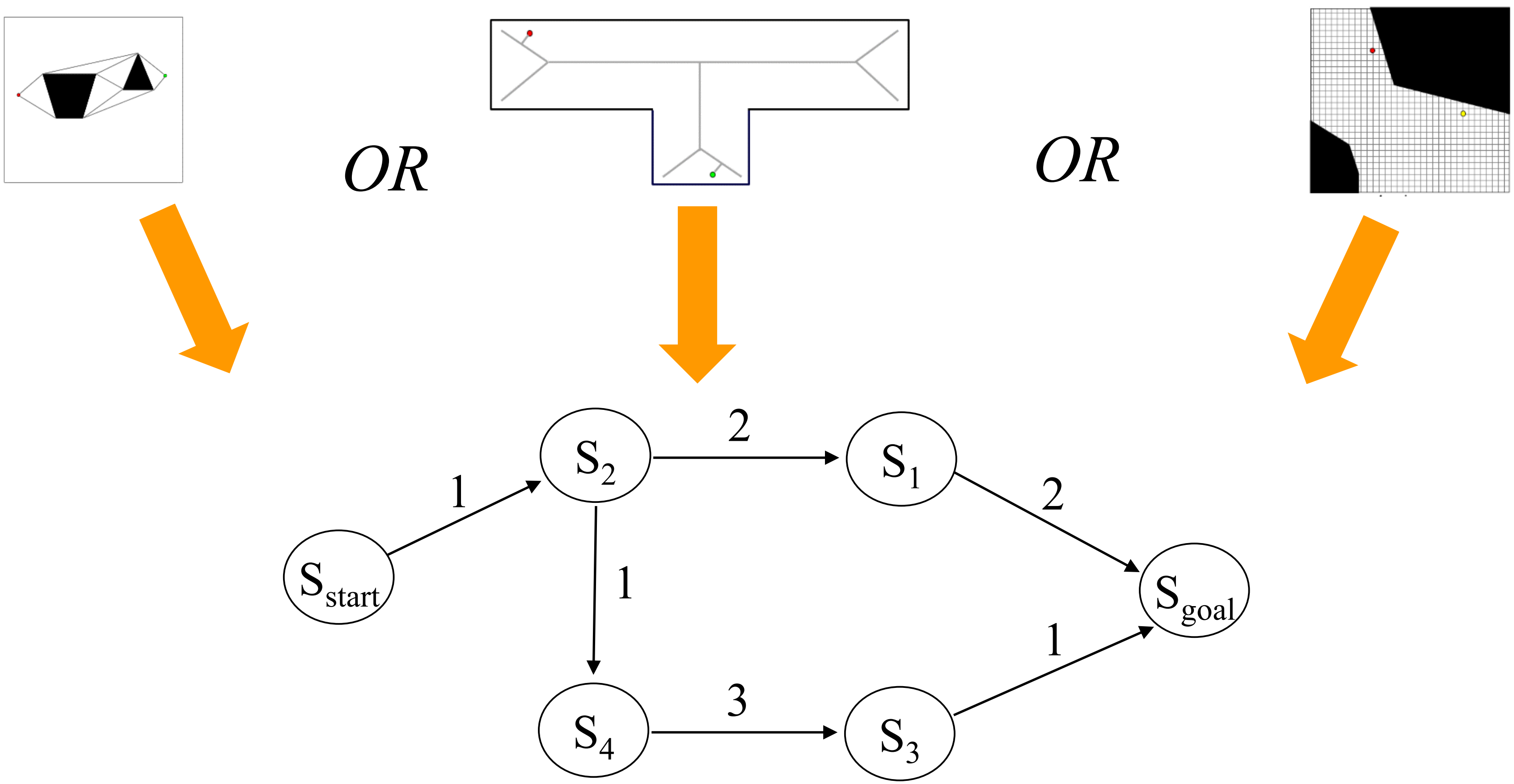
Maxim Likhachev

Robotics Institute

Carnegie Mellon University

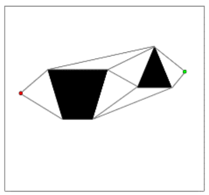
Searching Graphs for a Least-cost Path

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

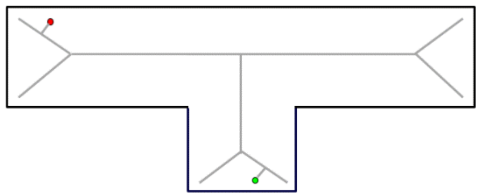


Searching Graphs for a Least-cost Path

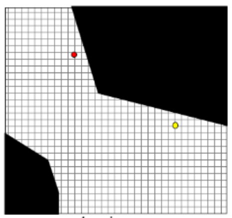
- Once a graph is constructed (from skeletonization or cell decomposition or whatever else), we need to search it for a least-cost path

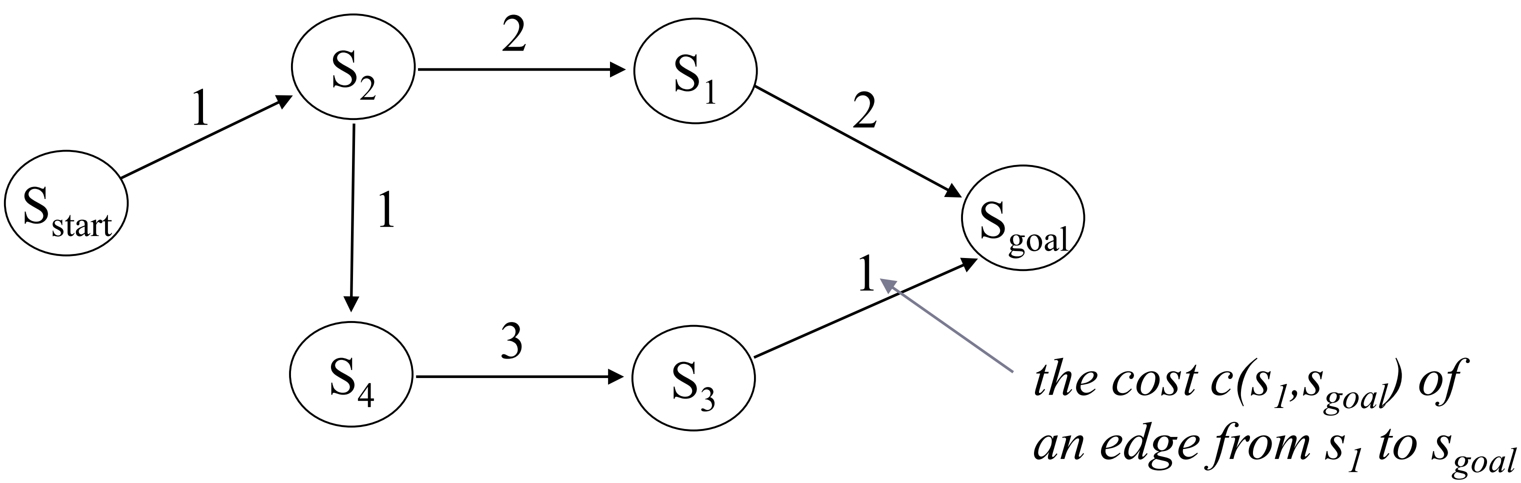


OR



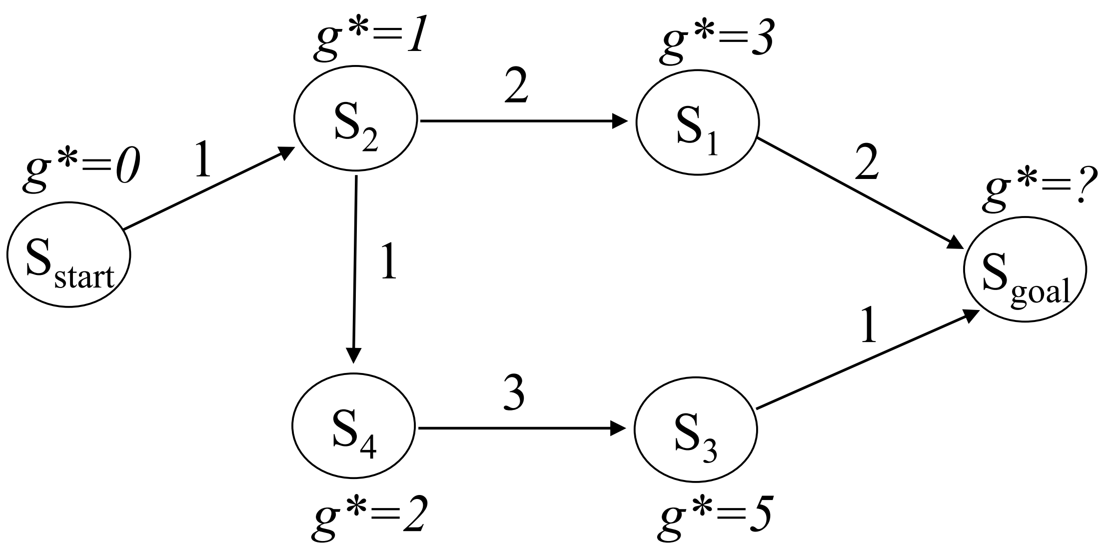
OR





Searching Graphs for a Least-cost Path

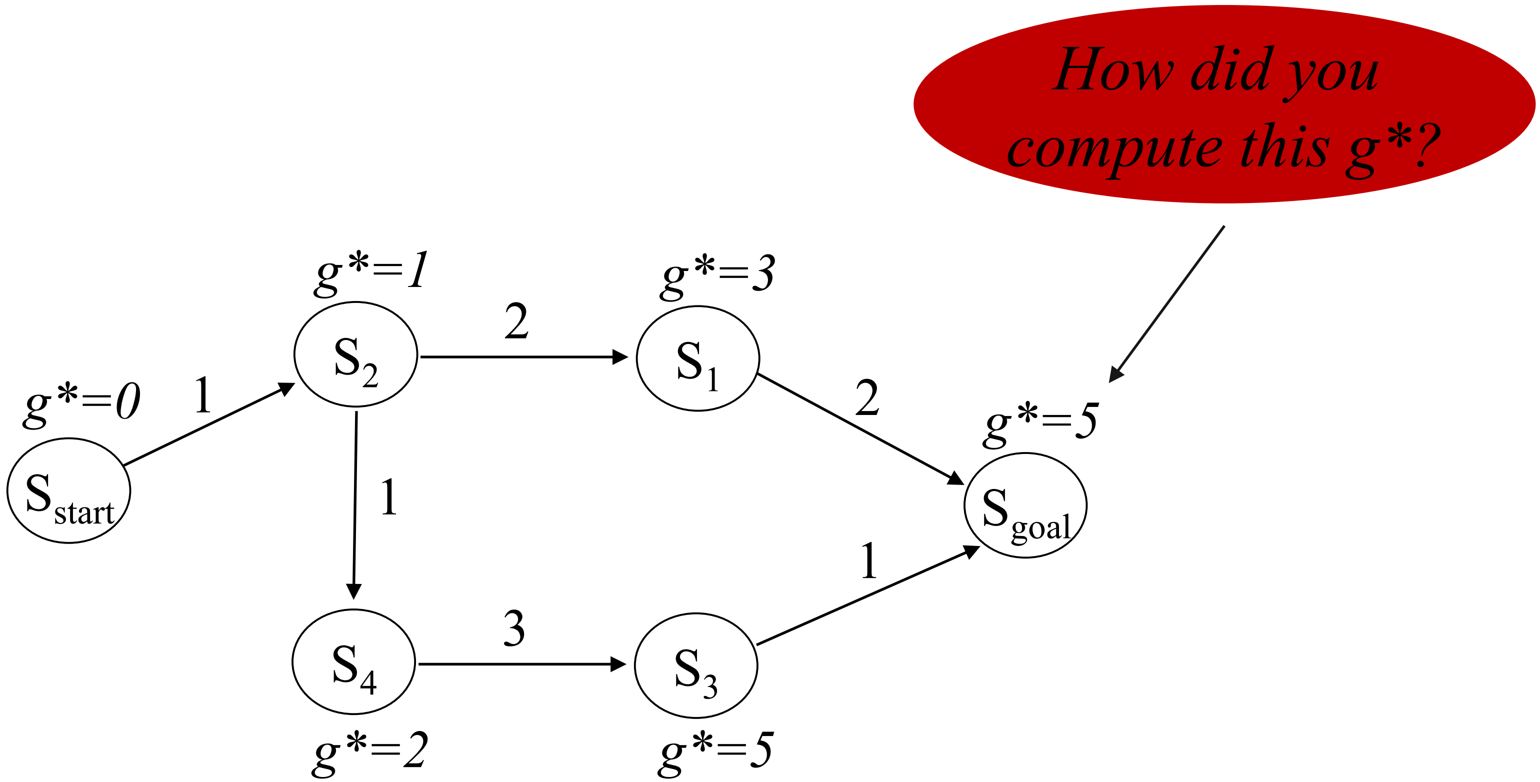
- Many searches (including A*) work by computing g* values for graph vertices (states)
  - $g^*(s)$ – the cost of a least-cost path from s_{start} to s



Searching Graphs 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_{start} to s

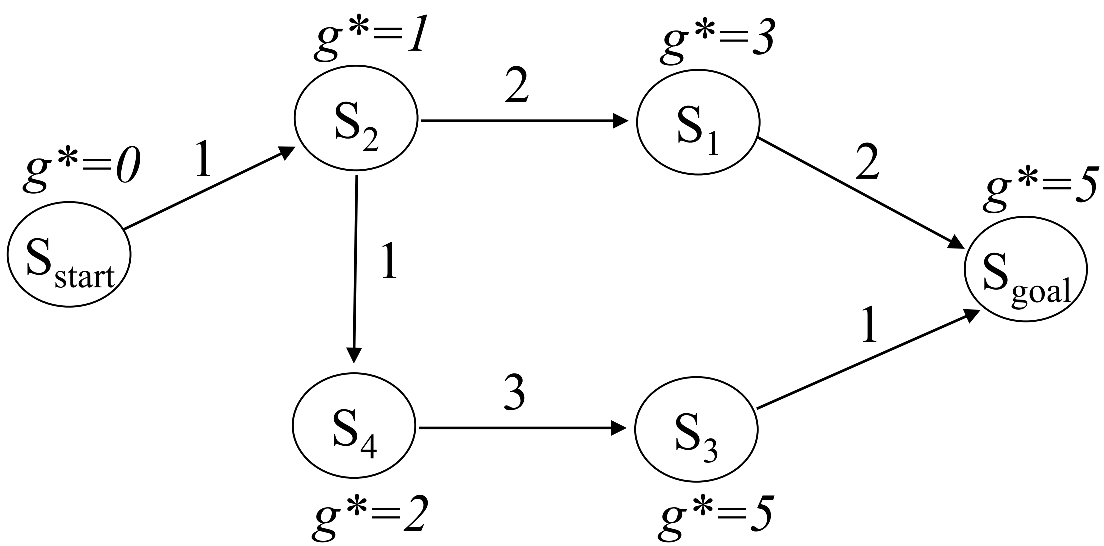


Searching Graphs for a Least-cost Path

- Many searches (including A*) work by computing g* values for graph vertices (states)

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

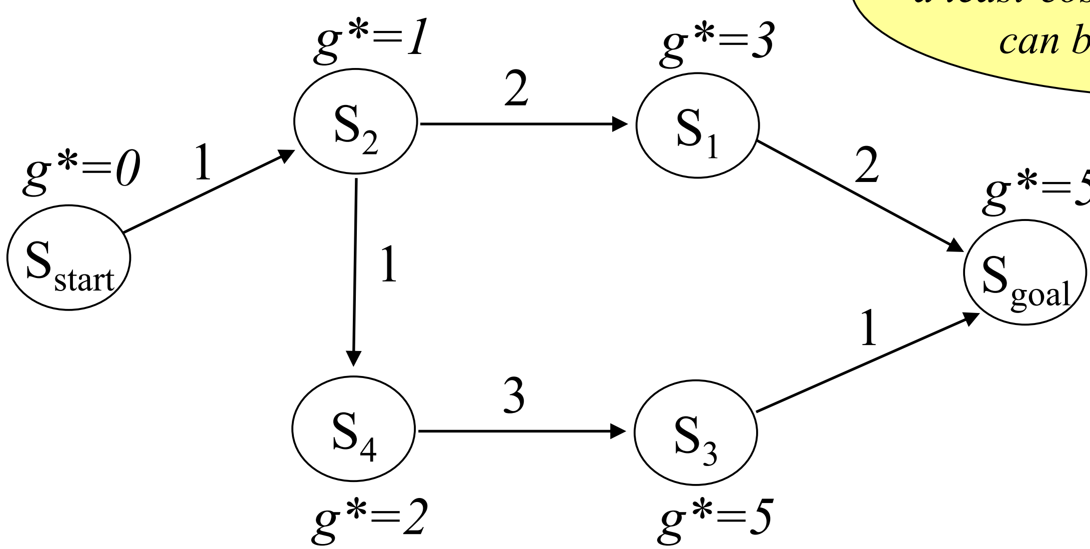
  - g* values satisfy: $g^*(s) = \min_{s'' \in pred(s)} g^*(s'') + c(s'',s)$



Searching Graphs 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_{start} to s
  - g* values satisfy: $g^*(s) = \min_{s'' \in pred(s)} g^*(s'') + c(s'',s)$

Once g-values are computed, a least-cost path from s_{start} to s_{goal} can be* ***easily*** *computed!*

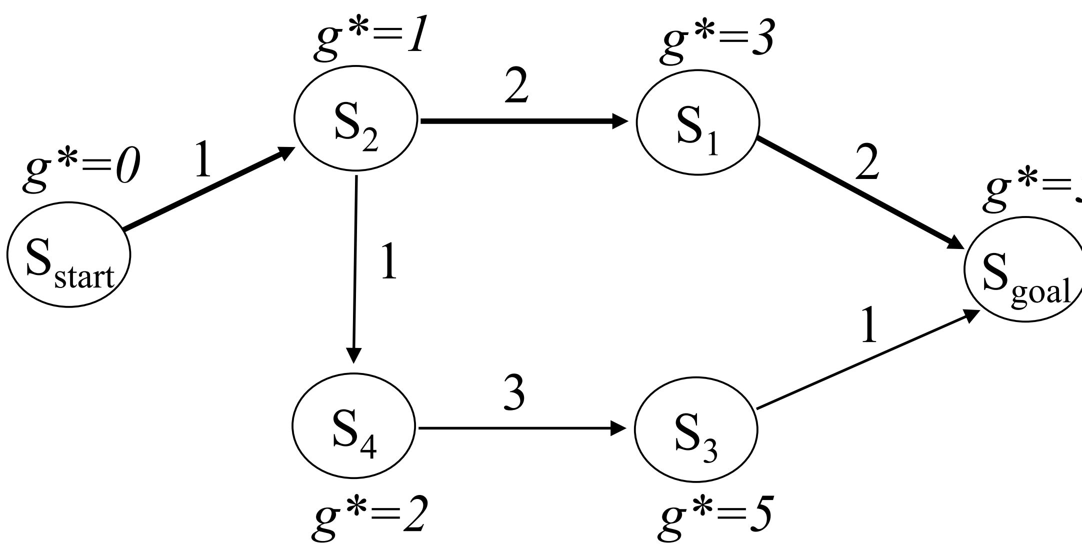


How?

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 backtrack to the predecessor state s' such that

$$s' = \arg\min_{s'' \in pred(s)} (g^*(s'') + c(s'', s))$$

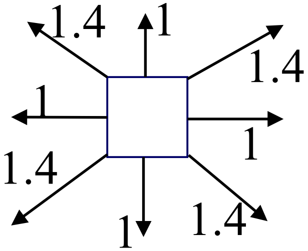


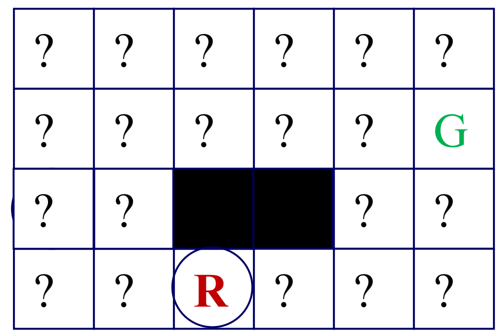
Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

How do we compute g-values?*

8-connected grid



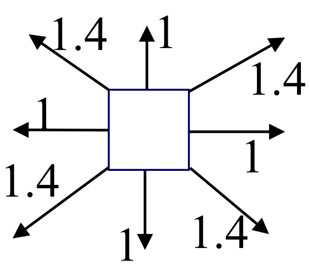


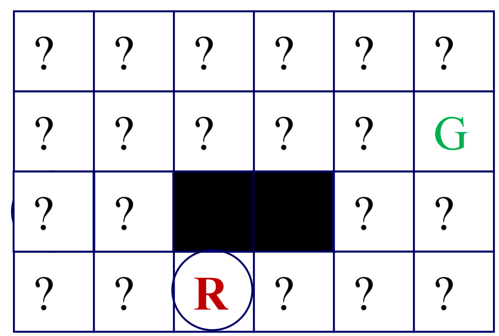
Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

How do we compute g-values?*

8-connected grid



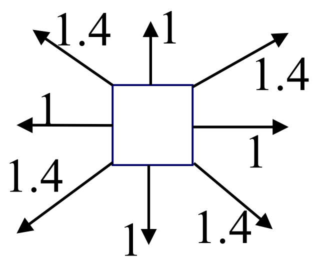


Starting with the start state (marked R),
always compute next the state with smallest g value!*

Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

8-connected grid



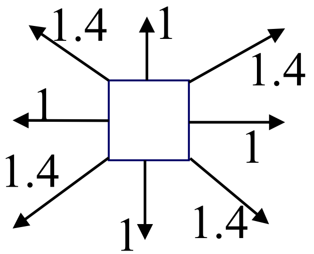
3.8	3.4	3.8	4.2	4.4	4.8
2.8	2.4	2.8	3.8	3.4	3.8
2.4	1.4			2.4	3.4
2	1	**0**	1	2	3

Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

Use g to compute the least-cost path*

8-connected grid



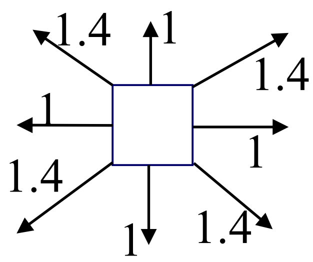
3.8	3.4	3.8	4.2	4.4	4.8
2.8	2.4	2.8	3.8	3.4	3.8
2.4	1.4			2.4	3.4
2	1	**0**	1	2	3

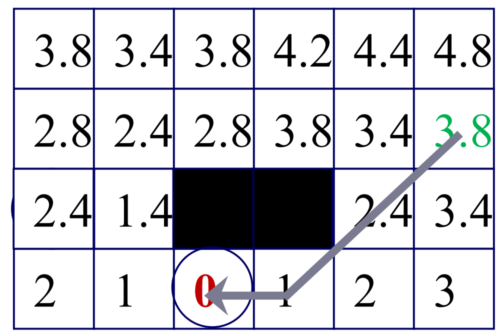
Searching Graphs for a Least-cost Path

- Example on a Grid-based Graph:

Use g to compute the least-cost path*

8-connected grid

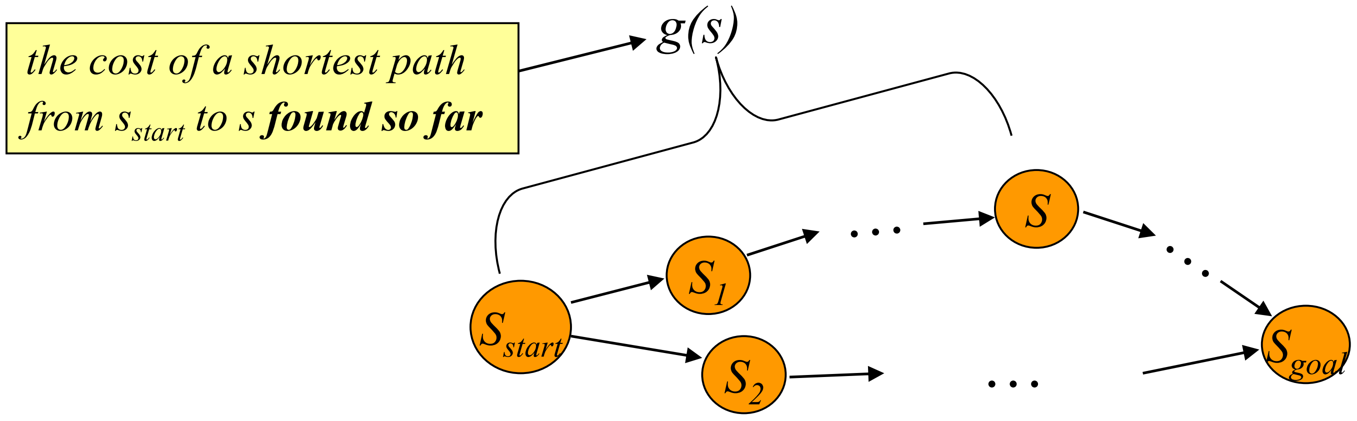




Uninformed A* Search

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

at any point of time:



Uninformed A* Search

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

Main function

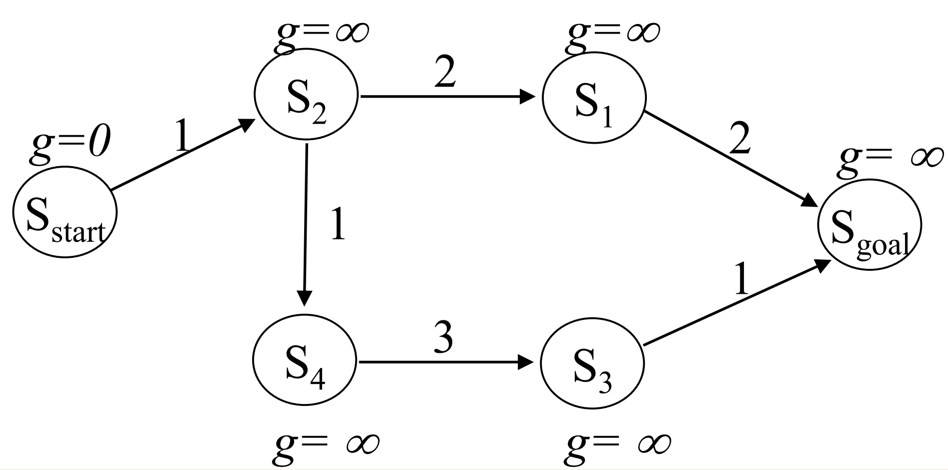
$g(s_{start}) = 0;$ all other g-values are infinite; $OPEN = \{s_{start}\};$
ComputePath();
publish solution; //compute least-cost path using g-values

set of candidates for expansion

ComputePath function

while(s_{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 ($g(s) = g^*(s)$)



Uninformed A* Search

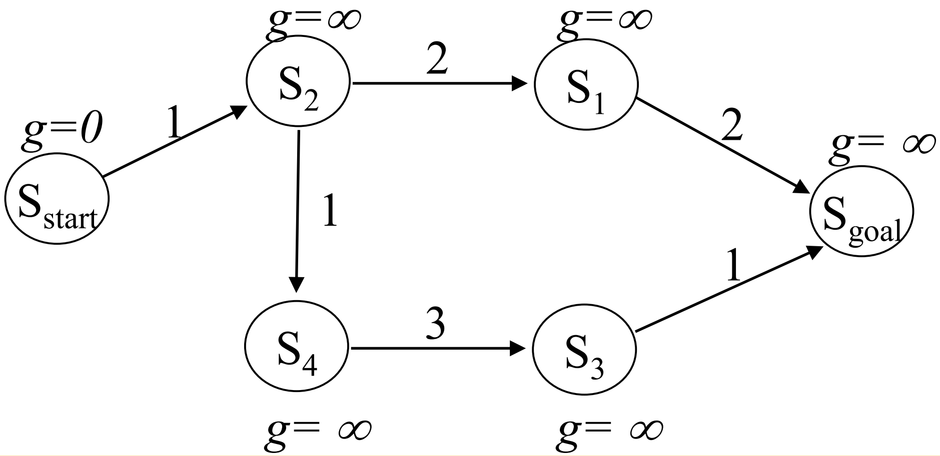
- Computes g*-values for **relevant** (not all) states

ComputePath function

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

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

  expand s;



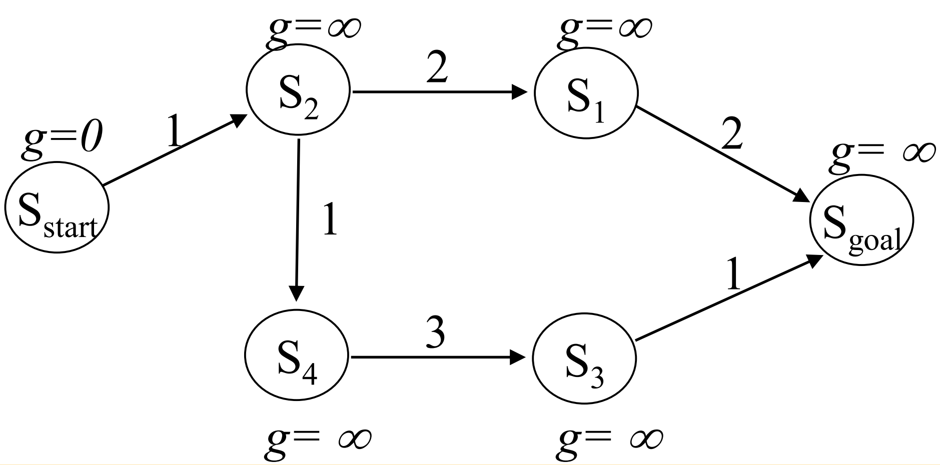
- Computes g*-values for **relevant** (not all) states

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 $OPEN$;

set of states that have already been expanded

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

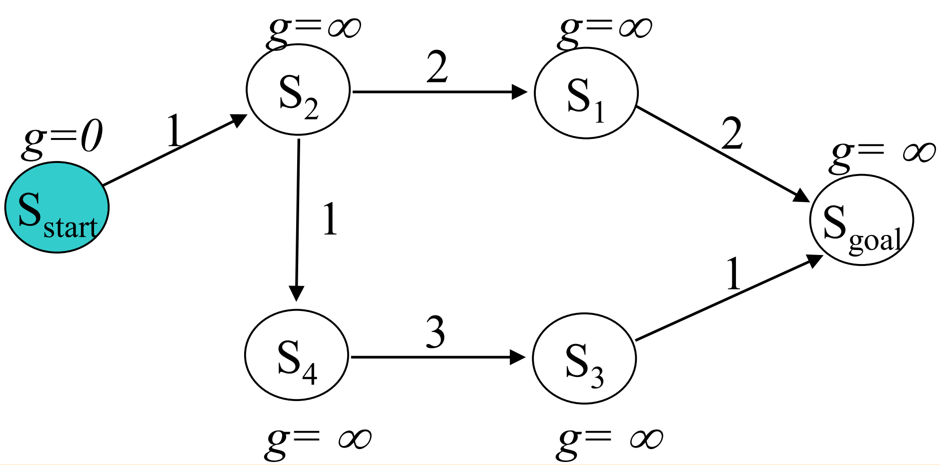
$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

$CLOSED = \{\}$

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

next state to expand: s_{start}



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

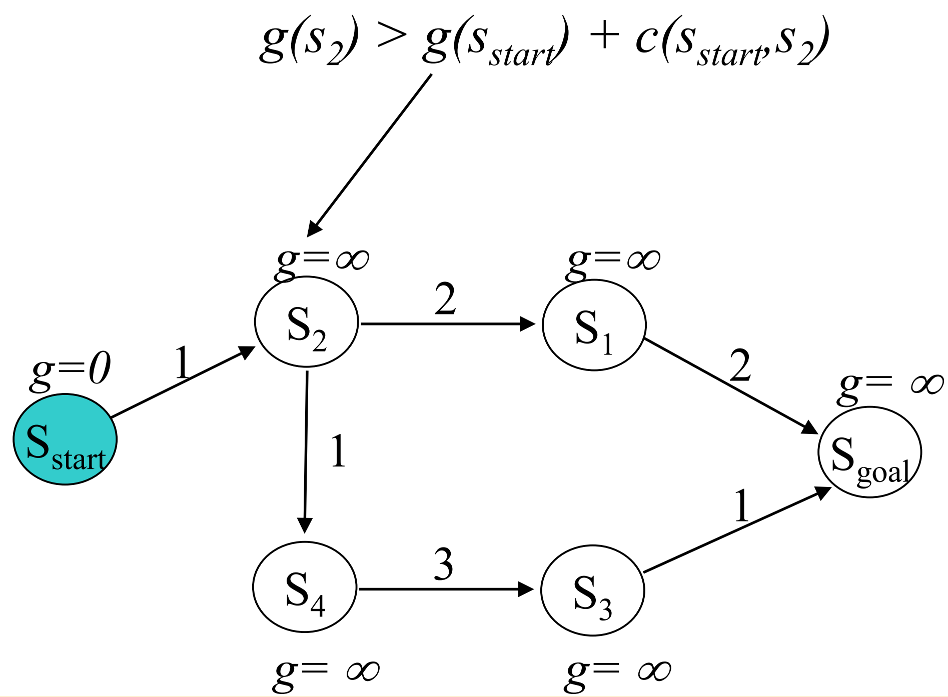
$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

$CLOSED = \{\}$

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

next state to expand: s_{start}



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 $OPEN$;

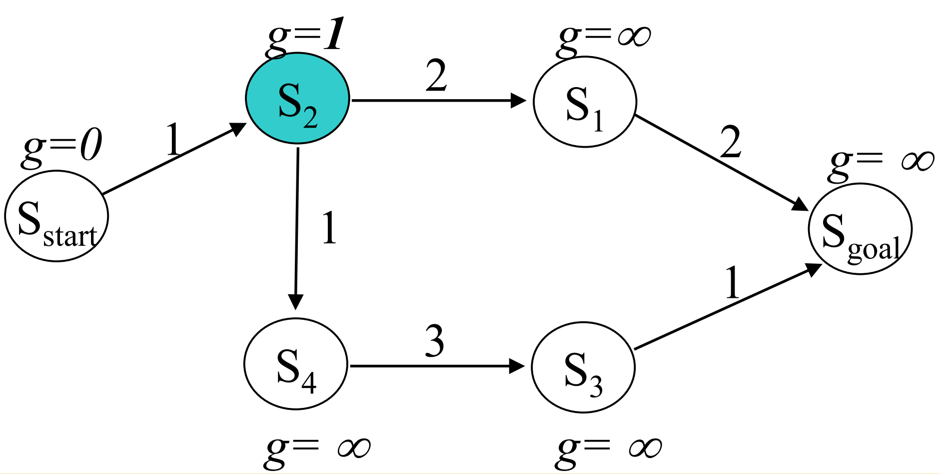
insert s into $CLOSED$;

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

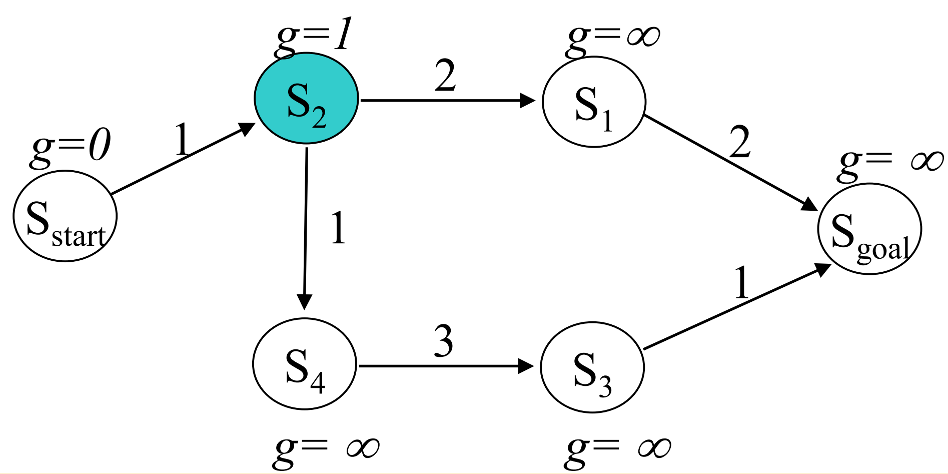
$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

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

$OPEN = \{s_2\}$

next state to expand: s_2



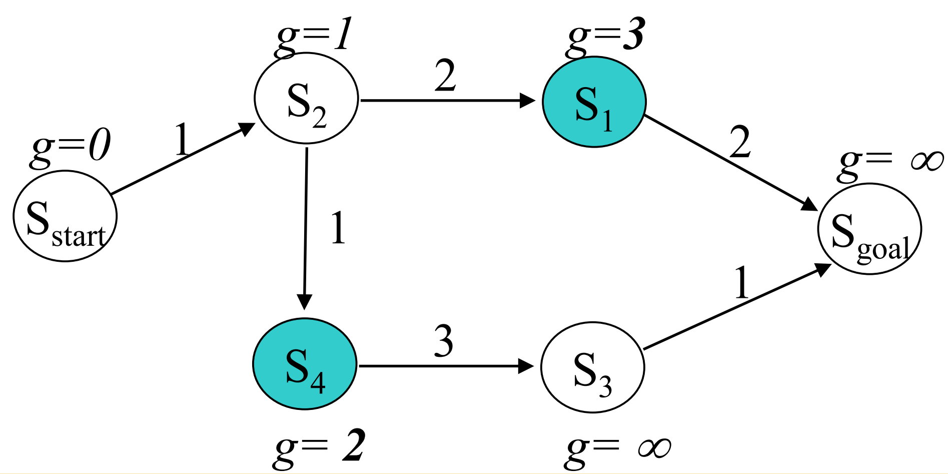
Uninformed A* Search

- 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: ?



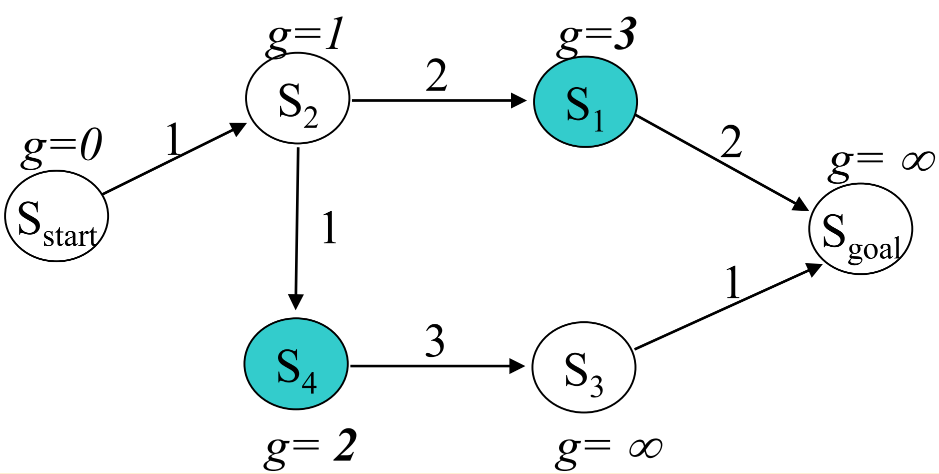
Uninformed A* Search

- 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: s_4



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

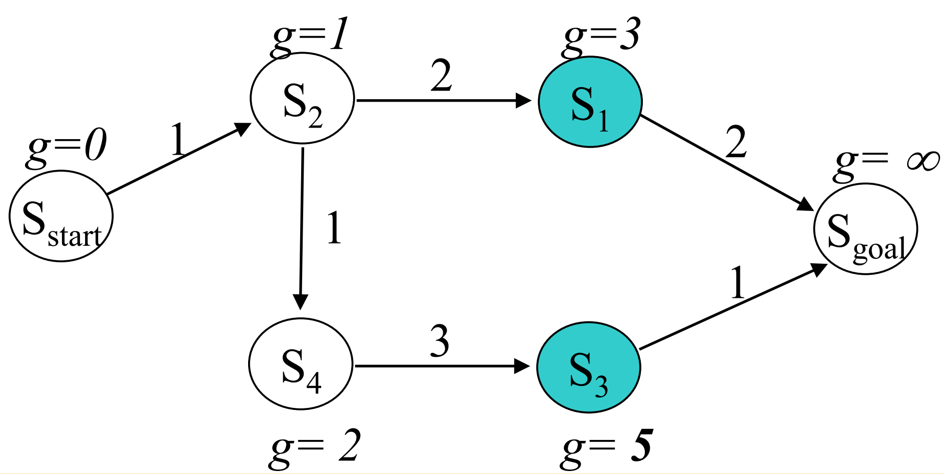
$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

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

$OPEN = \{s_1, s_3\}$

next state to expand: ?



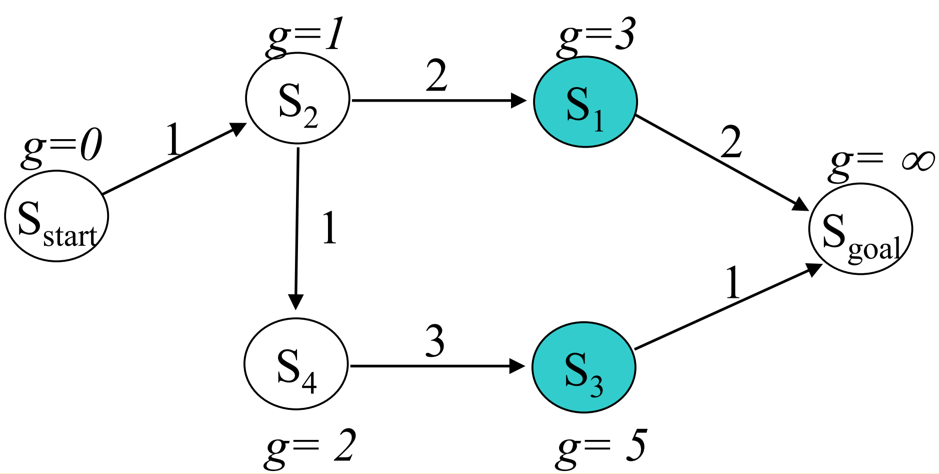
Uninformed A* Search

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

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: s_1



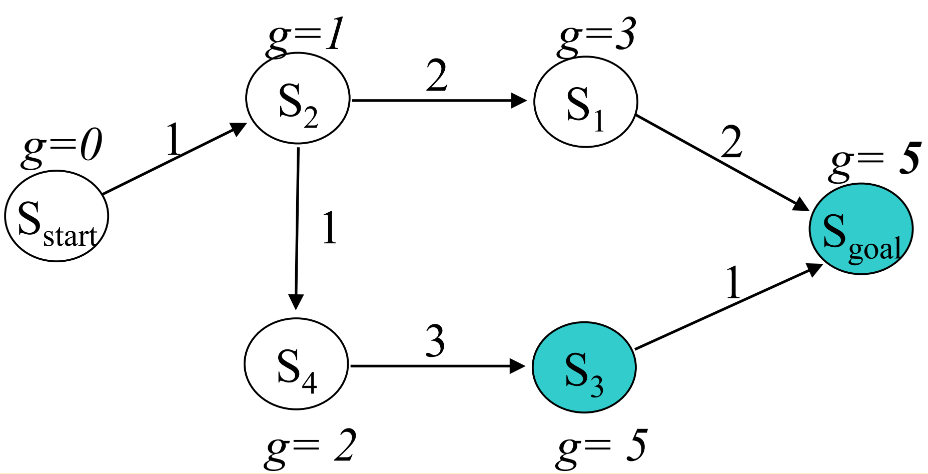
Uninformed A* Search

- 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: ?



Uninformed A* Search

- Computes g*-values f

Optional but useful optimization:
If OPEN contains multiple states with the smallest g-values and s_{goal} is one of them, then select s_{goal} for expansion

ComputePath function

while(s_{goal} is not expanded and

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

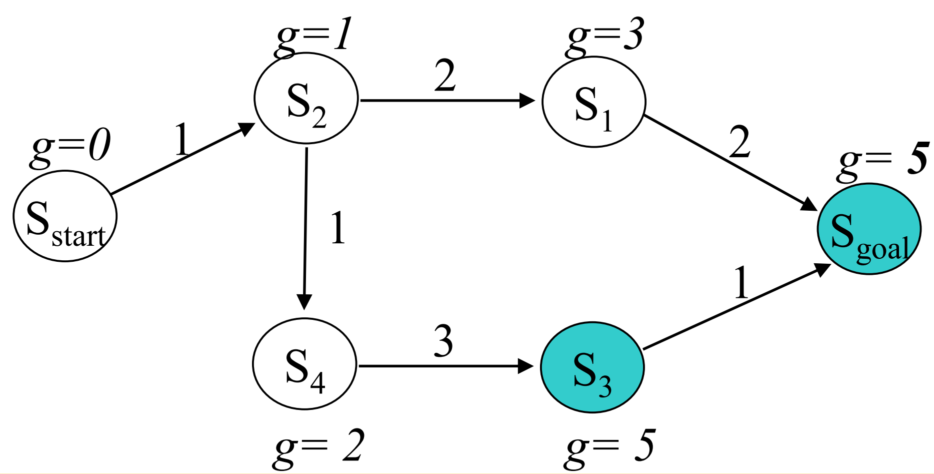
$g(s') = g(s) + c(s,s')$;

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

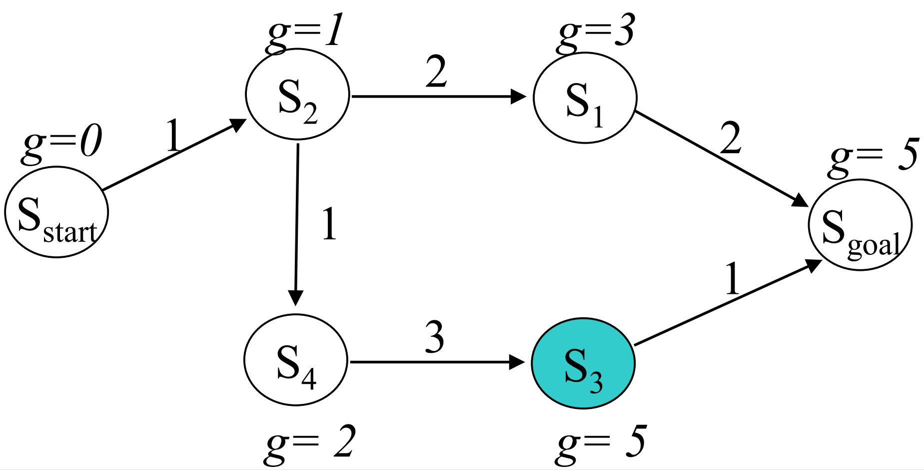
      $g(s') = g(s) + c(s,s')$;

      insert s' into $OPEN$;

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

$OPEN = \{s_3\}$

done

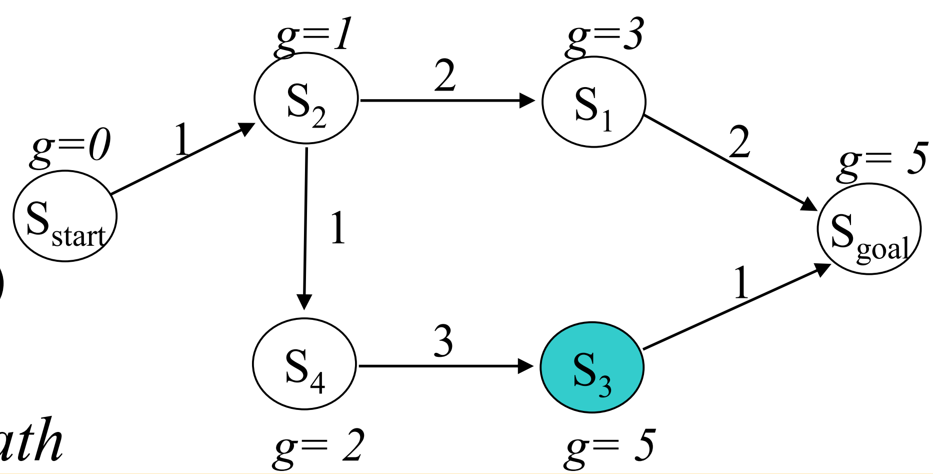


Uninformed A* Search

- 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 $OPEN$;



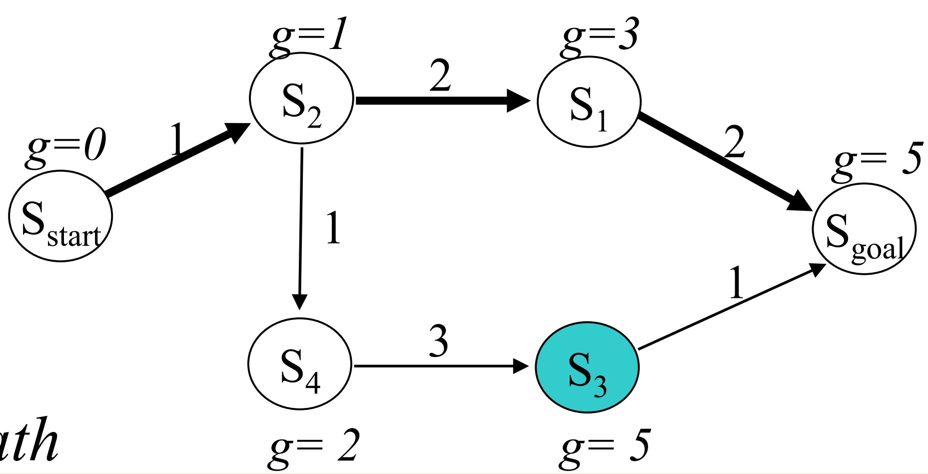
for every expanded state $g(s)=g^*(s)$
for every other state $g(s) \geq g^*(s)$
we can now compute a least-cost path

Uninformed A* Search

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

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

ComputePath function

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

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

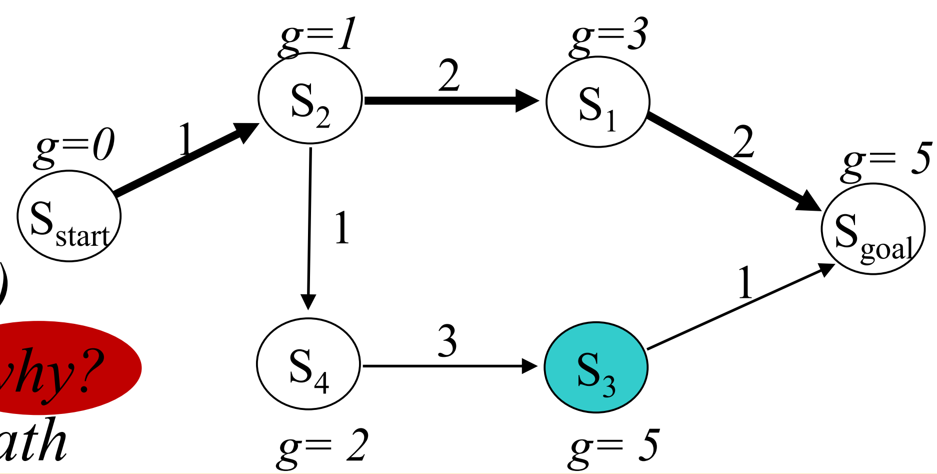
insert s into $CLOSED$;

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

Uninformed A* Search: Proofs

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*

Uninformed A* Search: Proofs

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

Sketch of proof:

Proof?

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 moves to the predecessor state that continues to be on a least-cost path (because all other predecessors have g-values that are strictly larger)*

Summary

- Given g*-values, one can re-construct a least-cost path
- Uninformed A* computes g*-values for relevant states efficiently
- Properties of uninformed A* search
  - g-values of expanded states are optimal (g=g*)