16-350
Planning Techniques for Robotics

Interleaving Planning and Execution: Anytime Heuristic Search

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

Planning during Execution

- Planning is a repeated process!

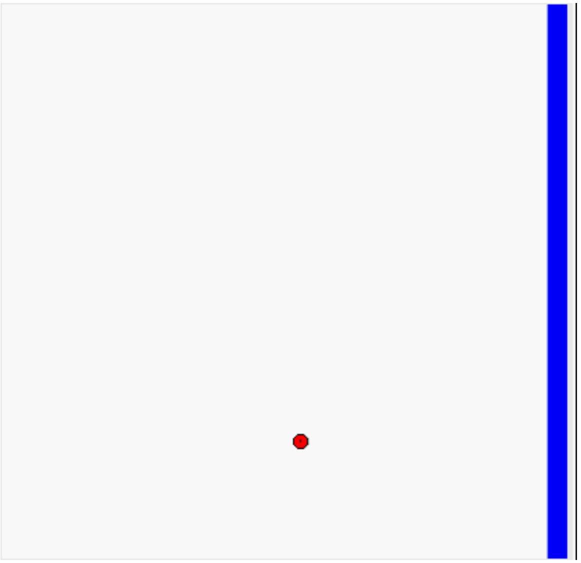
Reasons?

Planning during Execution

- Planning is a <u>repeated</u> process!
  - partially-known environments
  - dynamic environments
  - imperfect execution of plans
  - imprecise localization

ATRV navigating initially-unknown environment

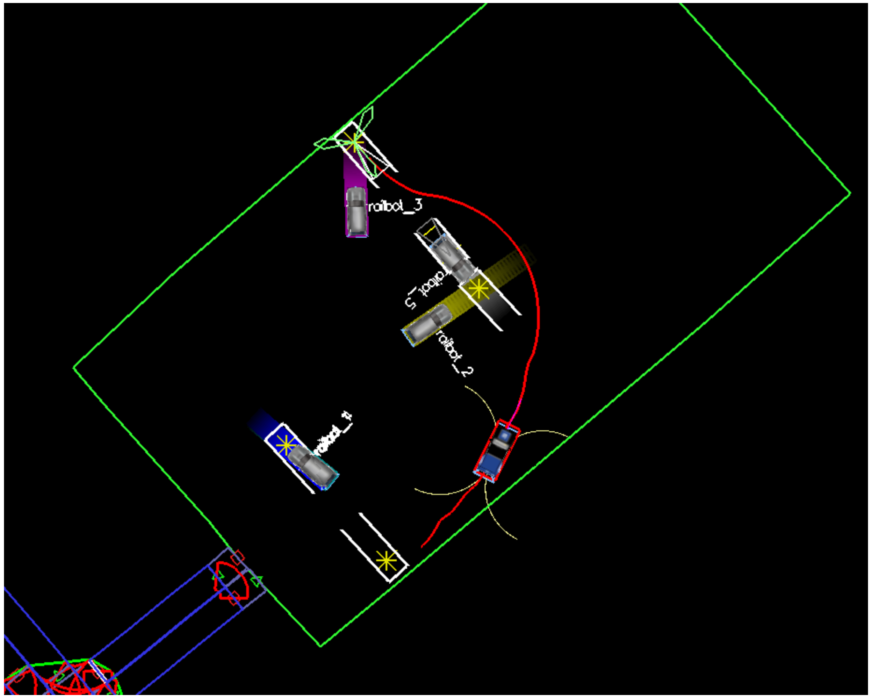
planning map and path



Planning during Execution

- Planning is a repeated process!
  - partially-known environments
  - dynamic environments
  - imperfect execution of plans
  - imprecise localization

planning in dynamic environments



Planning during Execution

- Planning is a <u>repeated</u> process!
  - partially-known environments
  - dynamic environments
  - imperfect execution of plans
  - imprecise localization

- Need to be able to re-plan fast!

- Several methodologies to achieve this:
  - anytime heuristic search: return the best plan possible within T msecs
  - incremental heuristic search: speed up search by reusing previous efforts
  - real-time heuristic search: plan few steps towards the goal and re-plan later

Planning during Execution

- Planning is a repeated process!
  - partially-known environments
  - dynamic environments
  - imperfect execution of plans
  - imprecise localization

- Need to be able to re-plan fast!

- Several methodologies to achieve this:
  - anytime heuristic search: return the best plan possible within T msecs
  - incremental heuristic search: speed up search by reusing previous efforts
  - real-time heuristic search: plan few steps towards the goal and re-plan later

this class

next two classes

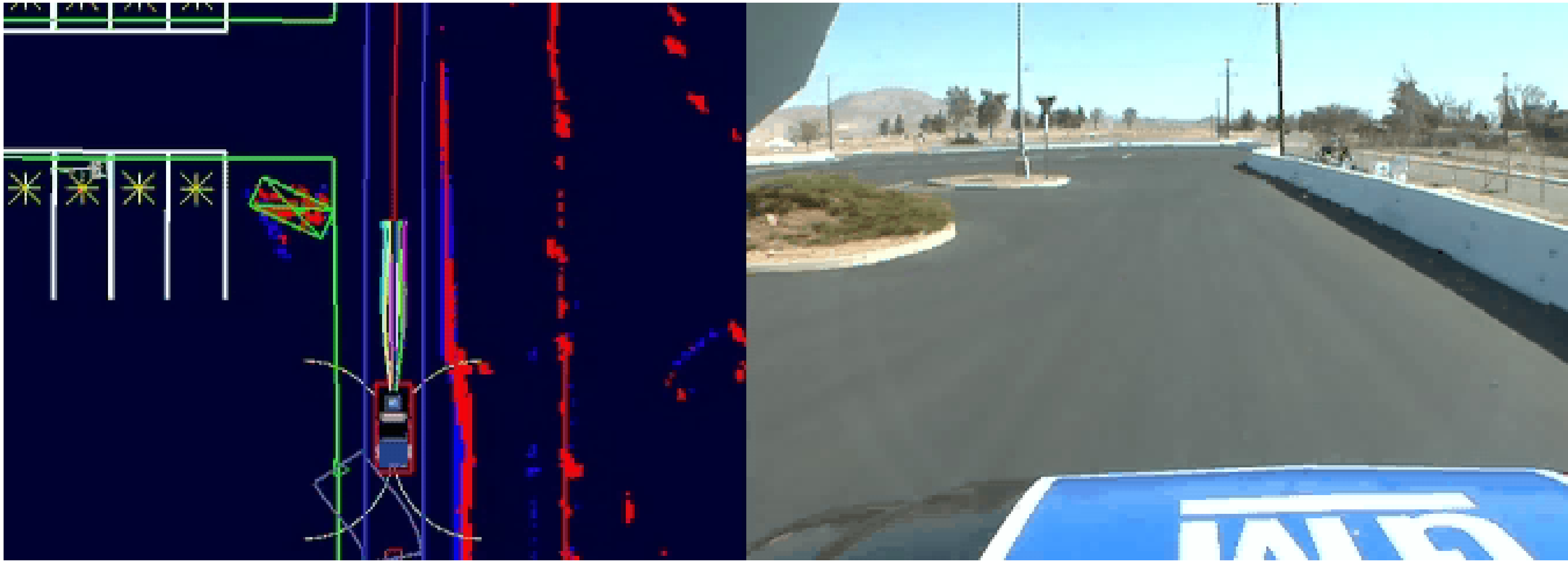
Anytime Algorithms

- Anytime algorithms are algorithms that are:
  - capable of returning **some** solution whenever they are interrupted
  - improve the solution over time until they are interrupted or until convergence to an optimal solution, whichever is first

- Anytime Planners
  - capable of returning some plans whenever they are interrupted
  - improve the plans over time until they are interrupted or until convergence to an optimal plan

Anytime Planning for an Autonomous Vehicle

- Running ARA* Search



Anytime Heuristic Search: Straw Man Approach

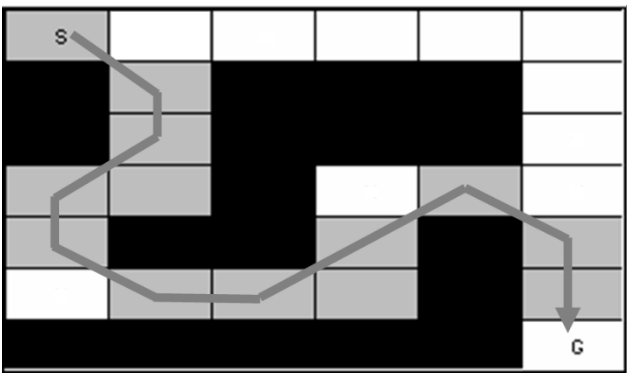
- Constructing anytime search based on weighted A*:
  - find the best path possible given some amount of time for planning
  - do it by running a series of weighted A* searches with decreasing ε:

Any ideas?

Anytime Heuristic Search: Straw Man Approach

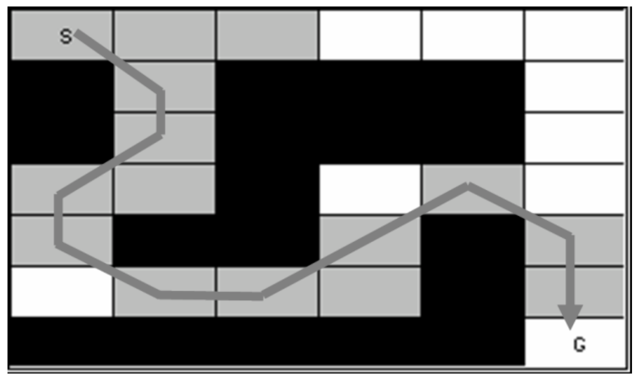
- Constructing anytime search based on weighted A*:
  - find the best path possible given some amount of time for planning
  - do it by running a series of weighted A* searches with decreasing *ε*:

ε =2.5



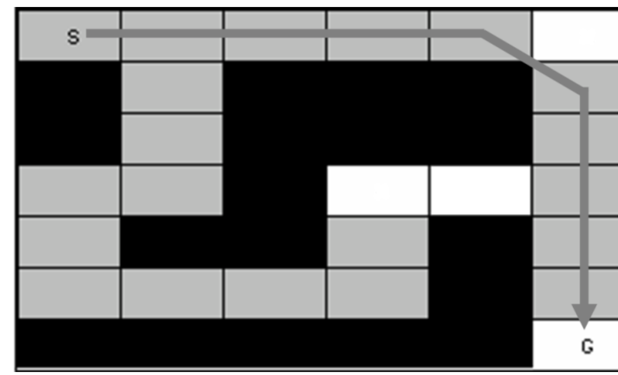
13 expansions
solution=11 moves

ε =1.5



15 expansions
solution=11 moves

ε =1.0

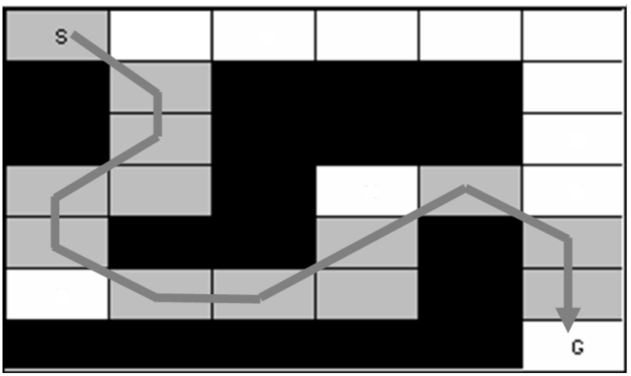


20 expansions
solution=10 moves

Anytime Heuristic Search: Straw Man Approach

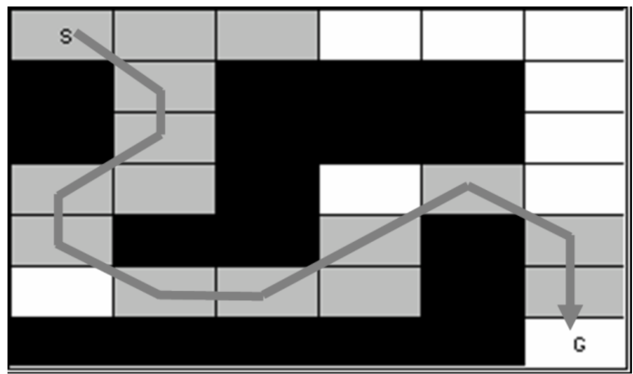
- Constructing anytime search based on weighted A*:
  - find the best path possible given some amount of time for planning
  - do it by running a series of weighted A* searches with decreasing ε:

ε =2.5



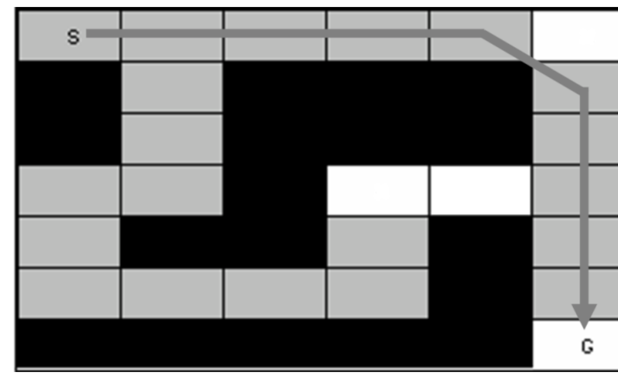
13 expansions
solution=11 moves

ε =1.5



15 expansions
solution=11 moves

ε =1.0



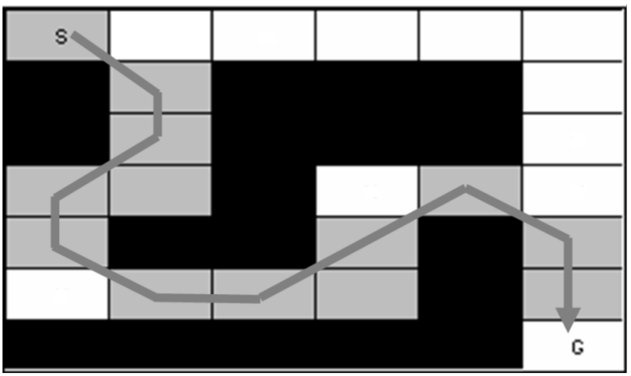
20 expansions
solution=10 moves

- Inefficient because
  – many state values remain the same between search iterations
  – we should be able to reuse the results of previous searches

Anytime Heuristic Search: Straw Man Approach

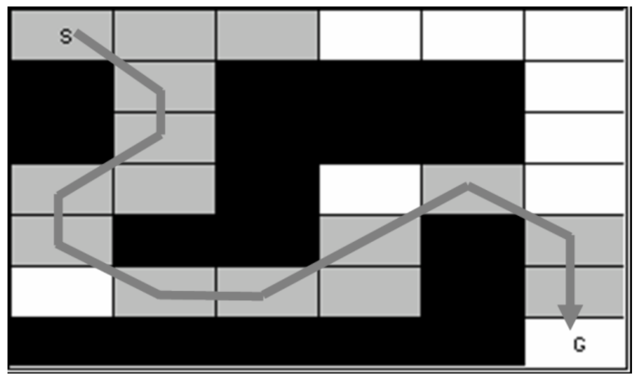
- Constructing anytime search based on weighted A*:
  - find the best path possible given some amount of time for planning
  - do it by running a series of weighted A* searches with decreasing *ε:*

ε =2.5



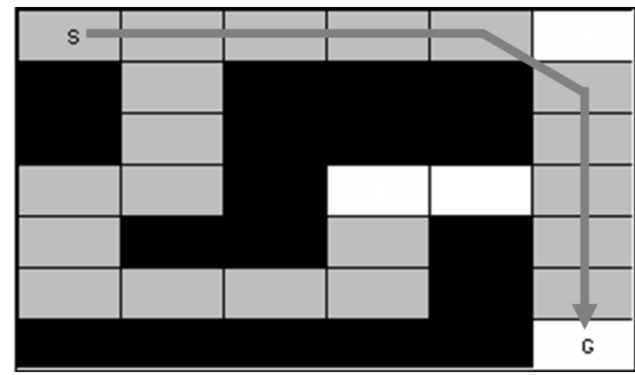
13 expansions
solution=11 moves

ε =1.5



15 expansions
solution=11 moves

ε =1.0



20 expansions
solution=10 moves

- ARA* (Anytime Repairing A*)
  - efficient version of above that reuses state values between iterations

A* with Reuse of State Values

- Alternative view of A*

all v-values initially are infinite;

ComputePath function

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

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

insert s into $CLOSED$;

$v(s)=g(s)$;

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* with Reuse of State Values

- Alternative view of A*

all *v*-values initially are infinite;

ComputePath function

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

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

  insert *s* into *CLOSED*;

  $v(s)=g(s);$

  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*;

v-value – the value of a state during its expansion (infinite if state was never expanded)

A* with Reuse of State Values

- Alternative view of A*

all v-values initially are infinite;

ComputePath function

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

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

  insert s into *CLOSED*;

  $v(s)=g(s);$

  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*;

- $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$

A* with Reuse of State Values

- Alternative view of A*

all *v*-values initially are infinite;

ComputePath function

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

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

insert *s* into *CLOSED*;

$v(s)=g(s);$

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*;

- $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$
- *OPEN*: a set of states with $v(s) > g(s)$

  all other states have $v(s) = g(s)$

overconsistent state

consistent state

A* with Reuse of State Values

- Alternative view of A*

all v-values initially are infinite;

ComputePath function

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

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

  insert s into $CLOSED$;

  $v(s)=g(s);$

  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 $OPEN$;

- $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$
- $OPEN$: a set of states with $v(s) > g(s)$
  all other states have $v(s) = g(s)$
- A* expands overconsistent states in the order of their f-values

A* with Reuse of State Values

- Making A* reuse old values:

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum f-value in *OPEN*)

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

insert s into *CLOSED*;

$v(s)=g(s);$

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*;

all you need to do to make it reuse old values!

- $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$
- *OPEN*: a set of states with $v(s) > g(s)$

  all other states have $v(s) = g(s)$
- A* expands overconsistent states in the order of their f-values

A* with Reuse of State Values

- Making A* reuse old values:

Why do we need this change?

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum *f*-value in *OPEN*)

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

insert *s* into *CLOSED*;

$v(s)=g(s);$

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

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

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

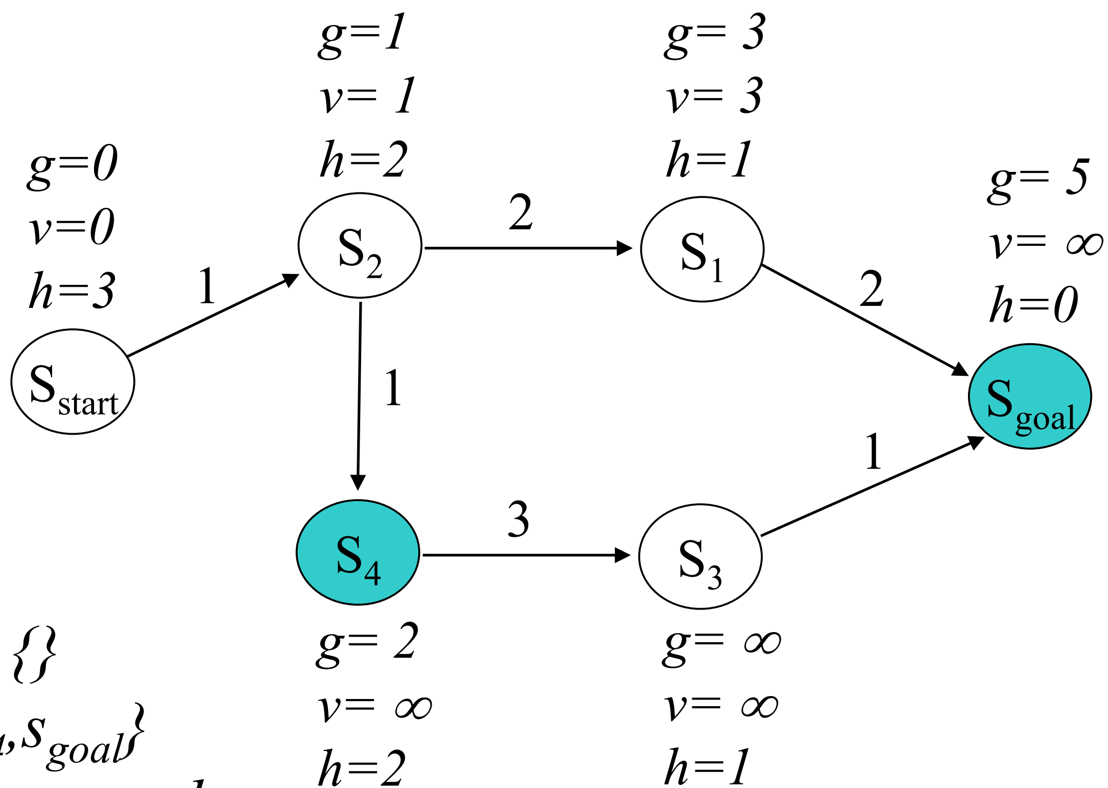
insert *s'* into *OPEN*;

all you need to do to make it reuse old values!

- $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$
- *OPEN*: a set of states with $v(s) > g(s)$

  all other states have $v(s) = g(s)$
- A* expands overconsistent states in the order of their f-values

A* with Reuse of State Values



CLOSED = {}

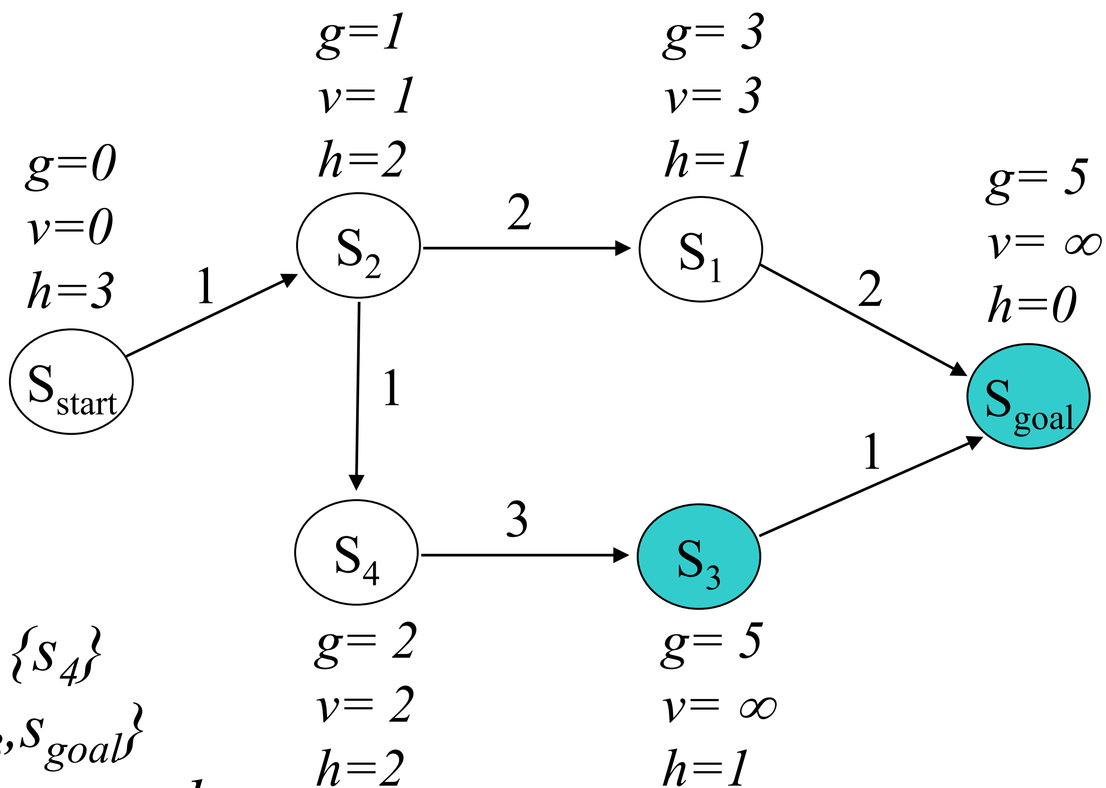
OPEN = $\{s_4, s_{goal}\}$

next state to expand: s_4

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

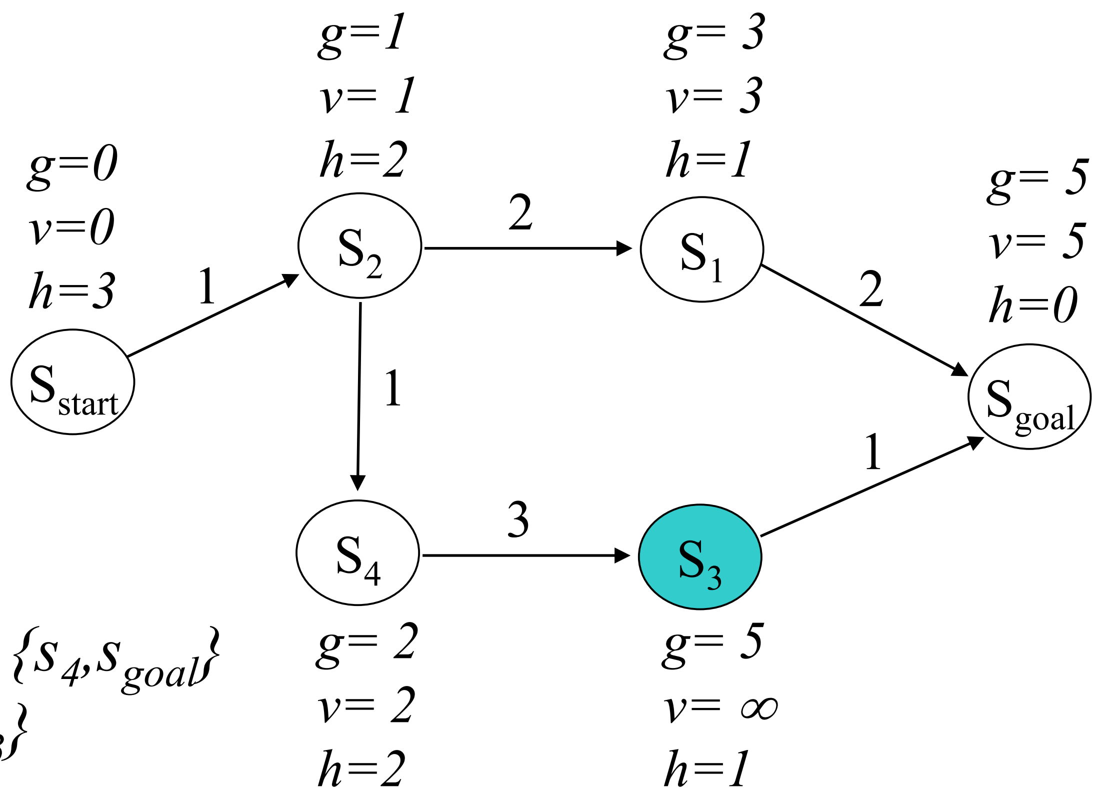
initially OPEN contains all overconsistent states

A* with Reuse of State Values



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

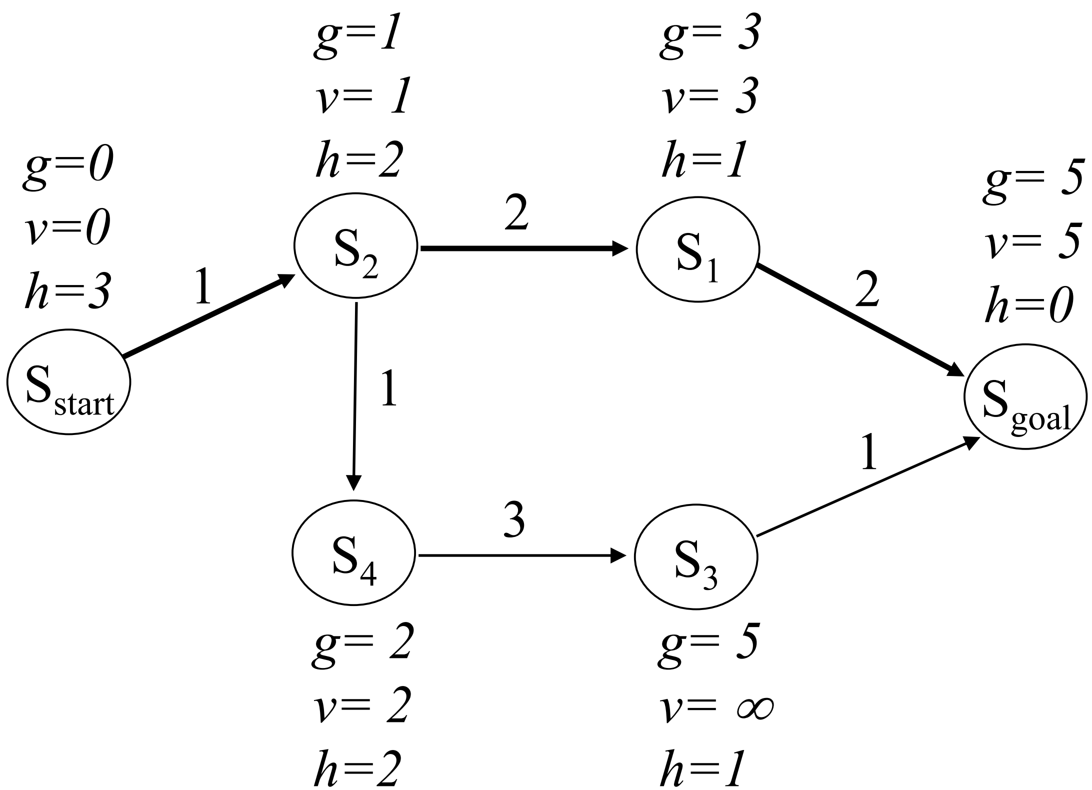
A* with Reuse of State Values



CLOSED = $\{s_4, s_{goal}\}$
OPEN = $\{s_3\}$
done

after ComputePathwithReuse terminates:
all g-values of states are equal to final A g-values*

A* with Reuse of State Values



we can now compute a least-cost path

A* with Reuse of State Values

- Making weighted A* reuse old values:

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum f-value in *OPEN*)

remove s with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;

insert s into *CLOSED*;

$v(s)=g(s);$

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*;

*the exact same thing as with A**

A* with Reuse of State Values

- Making weighted A* reuse old values:

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum f-value in *OPEN*)

remove s with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;

insert s into *CLOSED*;

$v(s)=g(s)$;

for every successor s' of s

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

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

if s' not in *CLOSED* then insert s' into *OPEN*;

*the exact same thing as with A**

To maintain the invariant:
$g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$

Anytime Repairing A* (ARA*)

- Efficient series of weighted A* searches with decreasing ε:

set ε to large value;

$g(s_{start}) = 0$; v-values of all states are set to infinity; $OPEN = \{s_{start}\}$;

while $\varepsilon \geq 1$

    $CLOSED = \{\}$;

    ComputePathwithReuse();

    publish current ε suboptimal solution;

    decrease ε;

    initialize *OPEN* with all overconsistent states;

ARA*

- Efficient series of weighted A* searches with decreasing ε:

set ε to large value;

$g(s_{start}) = 0$; v-values of all states are set to infinity; $OPEN = \{s_{start}\}$;

while $\varepsilon \geq 1$

  $CLOSED = \{\}$;

  ComputePathwithReuse();

  publish current ε suboptimal solution;

  decrease ε;

  initialize $OPEN$ with all overconsistent states;

need to keep track of those

ARA*

- Efficient series of weighted A* searches with decreasing ε:

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum f-value in *OPEN*)

remove s with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;

insert s into *CLOSED*;

$v(s)=g(s);$

for every successor s' of s

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

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

if s' not in *CLOSED* then insert s' into *OPEN*;

Does OPEN contain ALL overconsistent states (i.e., states s' whose $v(s') > g(s')$)?

ARA*

- Efficient series of weighted A* searches with decreasing ε:

initialize *OPEN* with all overconsistent states;

ComputePathwithReuse function

while($f(s_{goal})$ > minimum f-value in *OPEN*)

  remove s with the smallest $[g(s)+ \varepsilon h(s)]$ from *OPEN*;

  insert s into *CLOSED*;

  $v(s)=g(s);$

  for every successor s' of s

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

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

      if s' not in *CLOSED* then insert s' into *OPEN*;

      otherwise insert s' into *INCONS*

- *OPEN U INCONS* = all overconsistent states

ARA*

- Efficient series of weighted A* searches with decreasing ε:

set ε to large value;

$g(s_{start}) = 0$; v-values of all states are set to infinity; $OPEN = \{s_{start}\}$;

while $\varepsilon \geq 1$

    $CLOSED = \{\}$; $INCONS = \{\}$;

    ComputePathwithReuse();

    publish current ε suboptimal solution;

    decrease ε;

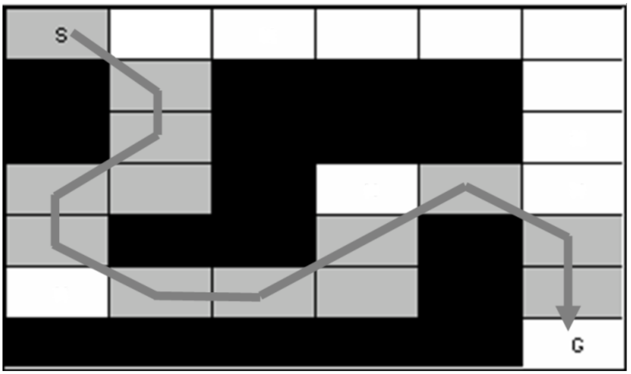
    initialize $OPEN = OPEN \cup INCONS$;

all overconsistent states (exactly what we need!)

ARA*

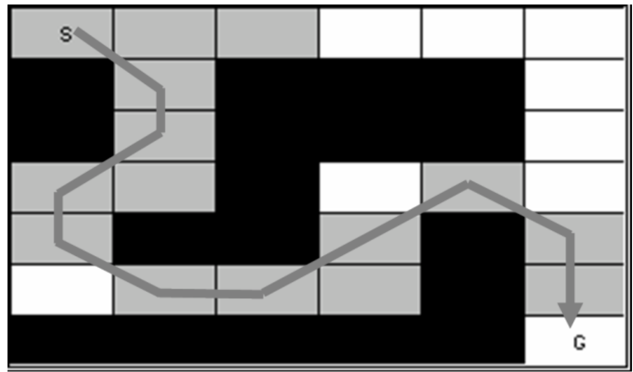
- **A series of weighted A* searches**

ε =2.5



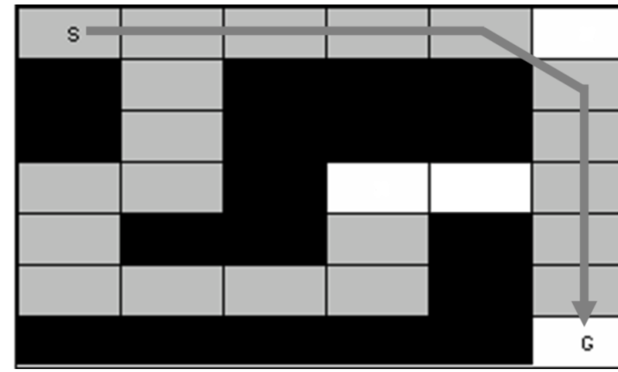
13 expansions
solution=11 moves

ε =1.5



15 expansions
solution=11 moves

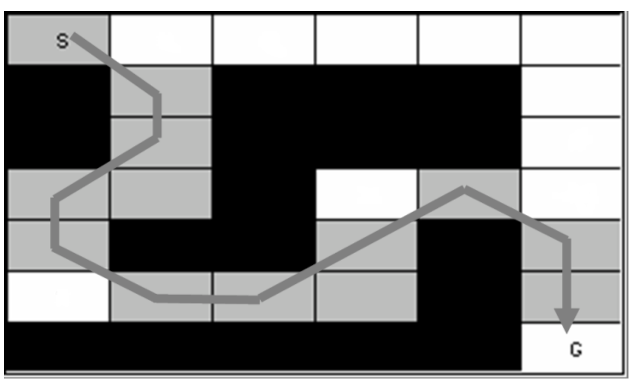
ε =1.0



20 expansions
solution=10 moves

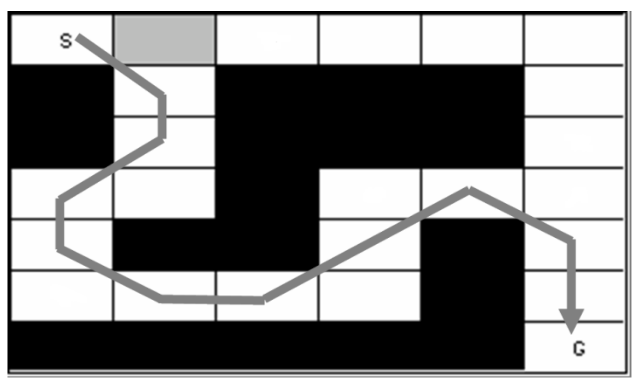
- **ARA***

ε =2.5



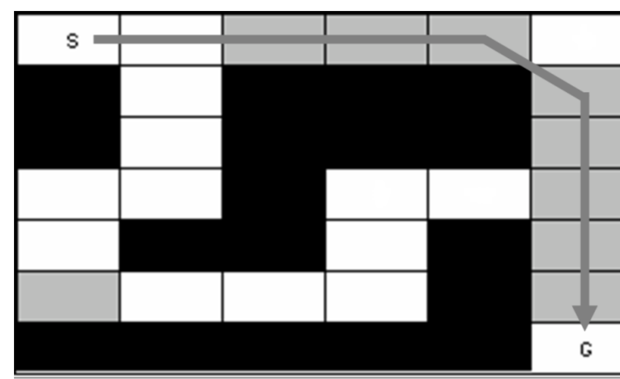
13 expansions
solution=11 moves

ε =1.5



1 expansion
solution=11 moves

ε =1.0



9 expansions
solution=10 moves

ARA*

- Simple example on the board!

What You Should Know…

- Reasons for repeated planning
- What are anytime algorithms, anytime planners
- How ARA* operates
- Theoretical properties of ARA*