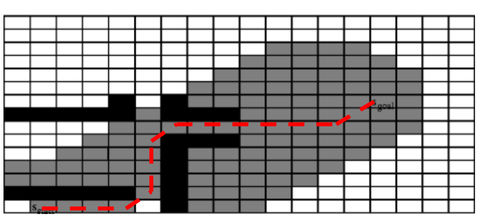
Path/Motion Planning III

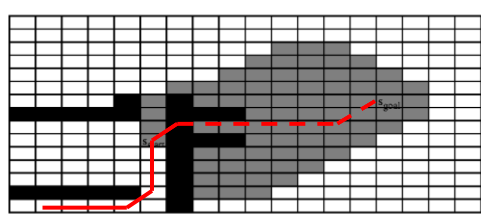
Examples in this lecture from Max Likhachev

Two related extensions

1. *Online/incremental:*

- World model changes as the path from start to goal is executed
- Naïve approach: Replan from scratch every time there is a change (!)



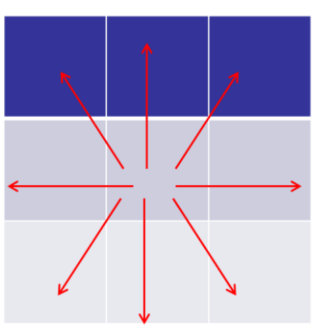


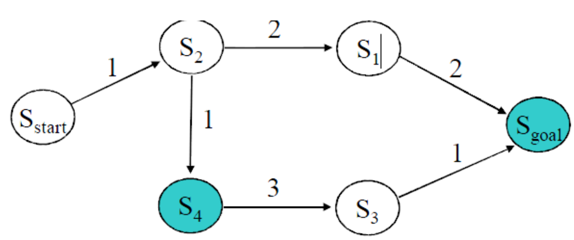
Two related extensions

2. Anytime:

- Must be able to work with a sound path from start to goal within time *T* (even if not optimal)
- Can update path as time passes
- Get sound path at *anytime*
- Get optimal path eventually

- Back to "simple" discrete setup for this lecture (for a while)
- Discrete grid or graph of discrete states



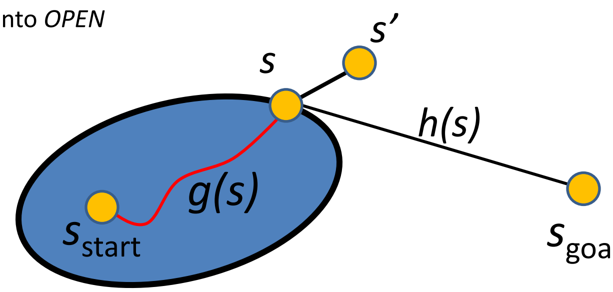


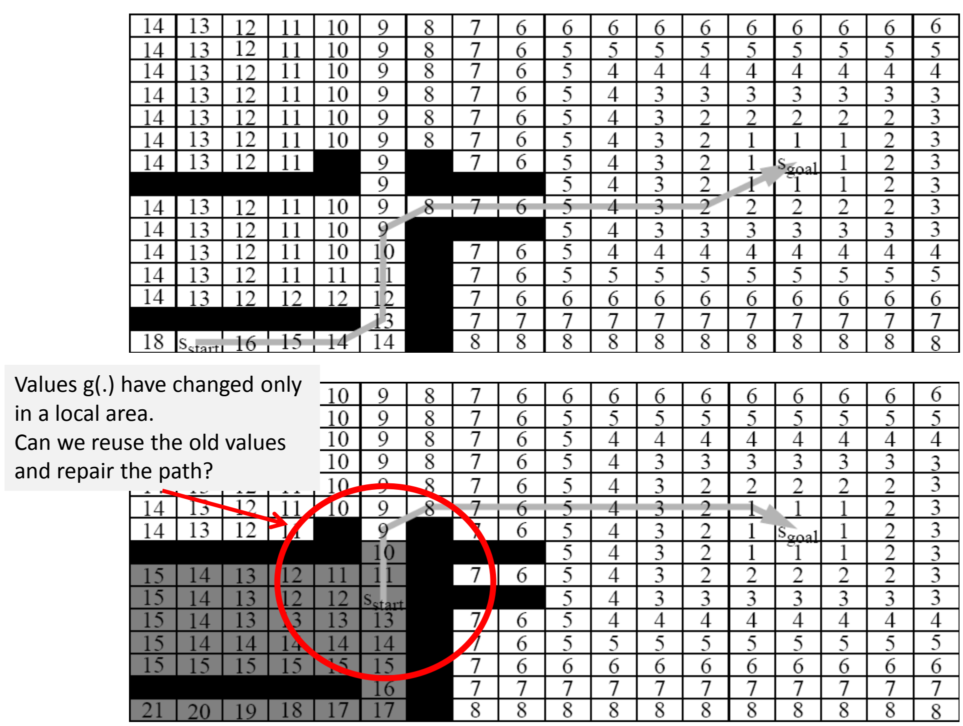
c(s,s') =

- Infinity if blocked cell
- Distance or traversal cost otherwise

A*

- **ComputePath function**
- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]*from *OPEN*;
  - for every successor *s'* of *s*
    - if *g(s') > g(s) + c(s,s')*
      - *g(s') = g(s) + c(s,s')*
      - insert *s'* into *OPEN*

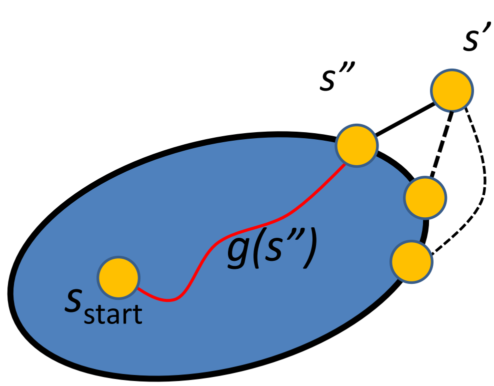




A*: Reusing previous values

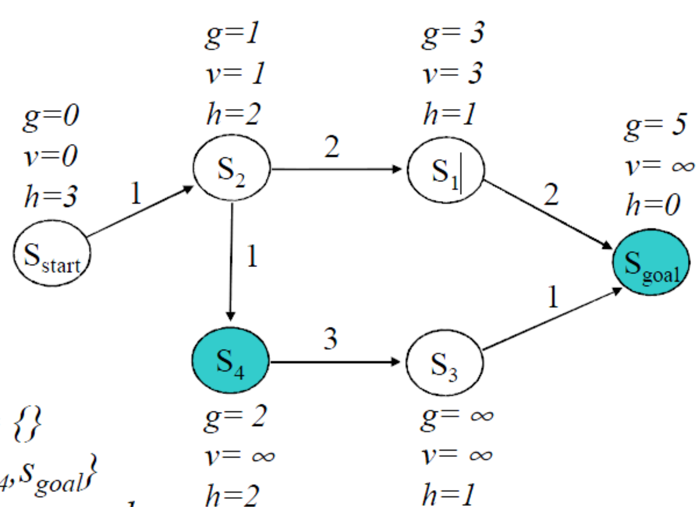
- *v(s) = infinite*
- **ComputePath function**
- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]*from *OPEN*;
  - *v(s) = g(s)*
  - for every successor *s'*of *s*
    - if *g(s') > g(s) + c(s,s')*
      - *g(s') = g(s) + c(s,s')*
      - insert *s'* into *OPEN*

- During A*, a node *s'* must satisfy:
  - $g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$
- If *v(s) > g(s)* then *s is* inconsistent with its neighbors (**over-consistent**)
- Property:
- The OPEN list is the set of over-consistent nodes
- A* expands overconsistent states in the order of *f(.) = g(.) + h(.)*



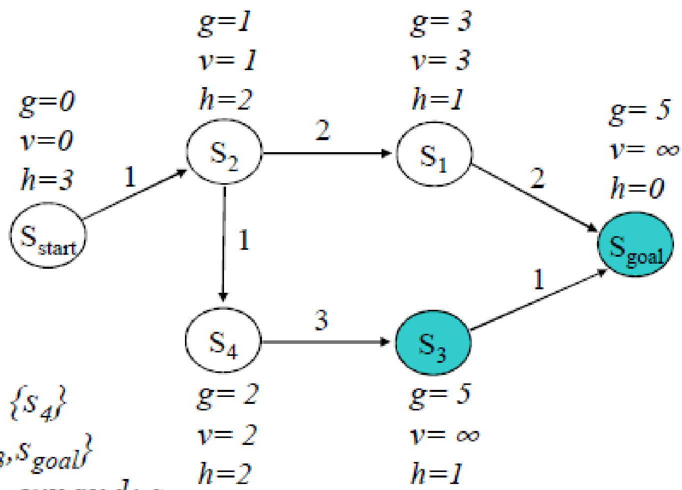
A*: Reusing previous values

- ~~*v(s) = infinite*~~
- *OPEN* = set of states such that $v(s) > g(s)$
- **ComputePathReuse function**
- while(s_{goal} is not expanded)
  - remove *s* with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  - $v(s) = g(s)$
  - 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*

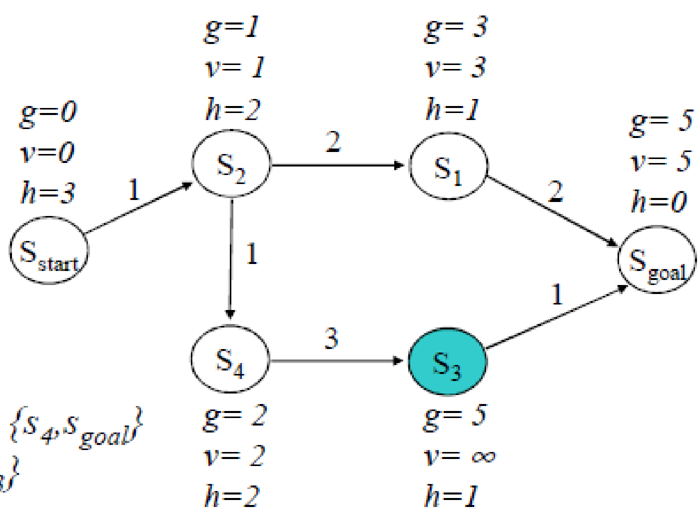


CLOSED = {}
OPEN = $\{s_4, s_{goal}\}$
next state to expand: s_4

- $g(s) = \min_{t \in pred(s)} v(t) + c(t,s)$
- OPEN = *s* such that $v(s) > g(s)$



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



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

Example: Repeated weighted A*

- Idea:
  - First plan with heuristic $\varepsilon h(.)$ instead of $h(.)$
  - Choose ε large → Few expansions → Really fast
  - Progressively decrease ε
- Key insight from previous slides: Can do that without recomputing all the values from scratch for each ε

Weighted A*

- **ComputePathReuse**
- while(s_{goal} is not expanded)
  - remove s with the smallest $[f(s) = g(s)+\varepsilon h(s)]$from *OPEN*;
  - $v(s) = g(s)$
  - 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*

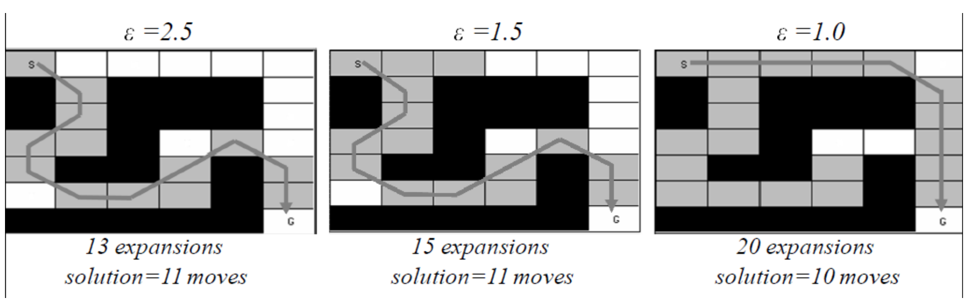
New conditional

ARA*: Anytime Repairing A*

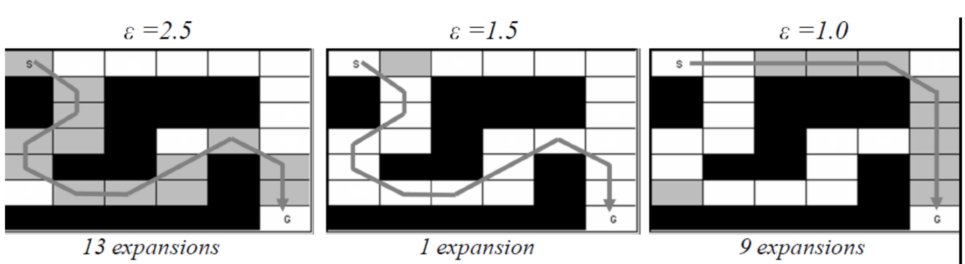
- *OPEN* = set of over-consistent states $v(s) > g(s)$
- **ComputePath function**
- while(s_{goal} is not expanded)
  - remove *s* with the smallest $[f(s) = g(s)+\varepsilon h(s)]$from *OPEN*;
  - $v(s) = g(s)$
  - insert *s* into *CLOSED*;
  - for every successor *s'* of *s* such that *s'* not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$
      - insert *s'* into *OPEN*

Example

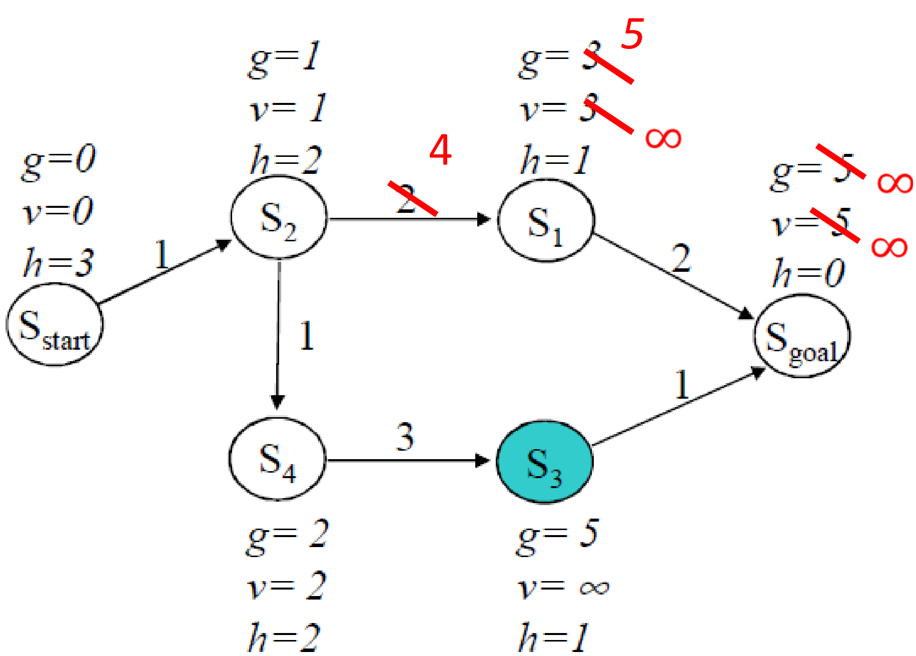
No reuse:

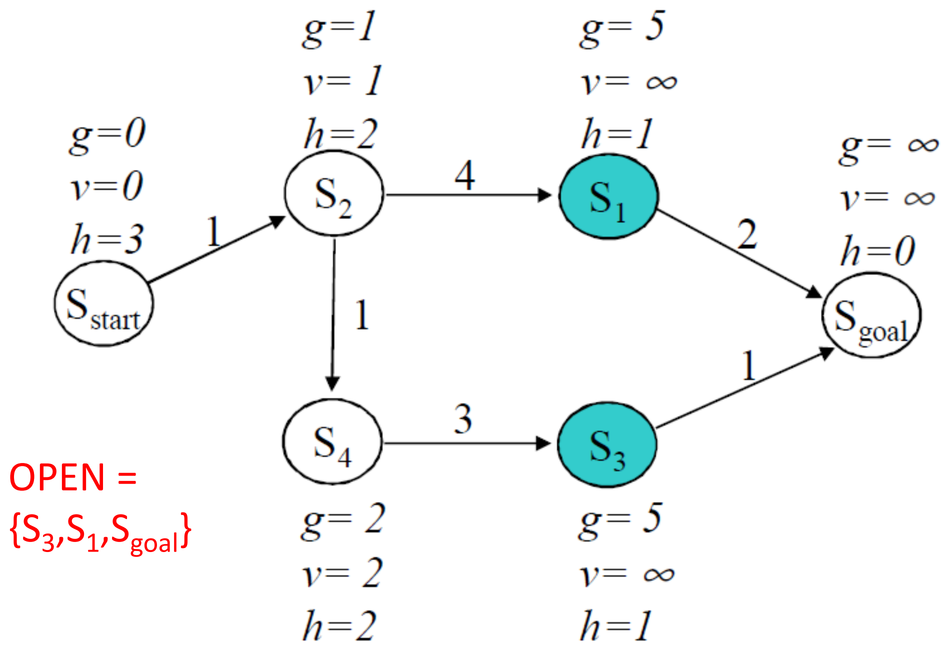


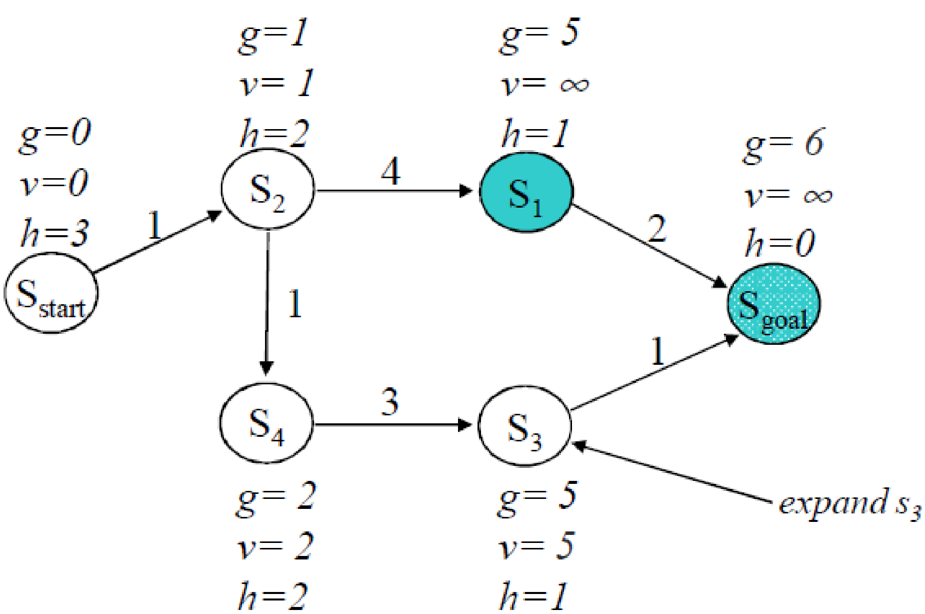
With reuse:

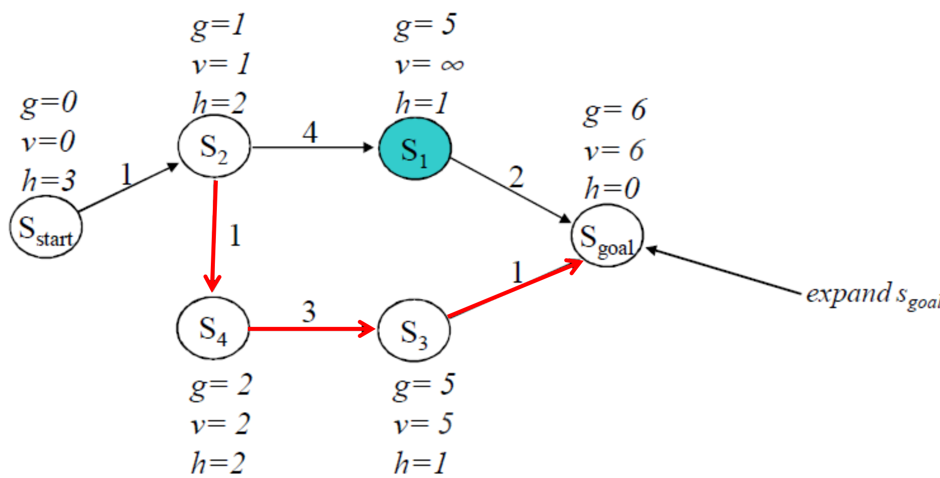


- This takes care of the inconsistent nodes such that *v(s) > g(s)*
- What if *v(s) < g(s)* ?
- Can happen if edge cost changes
- Solution:
  - Set *v(s)* to infinity
  - Propagate through connected nodes



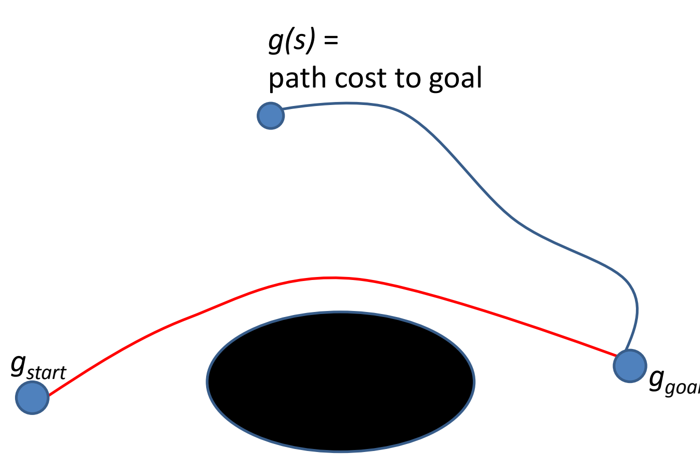

g=1
v= 1
h=2
g= 5
v= ∞
h=1
g=0
v=0
h=3
g= ∞
v= ∞
h=0
S2
S1
Sstart
Sgoal
S4
S3
4
1
2
1
1
3
OPEN =
{S3,S1,Sgoal}
g= 2
v= 2
h=2
g= 5
v= ∞
h=1

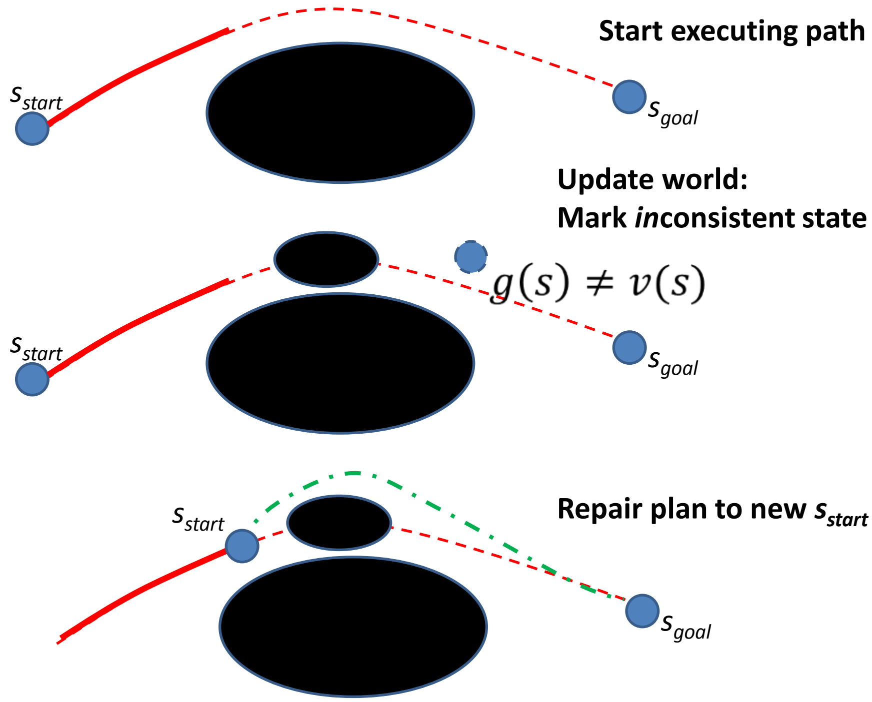

g=1
v= 1
h=2
g= 5
v= ∞
h=1
g=0
v=0
h=3
g= 6
v= ∞
h=0
S2
S1
Sstart
Sgoal
S4
S3
4
1
2
1
1
3
expand s3
g= 2
v= 2
h=2
g= 5
v= 5
h=1



Incremental/Online planning: D*

- Plan from goal to start so that most of the *g(.)* remain the same



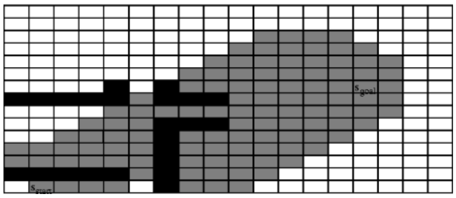


D* Lite

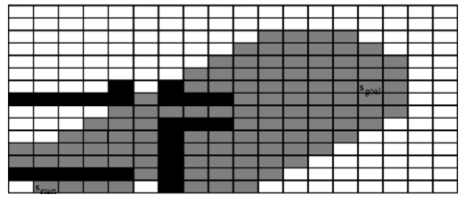
- until goal is reached:
  - ComputePathReuse()
  - follow the path until world is updated with new information
  - update the corresponding transition costs
  - set s_{start} to the current state of the agent
- Information-complete, information-optimal

Comparison

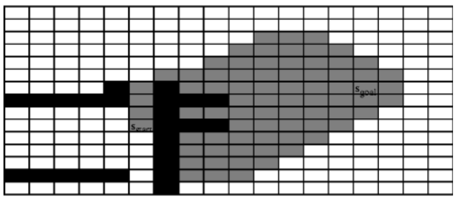
First A* search



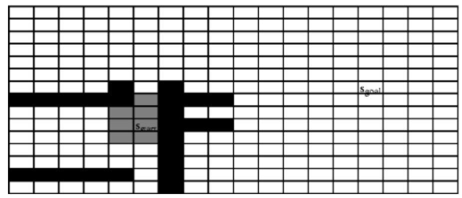
First D*-Lite search



Second A* search



Second D*-Lite search



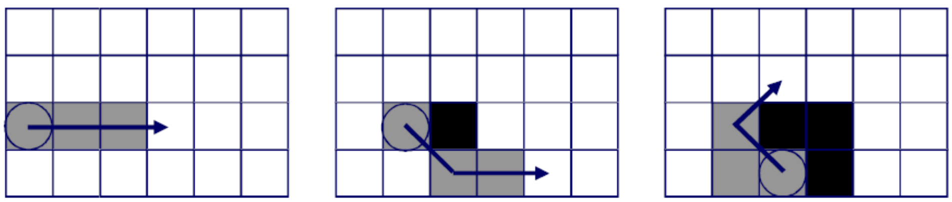
Anytime D*

- set ε to large value
- until goal is reached
  – ComputePathReuse() (weighted ε A*)
  – Follow the path until world is updated with new information
  – Update the corresponding edge costs
  – Set s_{start} to the current state of the agent
  – If "significant" changes were observed
    - increase ε or replan from scratch
  – else
    - decrease ε

No miracle: If too many changes we might as well recompute from scratch

Controlling computation: Agent-centered search

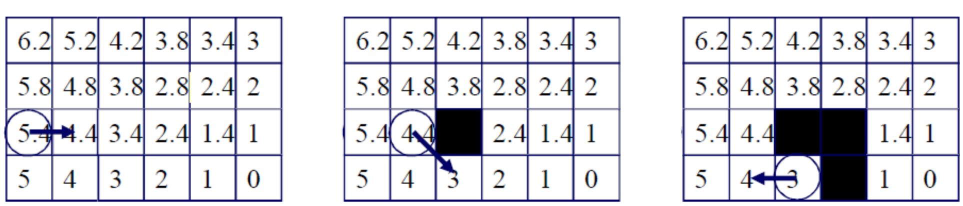
- Extreme case:
  - Constant (small) amount of computation
  - Don’t even try to plan to the goal
  - Just plan 1 step ahead



Controlling computation: Agent-centered search

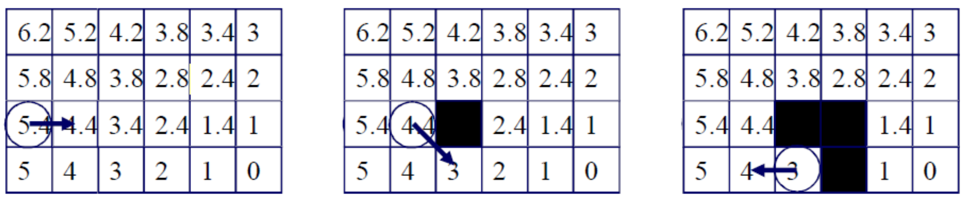
s_{start} = argmin $_{s \in \text{succ}(sstart)}$ $c(s_{start}, s) + h(s)$

Local minimum problem:



Controlling computation: Agent-centered search

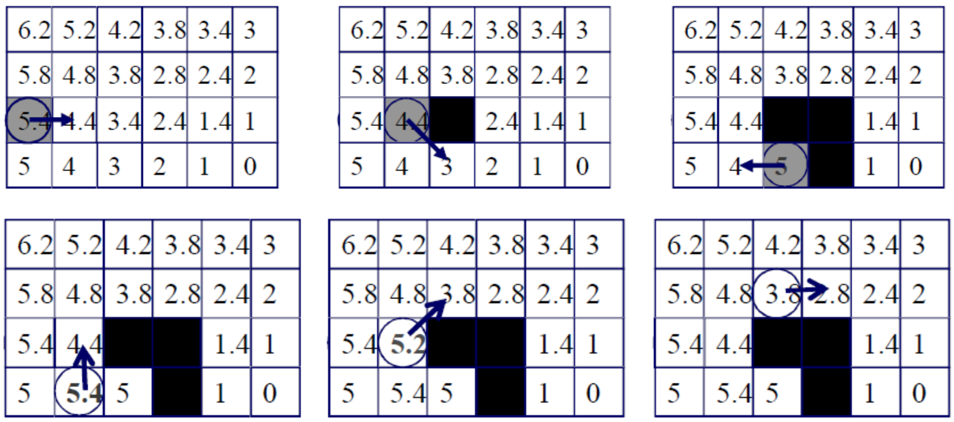
- Solution:
- Update $h(s_{start}) = \min_{s\, \epsilon succ(sstart)} c(s_{start}, s) + h(s)$



Learning Real-Time A* (LRTA*)

- s_{start} = current position

1. Update: $h(s_{start}) = \min_{s\, \epsilon succ(sstart)} c(s_{start}, s) + h(s)$
2. Move: $s_{start} = \text{argmin}_{s\, \epsilon succ(sstart)} c(s_{start}, s) + h(s)$



LRTA*

- robot is guaranteed to reach goal in finite number of steps if:
  - all costs are bounded from below with Δ > 0
  - graph is of finite size and there exists a finite-cost path to the goal
  - all actions are irreversible
- Extension: Expand *N* steps