16-350
Planning Techniques for Robotics

Interleaving Planning and Execution: Incremental Heuristic Search

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

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

Planning during Execution

- Planning is a repeated process!
  - partially-known environments → *edgecost changes*
  - dynamic environments → *edgecost changes, goal changes*
  - imperfect execution of plans → *robot pose changes/deviates off the path*
  - imprecise localization → *robot pose changes/deviates off the path*

- 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

Only Goal Changes

Any ideas how to handle it?

Only Goal Changes

Any ideas how to handle it?

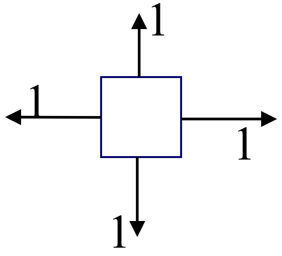
*Re-compute heuristics with respect to the **new** goal, and continue searching until the **new** goal state is expanded*

Only Goal Changes

- Example on the board!

$$h(cell\ \langle x,y\rangle) = |x-x_{goal}| + |y-y_{goal}|\ (Manhattan\ Distance)$$

4-connected grid



robot

	A	B	C	D	E
1	h=5	h=4	h=3	h=2	h=1
2	h=4	h=3	h=2	■	h=0
3	h=5	h=4	h=3	h=2	h=1
4	h=6	h=5	h=4	h=3	h=2

goal during 1st planning

robot

	A	B	C	D	E
1	h=6	h=5	h=4	h=3	h=2
2	h=5	h=4	h=3	■	h=1
3	h=4	h=3	h=2	h=1	h=0
4	h=5	h=4	h=3	h=2	h=1

goal during 2nd planning

Only Robot Pose Changes

Any ideas how to handle it?

Only Robot Pose Changes

Any ideas how to handle it?

Do the search backwards:
Then, the problem becomes "Only Goal Changes" that we know how to solve already

What if both Robot Pose and its Goal change?

Too bad!
Typically, you are better of re-planning from scratch then.

Changes to Edgecosts

- Two main reasons
  - Noisy perception (e.g., flickering obstacles, sensed position of obstacles is shifting, robot localization is noisy, etc.)
  - Partially-known environment

Changes to Edgecosts

- Two main reasons

Typically, it is important to do clever filtering to minimize flicker as much as possible without sacrificing safety

  - Noisy perception (e.g., flickering obstacles, sensed position of obstacles is shifting, robot localization is noisy, etc.)
  - Partially-known environment

Changes to Edgecosts

Typically, it is important to do clever filtering to minimize flicker as much as possible without sacrificing safety

- Two main reasons
  - Noisy perception (e.g., flickering obstacles, sensed position of obstacles is shifting, robot localization is noisy, etc.)
  - Partially-known environment

What should we assume about unknown space?

Changes to Edgecosts

- Two main reasons
  - Noisy perception (e.g., flickering obstacles, sensed position of obstacles is shifting, robot localization is noisy, etc.)
  - Partially-known environment

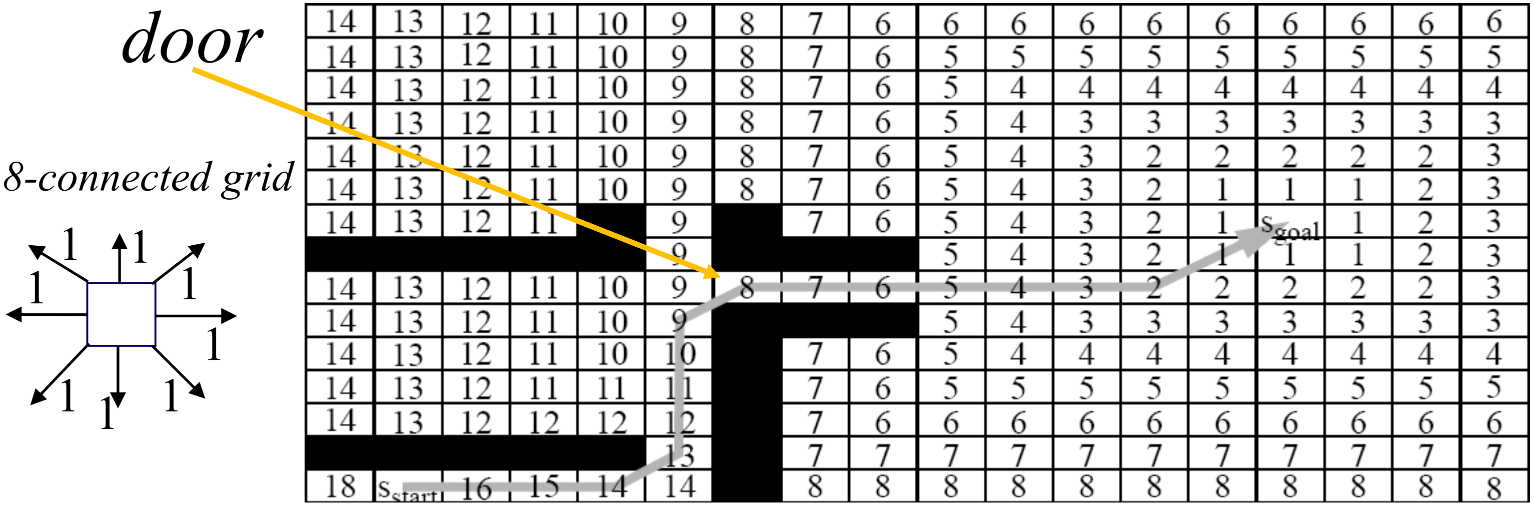
Typically, it is important to do clever filtering to minimize flicker as much as possible without sacrificing safety

What should we assume about unknown space?

***Freespace Assumption**: Assume that any "unknown" space is traversable until sensed otherwise!*

Changes to Edgecosts

- The robot doesn't initially know the status of the door



14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11	■	9	■	7	6	5	4	3	2	1	s_{goal}	1	2	3
■	■	■	■	■	9	■	■	■	5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	■	■	■	5	4	3	3	3	3	3	3	3
14	13	12	11	10	10	■	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11	■	7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12	■	7	6	6	6	6	6	6	6	6	6	6
■	■	■	■	■	13	■	7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14	■	8	8	8	8	8	8	8	8	8	8	8

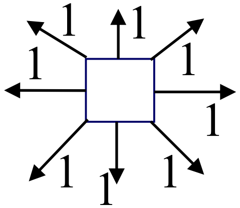
We ran an uninformed A search backwards (that is, all g-values are costs to s_{goal})*

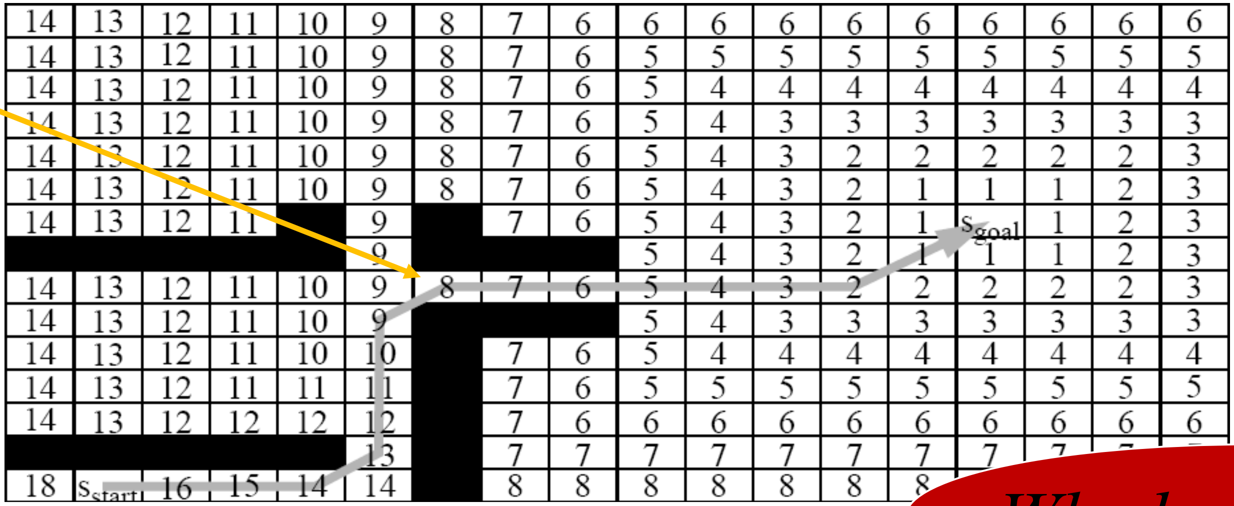
Changes to Edgecosts

- The robot doesn’t initially know the status of the door

door

8-connected grid





14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					9				5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9				5	4	3	3	3	3	3	3	3
14	13	12	11	10	10		7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11		7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12		7	6	6	6	6	6	6	6	6	6	6
					13		7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14		8	8	8	8	8	8	8				

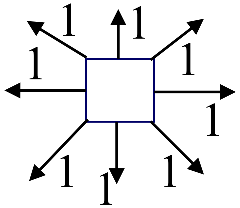
Why backwards?

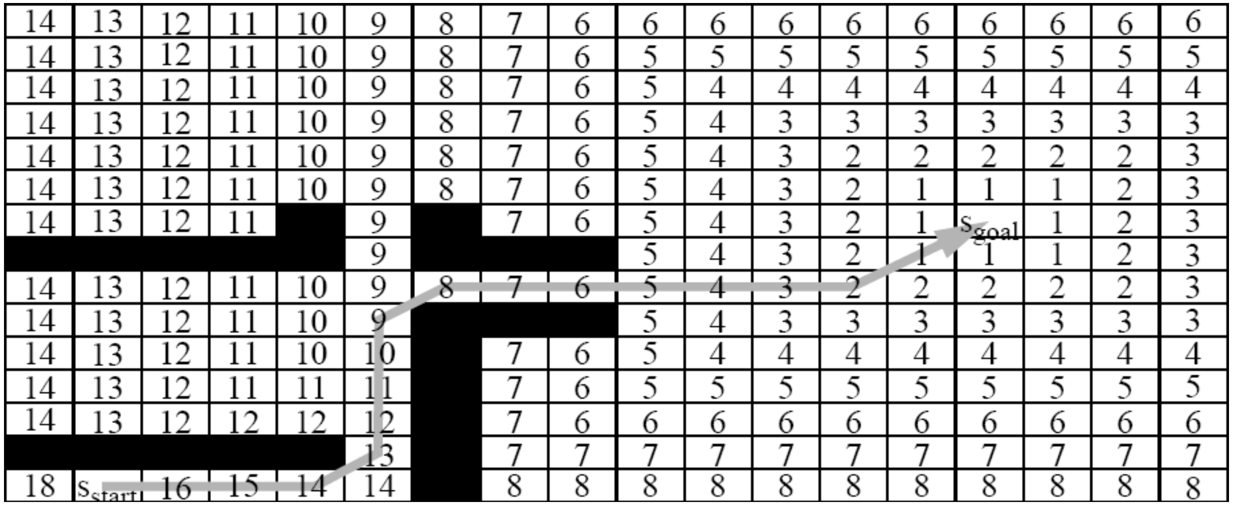
We ran an uninformed A search backwards (that is, all g-values are costs to s_{goal})*

Changes to Edgecosts

- The robot doesn’t initially know the status of the door

8-connected grid

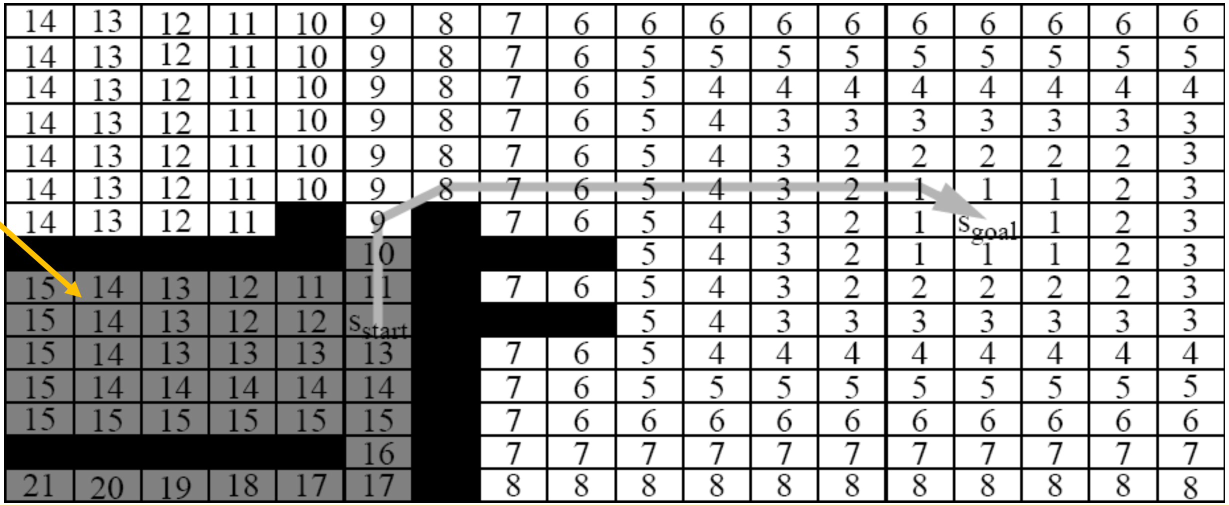




14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					9				5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9				5	4	3	3	3	3	3	3	3
14	13	12	11	10	10		7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11		7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12		7	6	6	6	6	6	6	6	6	6	6
					13		7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14		8	8	8	8	8	8	8	8	8	8	8

States with changed g-values

during execution, the robot found out that the door is closed



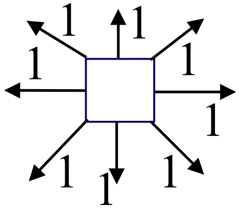
14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					10				5	4	3	2	1	1	1	2	3
15	14	13	12	11	11		7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	s_{start}				5	4	3	3	3	3	3	3	3
15	14	13	13	13	13		7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14		7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15		7	6	6	6	6	6	6	6	6	6	6
					16		7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17		8	8	8	8	8	8	8	8	8	8	8

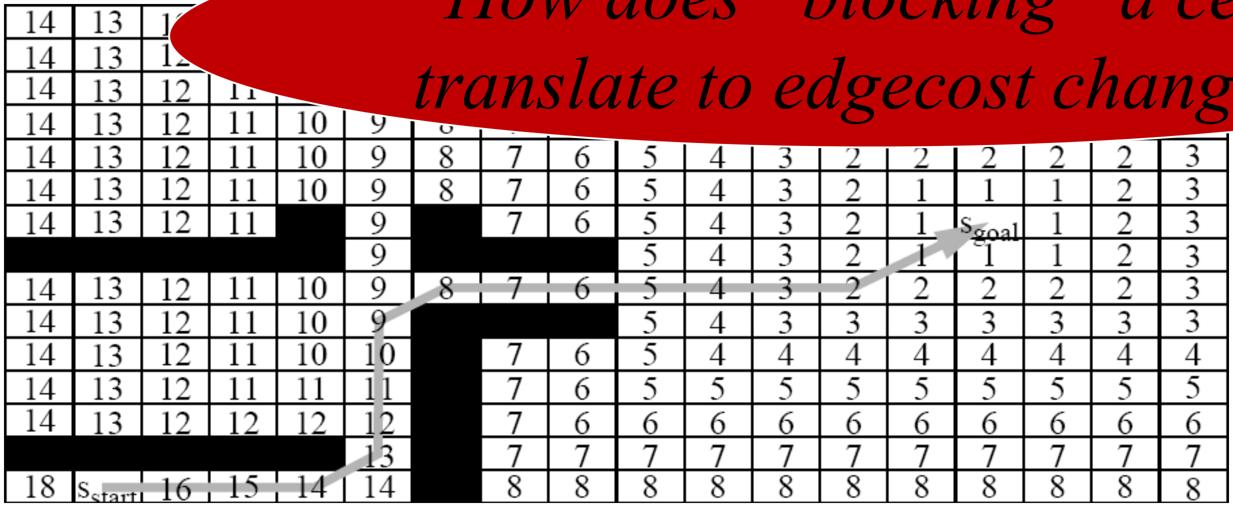
Changes to Edgecosts

- The robot doesn't initially know the status of the door

How does "blocking" a cell translate to edgecost changes?

8-connected grid

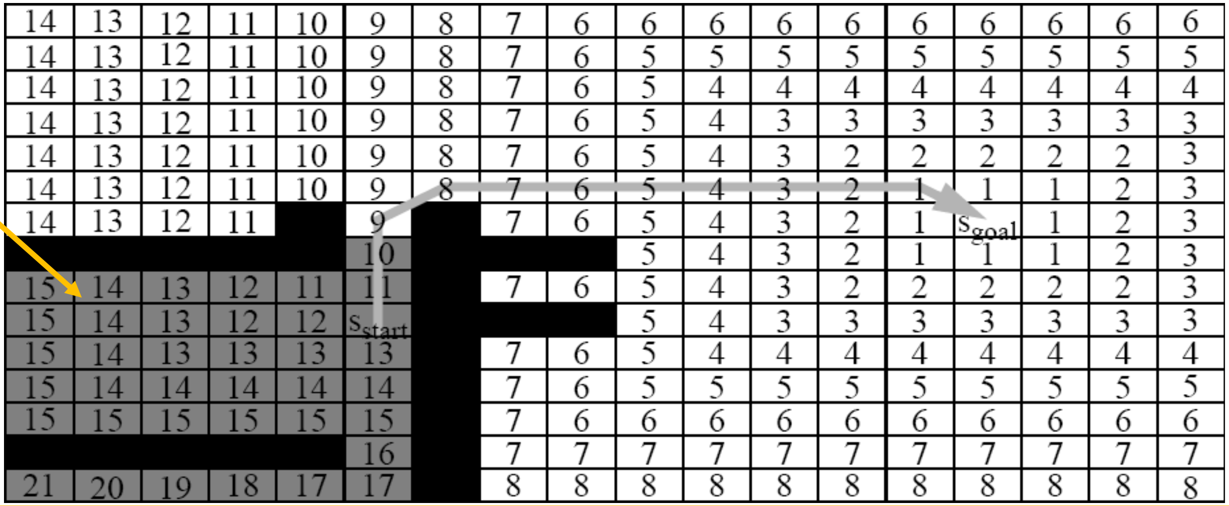




14	13																
14	13	12															
14	13	12	11														
14	13	12	11	10	9	8											
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11	■	9	■	7	6	5	4	3	2	1	s_{goal}	1	2	3
■	■	■	■	■	9	■	■	■	5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	■	■	■	5	4	3	3	3	3	3	3	3
14	13	12	11	10	10	■	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11	■	7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12	■	7	6	6	6	6	6	6	6	6	6	6
■	■	■	■	■	13	■	7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14	■	8	8	8	8	8	8	8	8	8	8	8

during execution, the robot found out that the door is closed

States with changed g-values

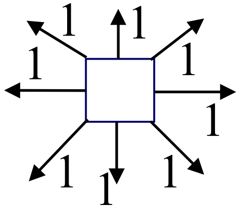


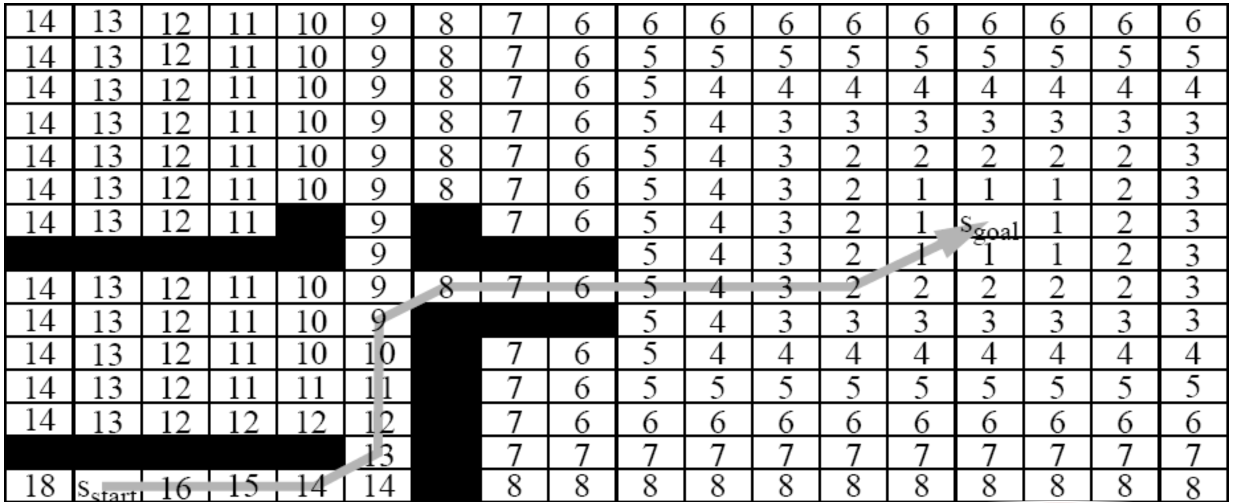
14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11	■	9	■	7	6	5	4	3	2	1	s_{goal}	1	2	3
■	■	■	■	■	10	■	■	■	5	4	3	2	1	1	1	2	3
15	14	13	12	11	11	■	7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	s_{start}	■	■	■	5	4	3	3	3	3	3	3	3
15	14	13	13	13	13	■	7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14	■	7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15	■	7	6	6	6	6	6	6	6	6	6	6
■	■	■	■	■	16	■	7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17	■	8	8	8	8	8	8	8	8	8	8	8

Changes to Edgecosts

- The robot doesn’t initially know the status of the door

8-connected grid



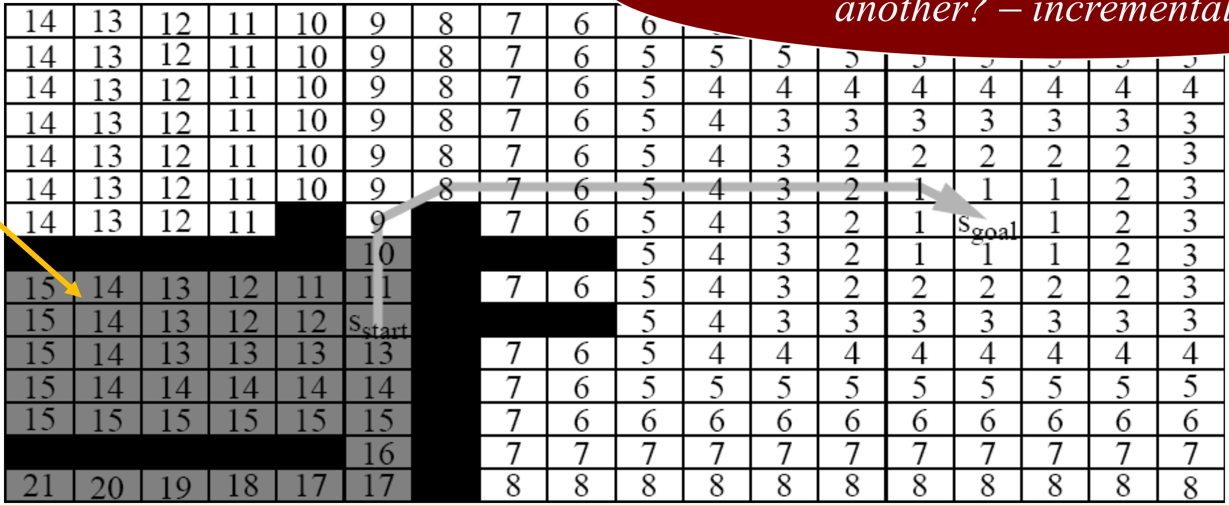


14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					9				5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9				5	4	3	3	3	3	3	3	3
14	13	12	11	10	10		7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11		7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12		7	6	6	6	6	6	6	6	6	6	6
					13		7	7	7	7	7	7	7	7	7	7	7
18	s_{start}	16	15	14	14		8	8	8	8	8	8	8	8	8	8	8

during execution, the robot fo

*Can we reuse these g-values from one search to another? – incremental A**

States with changed g-values

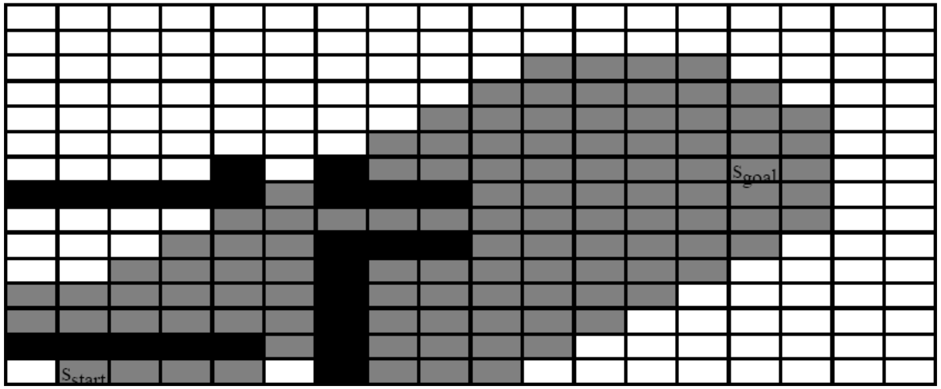


14	13	12	11	10	9	8	7	6	6								
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11		9		7	6	5	4	3	2	1	s_{goal}	1	2	3
					10				5	4	3	2	1	1	1	2	3
15	14	13	12	11	11		7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	s_{start}				5	4	3	3	3	3	3	3	3
15	14	13	13	13	13		7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14		7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15		7	6	6	6	6	6	6	6	6	6	6
					16		7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17		8	8	8	8	8	8	8	8	8	8	8

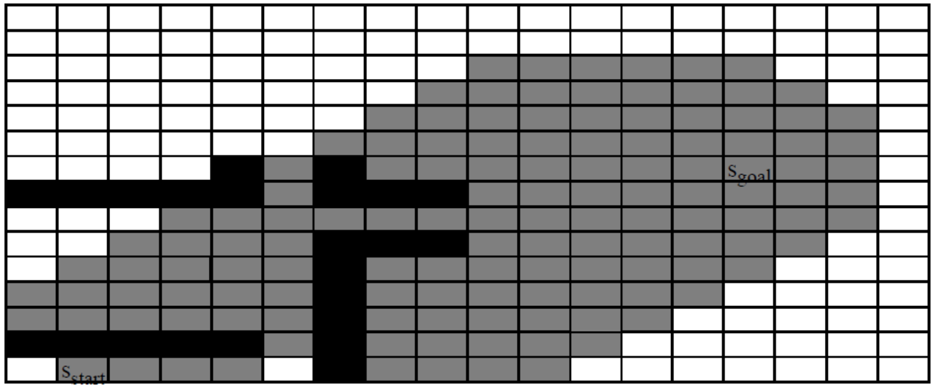
Incremental Heuristic Search

- D*/D* Lite: Incremental Heuristic Search Algorithms

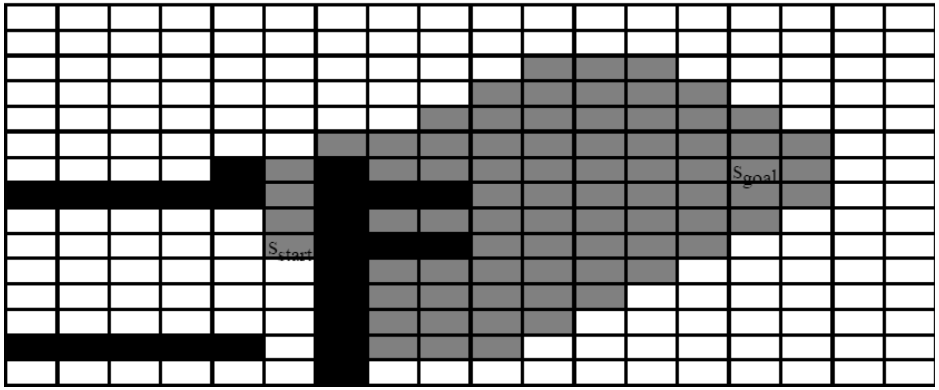
*initial search by backwards A**



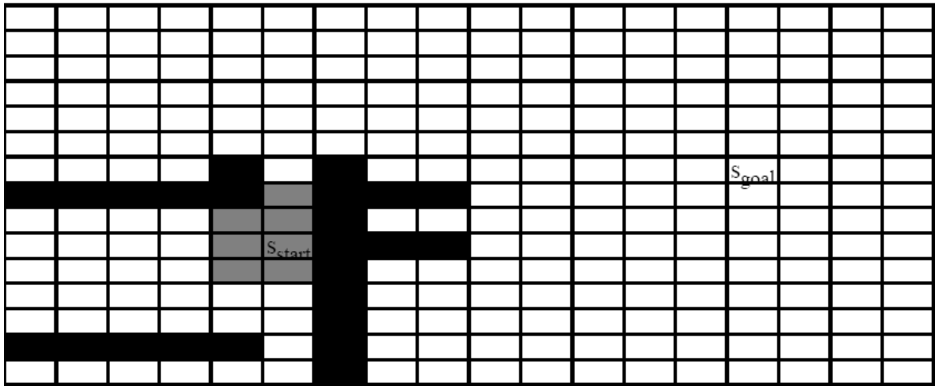
initial search by D Lite*



*second search by backwards A**

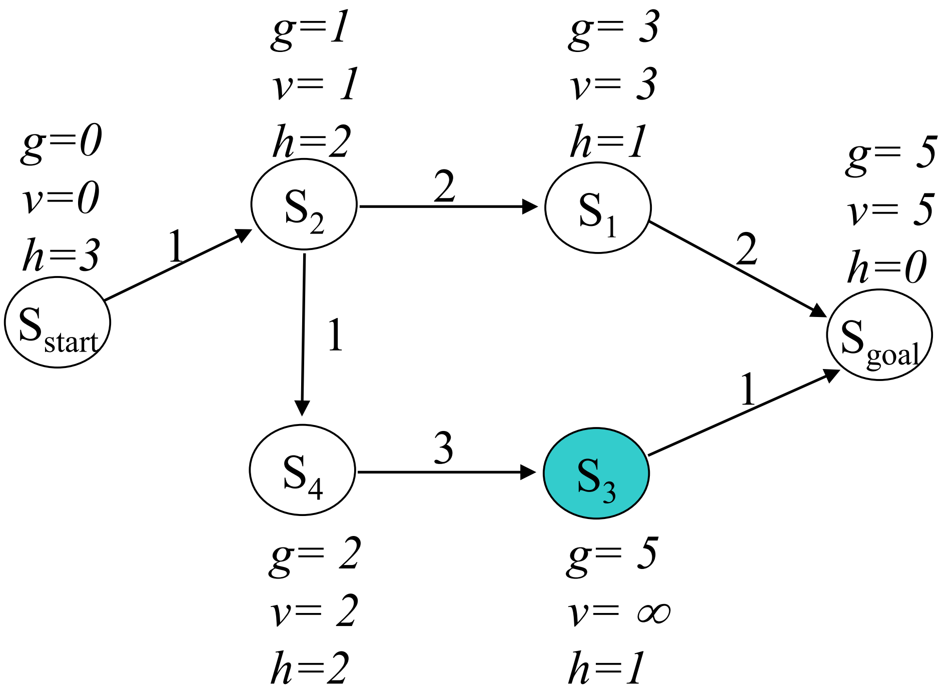


second search by D Lite*



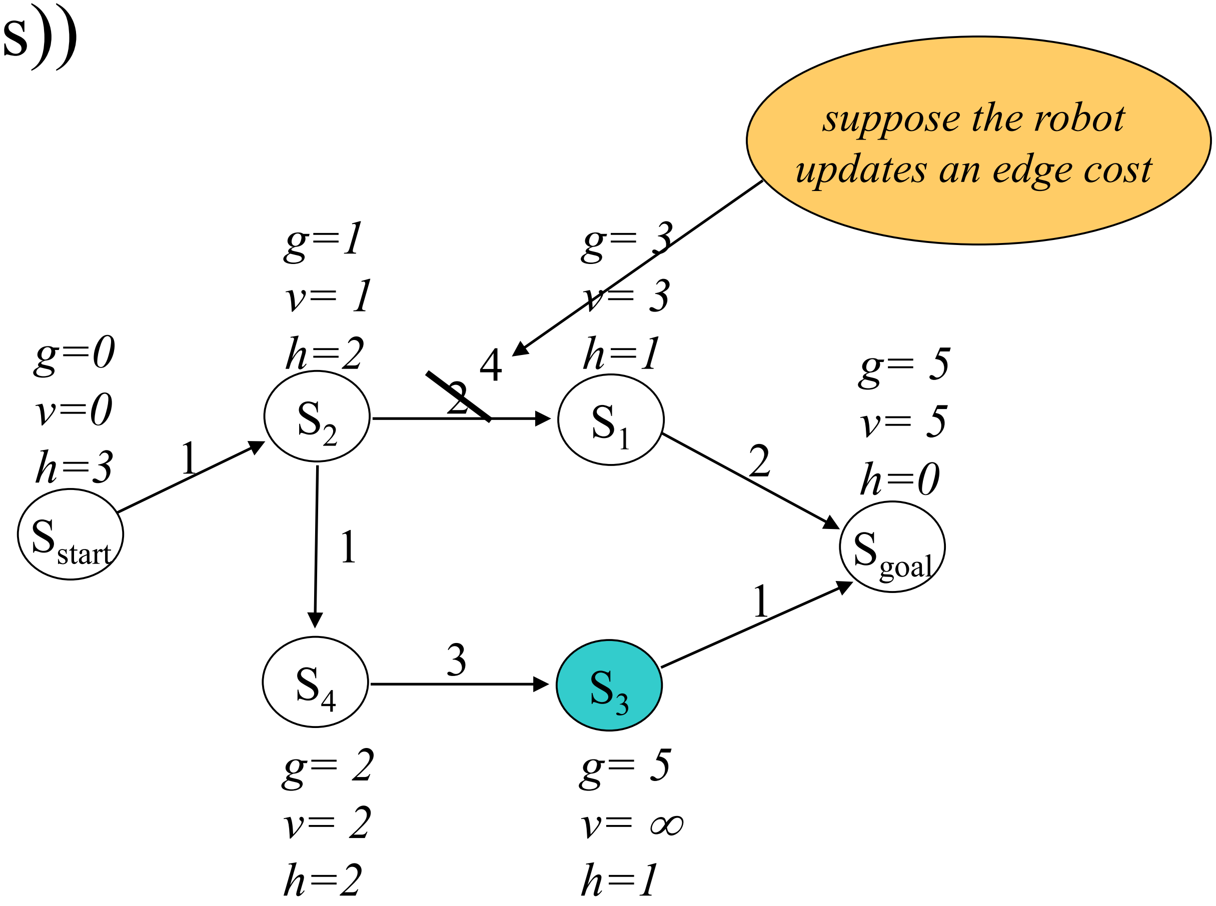
A* with Reuse of State Values

- So far, ComputePathwithReuse() could only deal with states whose *v(s)* ≥ *g(s)* (overconsistent or consistent)
- Edge cost increases may introduce underconsistent states (v(s) < g(s))



A* with Reuse of State Values

- So far, ComputePathwithReuse() could only deal with states whose *v(s)* ≥ *g(s)* (overconsistent or consistent)
- Edge cost increases may introduce underconsistent states (v(s) < g(s))

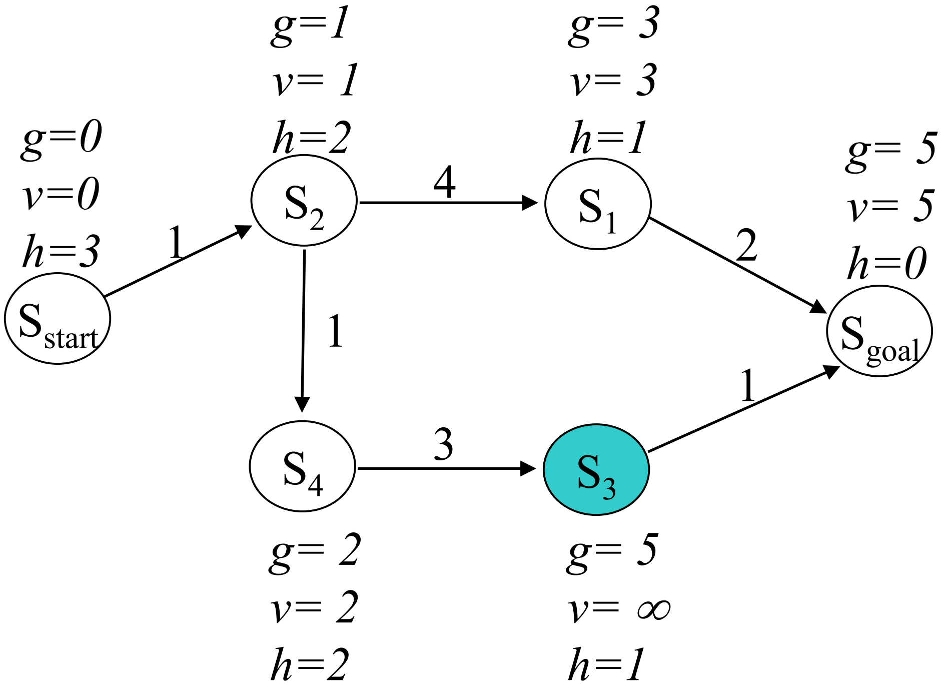


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states ($v(s) < g(s)$)

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

need to update $g(s_1)$



A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))

ComputePathwithReuse invariant:

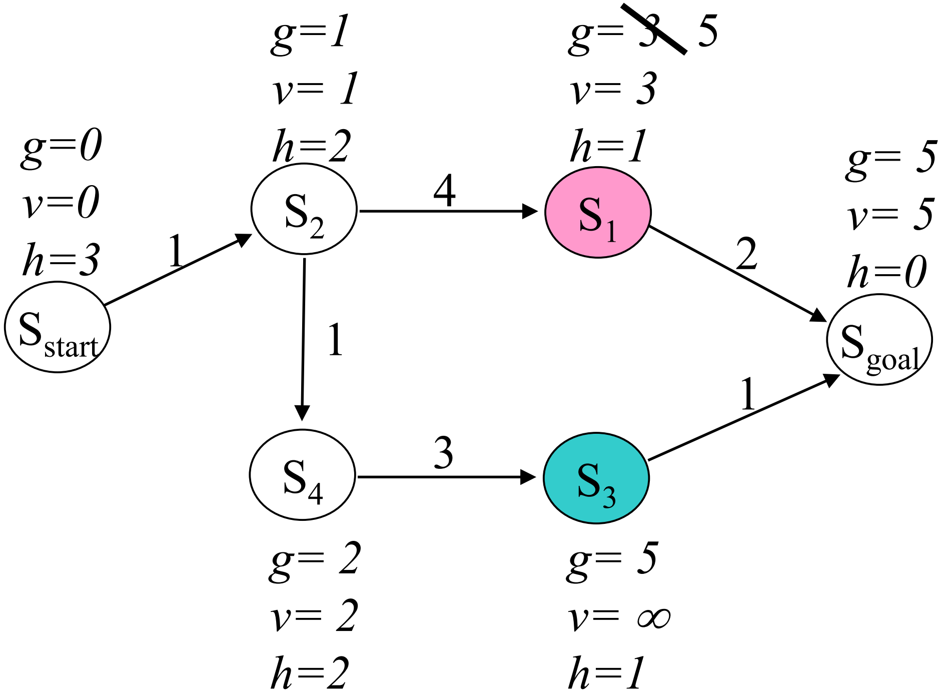
$g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$

↓

need to update $g(s_1)$

↓

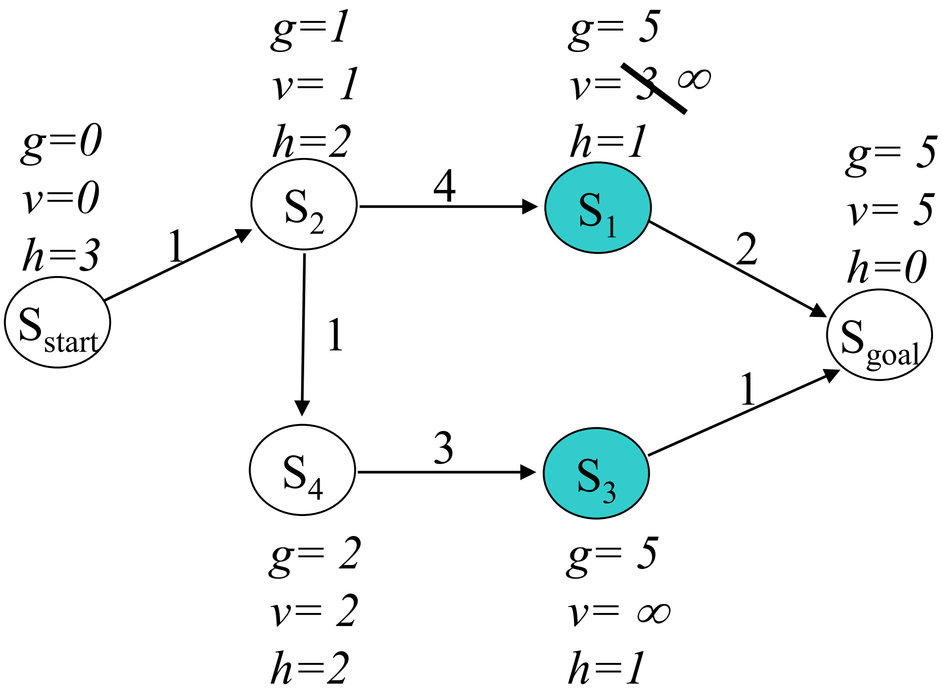
$v(s_1) < g(s_1)$



A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting $v(s) = \infty$

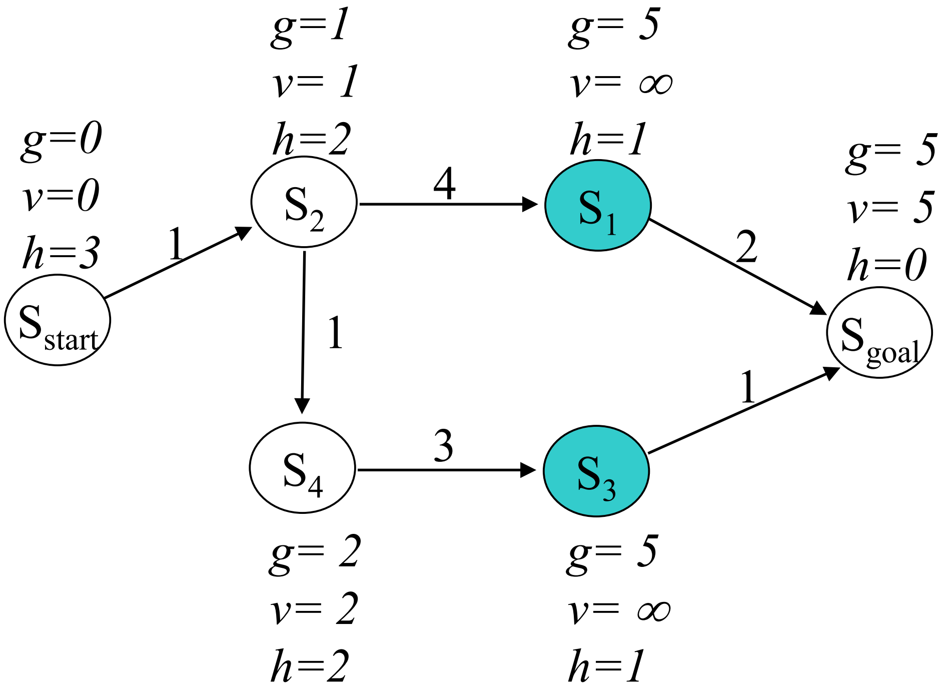
ComputePathwithReuse invariant:
$g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$



A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consistent *v(s)* ≥ *g(s)*

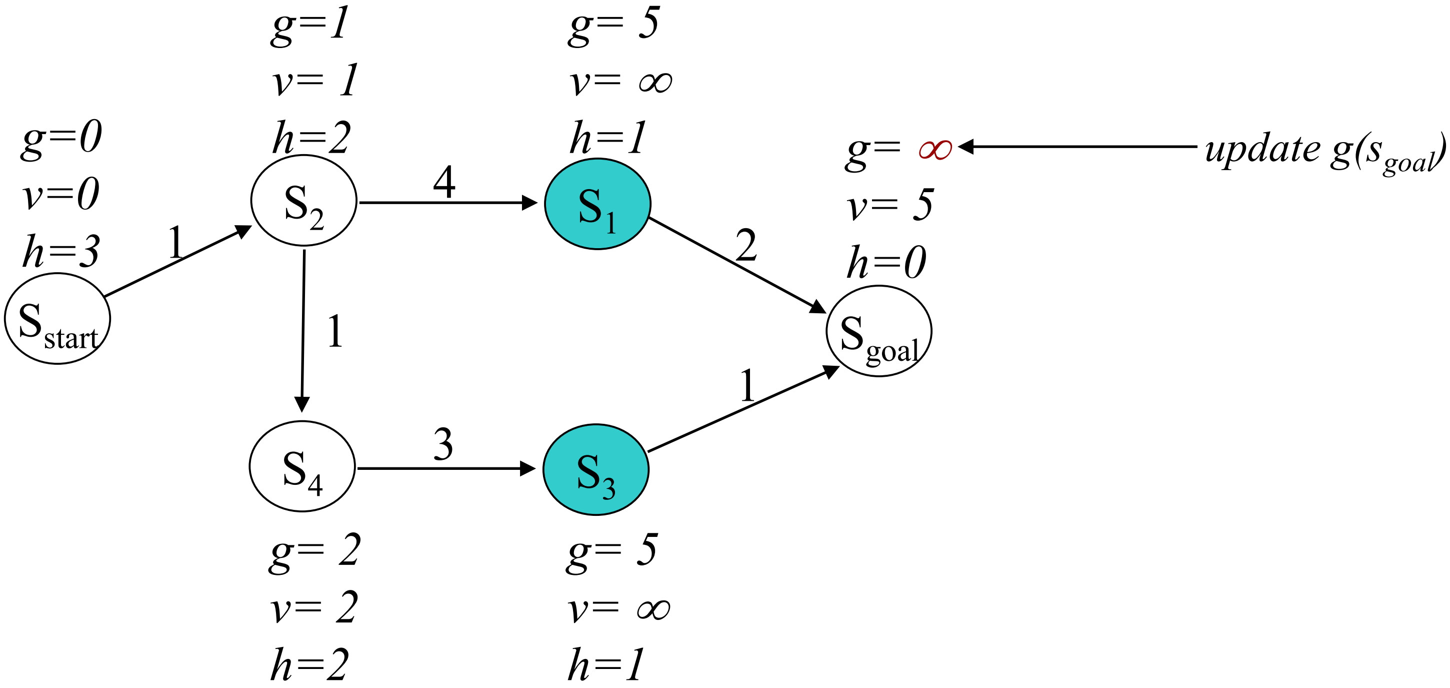
ComputePathwithReuse invariant:
$g(s') = \min_{s'' \in pred(s')} v(s'') + c(s'',s')$



A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consistent *v(s)* ≥ *g(s)*
- Propagate the changes

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

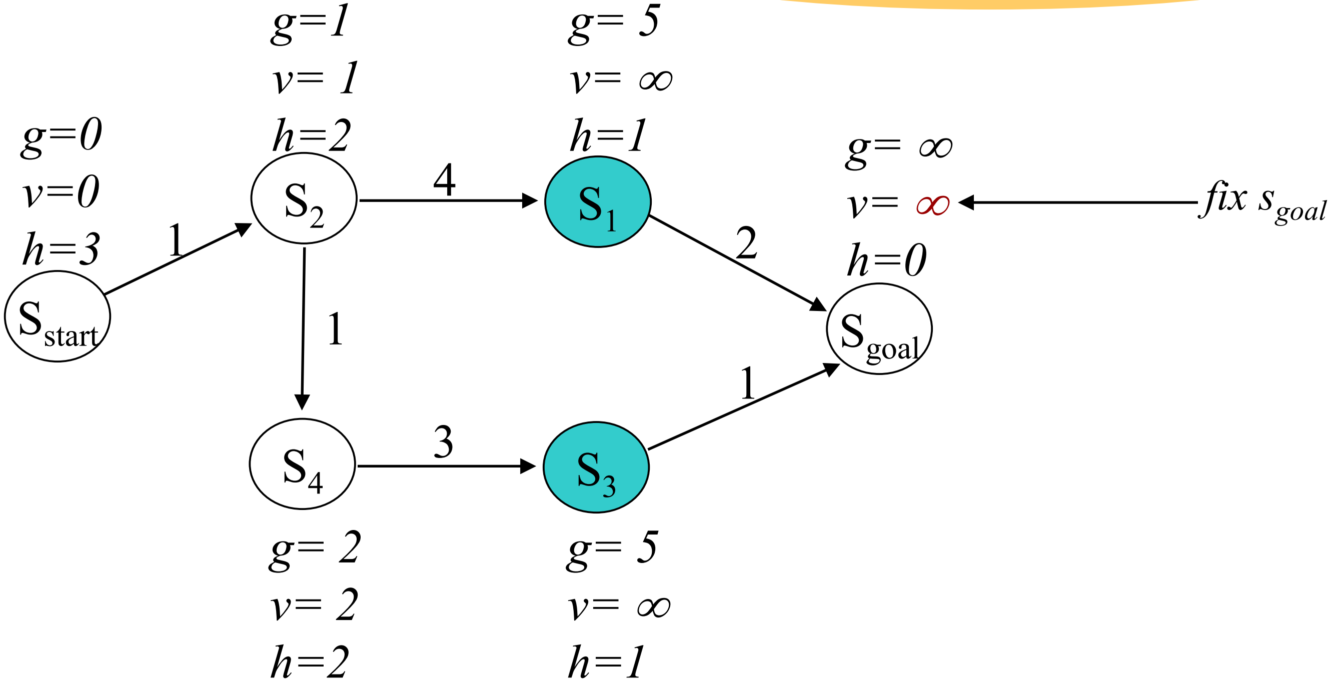


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting $v(s) = \infty$
- Makes *s* overconsistent or consistent $v(s) \geq g(s)$
- Propagate the changes

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

no more underconsistent states!

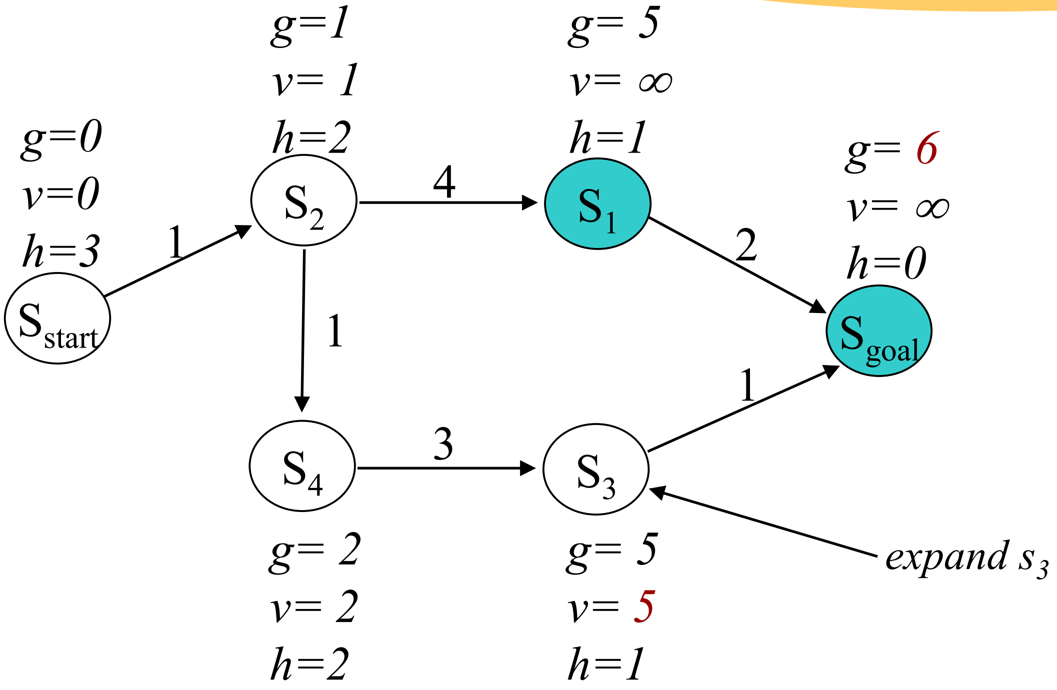


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consistent *v(s)* ≥ *g(s)*
- Propagate the changes

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

no more underconsistent states!

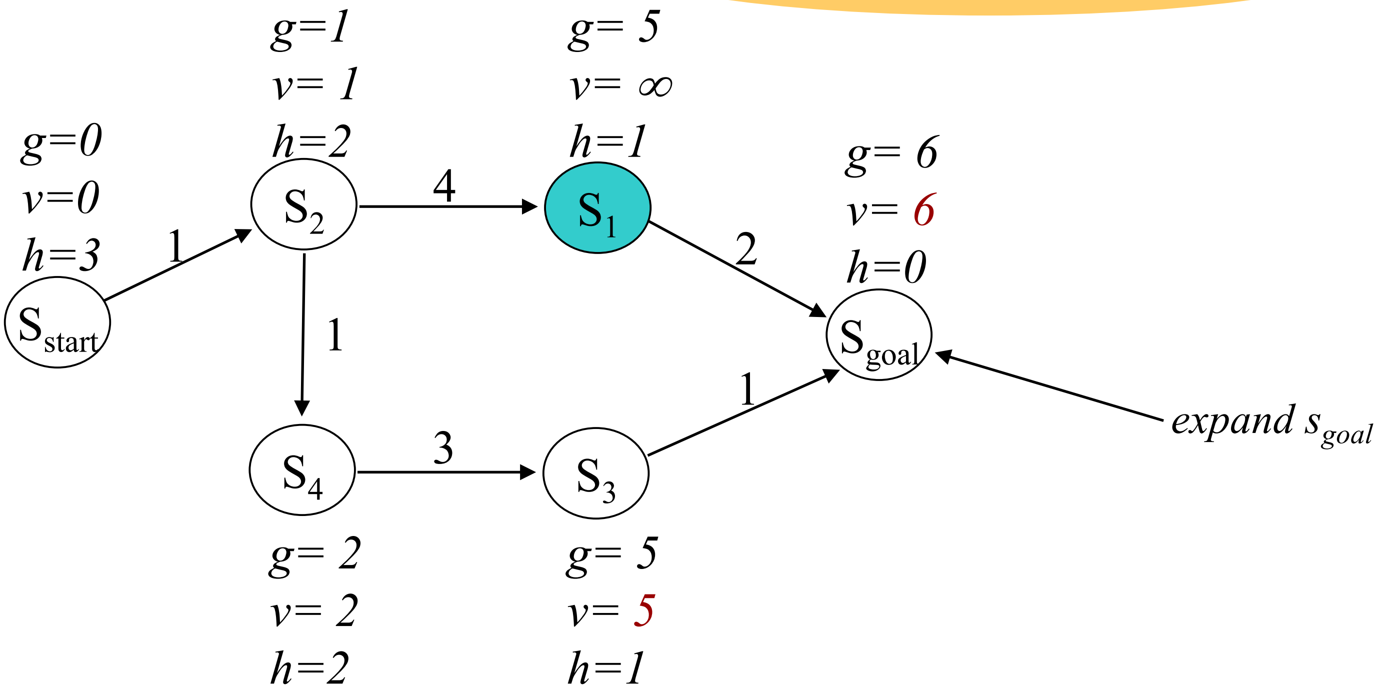


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consistent *v(s)* ≥ *g(s)*
- Propagate the changes

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

no more underconsistent states!

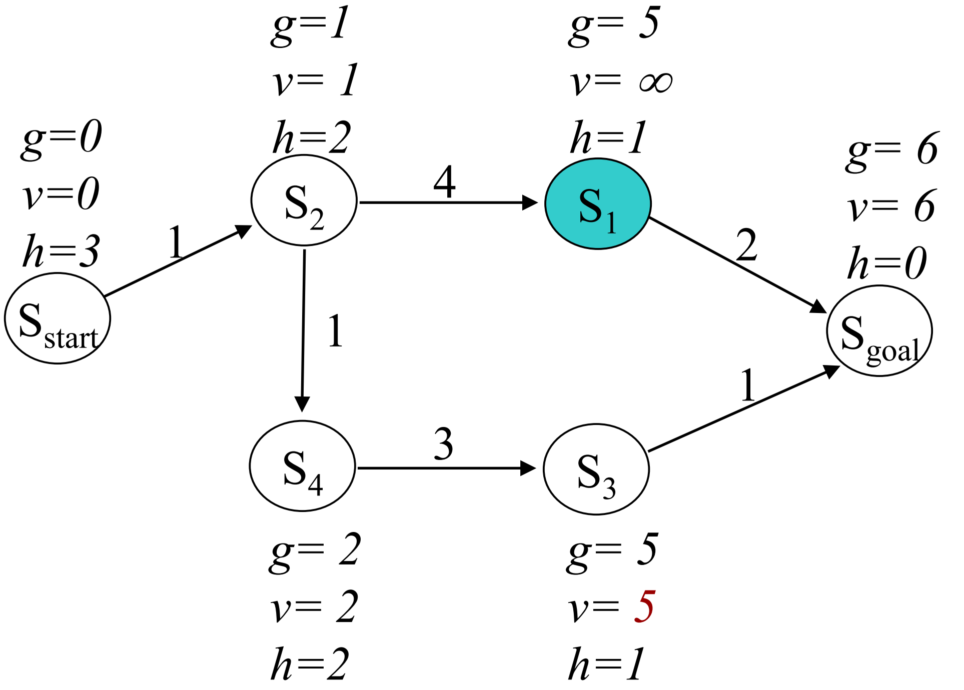


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consisten
- Propagate the changes

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

we can backtrack an optimal path
(start at s_{goal}, proceed to pred that minimizes g+c)

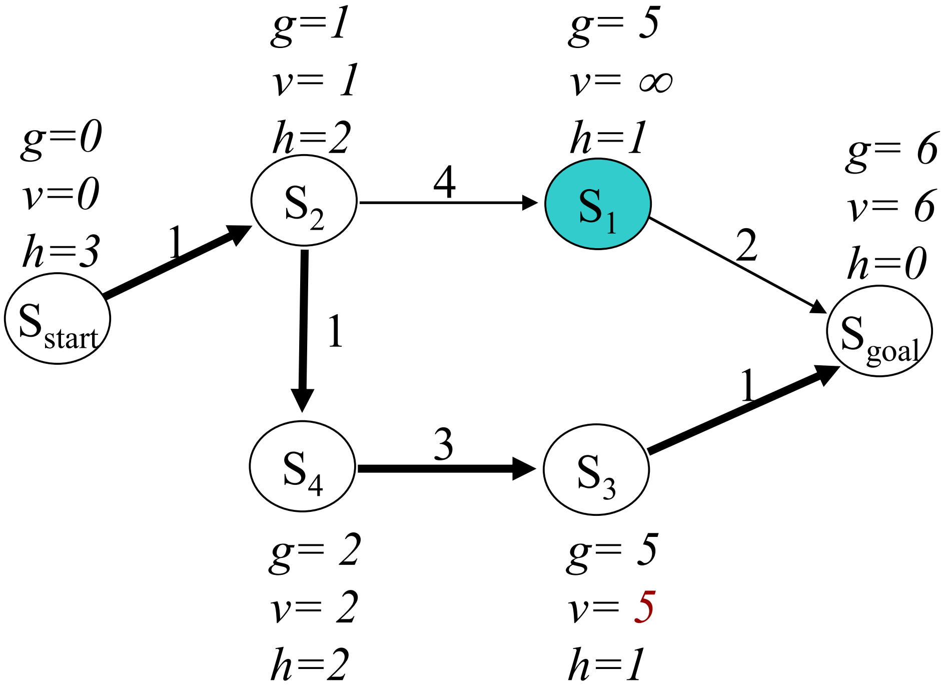


A* with Reuse of State Values

- Edge cost increases may introduce underconsistent states (v(s) < g(s))
- Fix these by setting *v(s)* = ∞
- Makes *s* overconsistent or consistent
- Propagate the changes

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

we can backtrack an optimal path
(start at s_{goal}, proceed to pred that minimizes g+c)



D* Lite

- Optimal re-planning algorithm
- Simpler and with nicer theoretical properties version of D*

```
until goal is reached
    ComputePathwithReuse();    //modified to fix underconsistent states
    publish optimal path;
    follow the path until map is updated with new sensor information;
    update the corresponding edge costs;
    set $s_{start}$ to the current state of the agent;
```

Anytime Incremental Heuristic Search

- Anytime D*:
  - decrease ε and update edge costs at the same time
  - re-compute a path by reusing previous state-values

```
set $\varepsilon$ to large value;
until goal is reached
    ComputePathwithReuse();    //modified to fix underconsistent states
    publish $\varepsilon$-suboptimal path;
    follow the path until map is updated with new sensor information;
    update the corresponding edge costs;
    set $s_{start}$ to the current state of the agent;
    if significant changes were observed
        increase $\varepsilon$ or replan from scratch;
    else
        decrease $\varepsilon$;
```

What for?

What You Should Know…

- How to handle changes to Robot Pose Only or Goal Only
- What is Freespace Assumption
- What is D*/D* Lite and the general principles behind it (don’t need to know the exact algorithm)