16-350
Planning Techniques for Robotics

Interleaving Planning and Execution: Real-time Heuristic Search

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

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

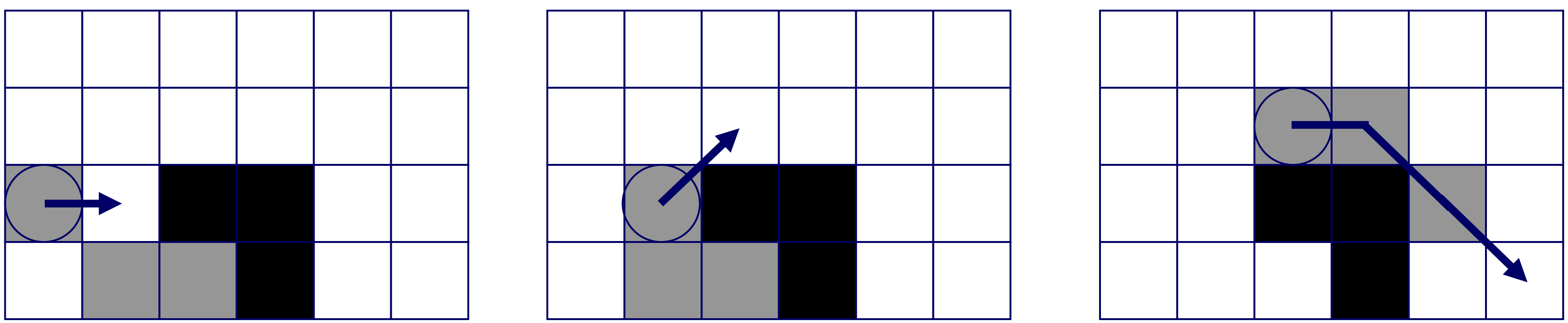
Real-time (Agent-centered) Heuristic Search

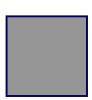
Enforce a strict limit on the amount of computations (no requirement on planning all the way to the goal)

Real-time (Agent-centered) Heuristic Search

1. Compute a partial path by expanding at most N states around the robot
2. Move once, incorporate sensor information, and goto step 1

Example in a fully-known terrain:

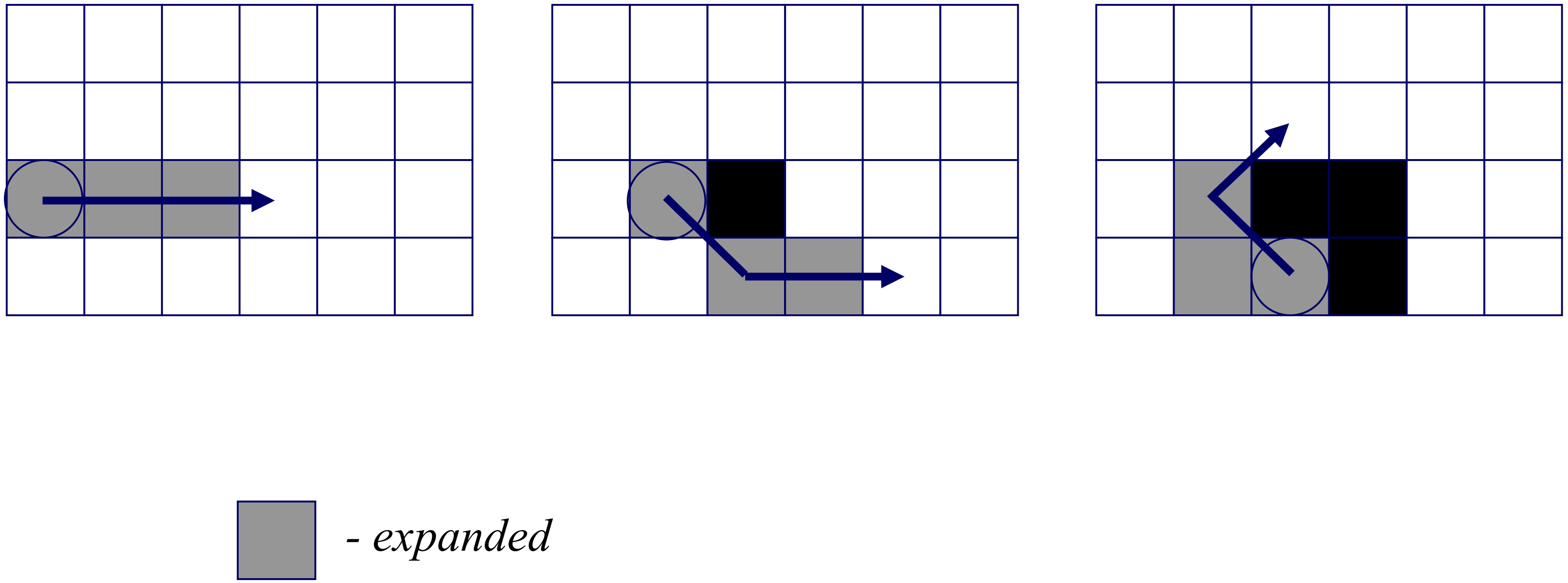


 - *expanded*

Real-time (Agent-centered) Heuristic Search

1. Compute a partial path by expanding at most N states around the robot
2. Move once, incorporate sensor information, and goto step 1

Example in an unknown terrain (planning with Freespace Assumption):



Real-time (Agent-centered) Heuristic Search

1. Compute a partial path by expanding at most N states around the robot
2. Move once, incorporate sensor information, and goto step 1

Research issues:

- how to compute partial path

- how to guarantee complete behavior (guarantee to reach the goal)

- provide bounds on the number of steps before reaching the goal

Suppose planner only has time to examine successors

What should the planner decide for the robot's next move?

$h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal})) + 0.4*min(abs(x-x_{goal}), abs(y-y_{goal}))$

6.2	5.2	4.2	3.8	3.4	3
5.8	4.8	3.8	2.8	2.4	2
5.4	4.4			1.4	1
5	4	3		1	0

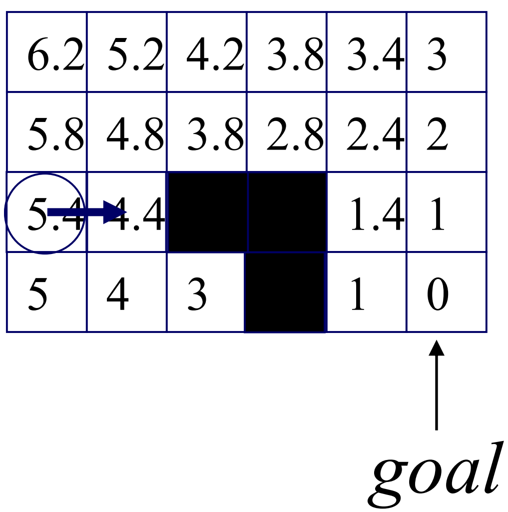
goal

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using heuristics

   1. *always move as follows:* $s_{start} = \arg\min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$

$h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal})) + 0.4*min(abs(x-x_{goal}), abs(y-y_{goal}))$



Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using heuristics

  1. *always move as follows:* $s_{start} = \text{argmin}_{s \in succ(sstart)} c(s_{start}, s) + h(s)$

$h(x,y) = \max(abs(x-x_{goal}), abs(y-y_{goal})) + 0.4 * \min(abs(x-x_{goal}), abs(y-y_{goal}))$

6.2	5.2	4.2	3.8	3.4	3
5.8	4.8	3.8	2.8	2.4	2
5.4	4.4			1.4	1
5	4	3		1	0

goal

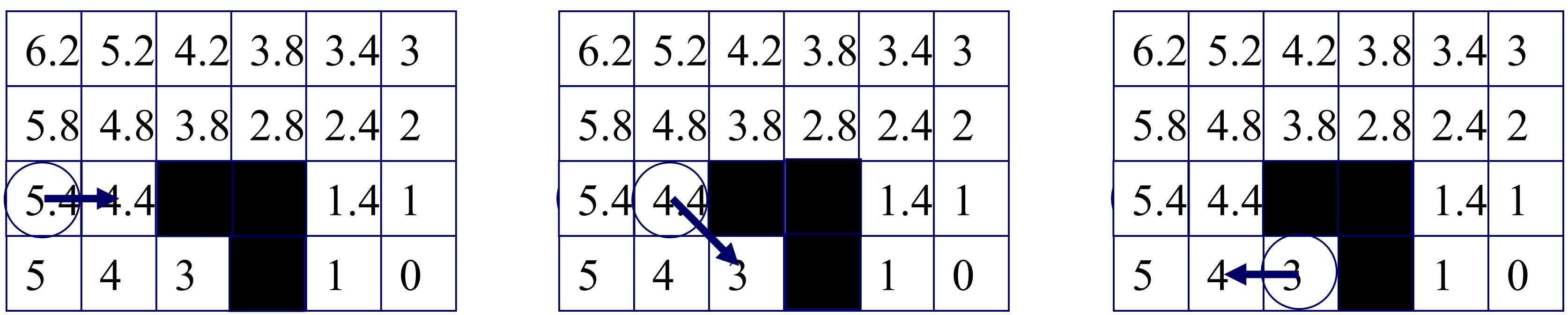
Any problems?

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using heuristics

    1. *always move as follows:* $s_{start} = \arg\min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$

$h(x,y) = \max(abs(x-x_{goal}), abs(y-y_{goal})) + 0.4*\min(abs(x-x_{goal}), abs(y-y_{goal}))$



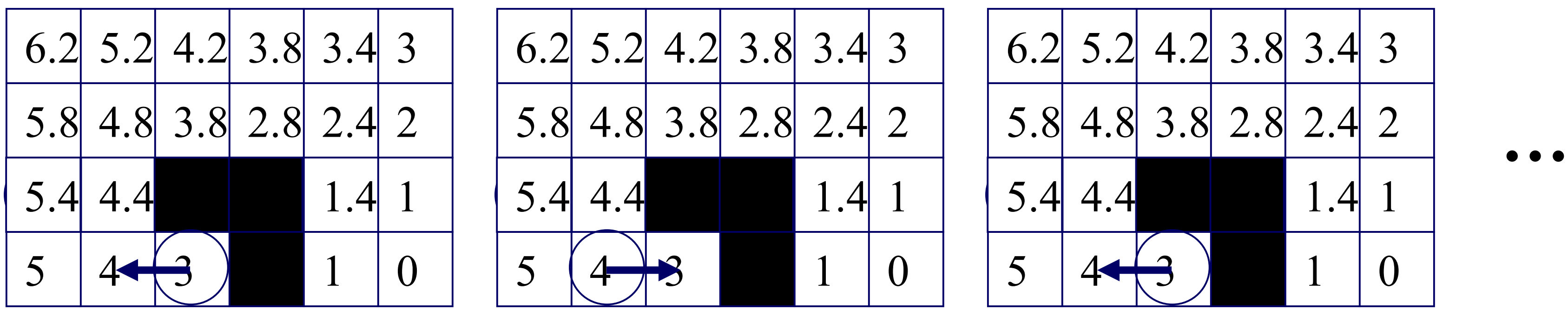
Any problems?

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using heuristics

    1. *always move as follows:* $s_{start} = argmin_{\, s \in succ(sstart)} c(s_{start}, s) + h(s)$

$h(x,y) = max(abs(x-x_{goal}), abs(y-y_{goal})) + 0.4*min(abs(x-x_{goal}), abs(y-y_{goal}))$



Local minima problem (myopic or incomplete behavior)

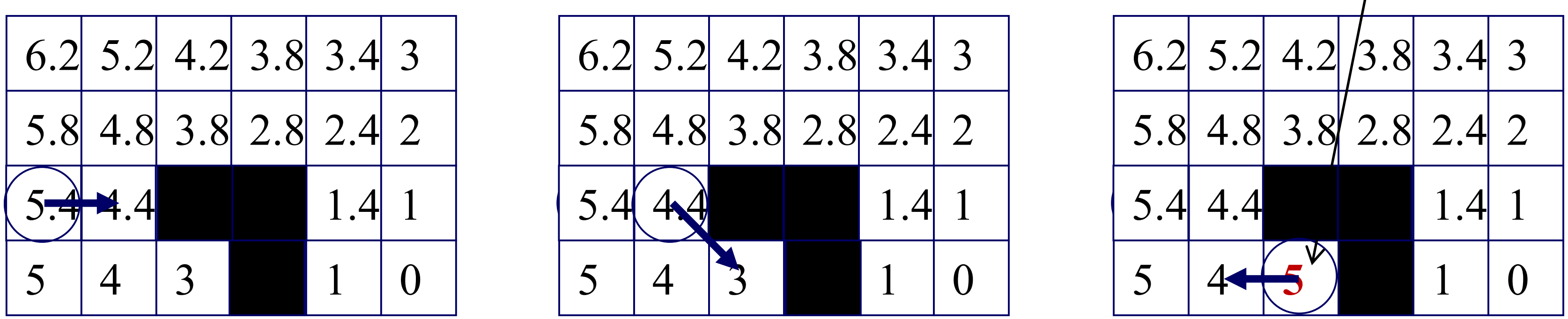
Any solutions?

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

makes h-values more informed

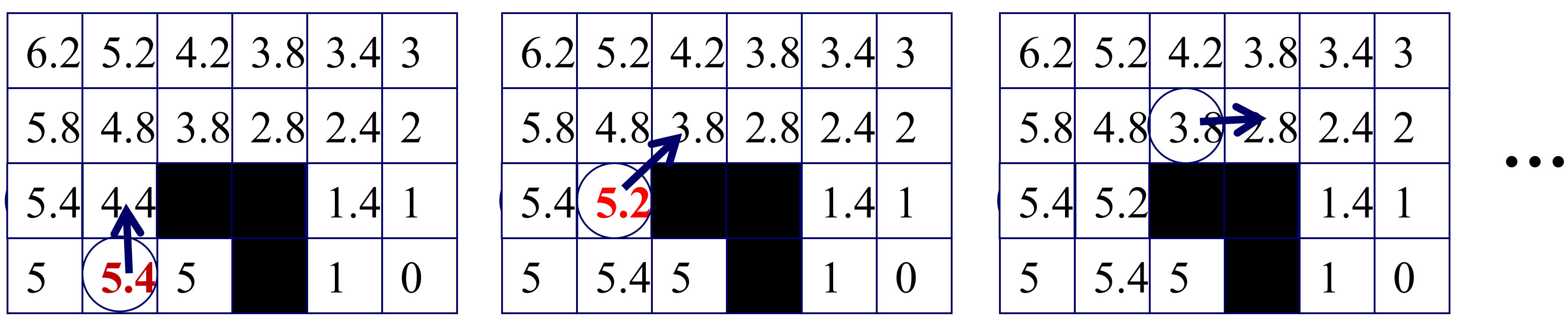
1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
2. *always move as follows:* $s_{start} = \arg\min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$



Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

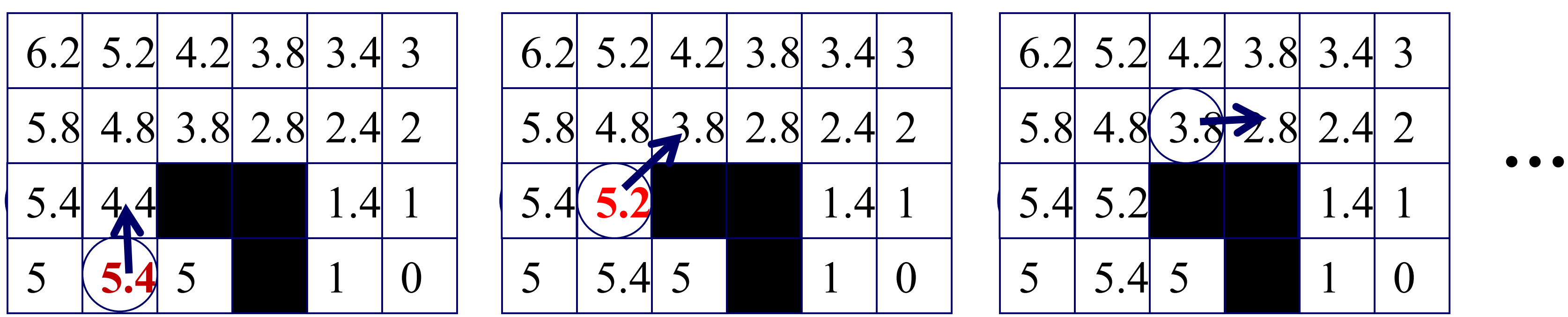
   1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
   2. *always move as follows:* $s_{start} = \operatorname{argmin}_{s \in succ(sstart)} c(s_{start}, s) + h(s)$



Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

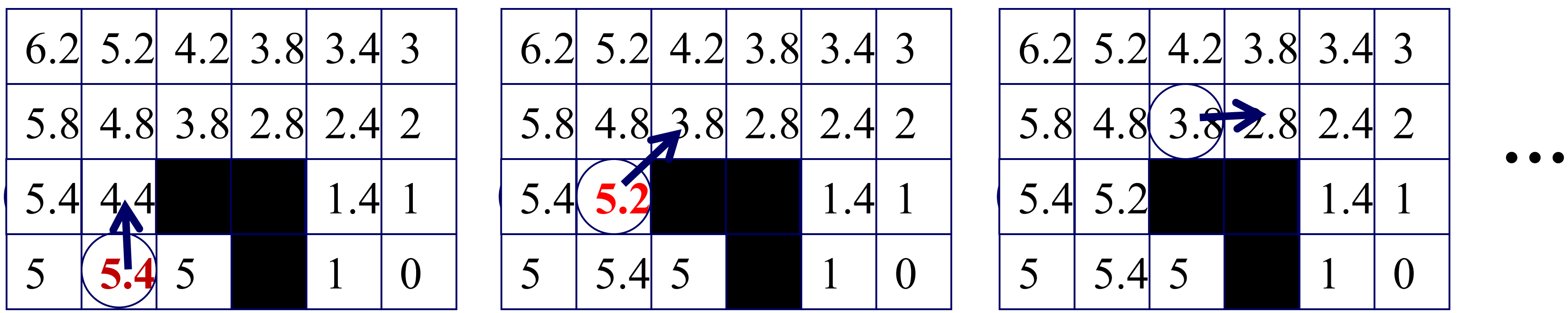
1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
2. *always move as follows:* $s_{start} = \operatorname{argmin}_{s \in succ(sstart)} c(s_{start}, s) + h(s)$



h-values guaranteed to remain admissible and consistent

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

  1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
  2. *always move as follows:* $s_{start} = \arg\min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$



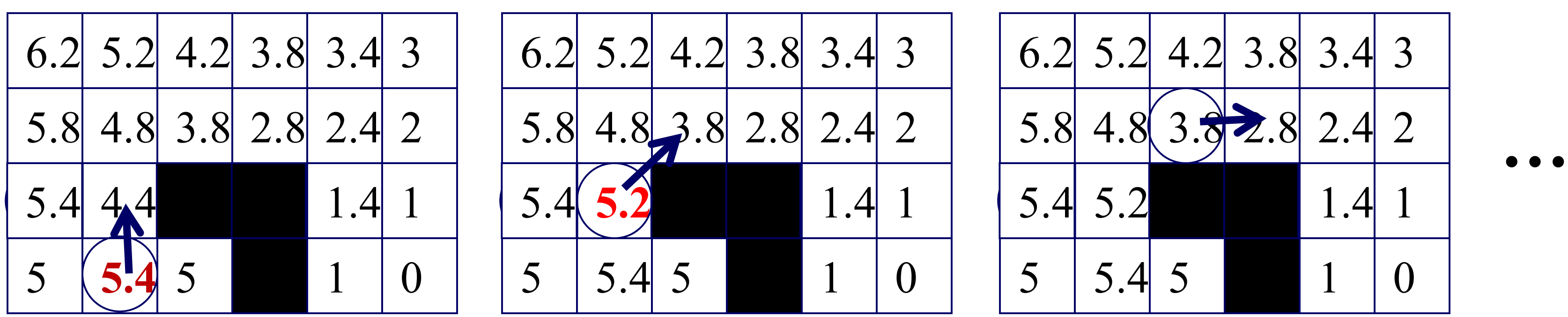
robot is guaranteed to reach goal in finite number of steps if:

- all costs are bounded from below with $\Delta > 0$
- graph is of finite size and there exists a finite-cost path to the goal
- all actions are reversible

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

    1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
    2. *always move as follows:* $s_{start} = \text{argmin}_{s \in succ(sstart)} c(s_{start}, s) + h(s)$



robot is guaranteed to reach goal in finite number of steps if:

- all costs are bounded from below with $\Delta > 0$
- graph is of finite size and there exists a finite-cost path to the goal
- all actions are reversible

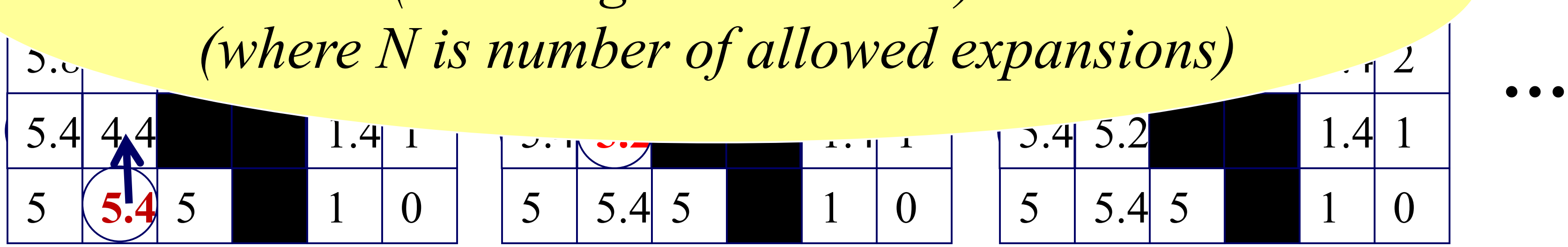
Why conditions?

Suppose planner only has time to examine successors

- Repeatedly move the robot to the most promising adjacent state, using **and updating** heuristics

  1. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
  2. *always move as follows:* $s_{start} = \text{argmin}_{s \in succ(sstart)} c(s_{start}, s) + h(s)$

This algorithm is called
LRTA (Learning Real-time A*) with N=1*
(where N is number of allowed expansions)



robot is guaranteed to reach goal in finite number of steps if:

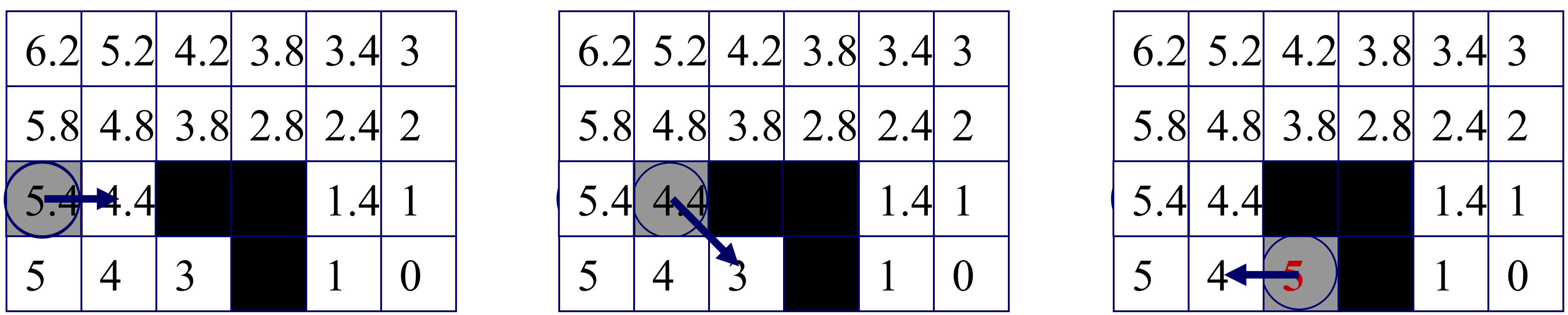
- all costs are bounded from below with $\Delta > 0$
- graph is of finite size and there exists a finite-cost path to the goal
- all actions are reversible

Learning Real-Time A* (LRTA*) with N=1

- expand $N = 1$ state, make a move towards a state s in *OPEN* with smallest $g(s)+h(s)$:

  1. *expand* s_{start}
  2. *update* $h(s_{start}) = \min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$
  3. *always move as follows:* $s_{start} = \arg\min_{s \in succ(sstart)} c(s_{start}, s) + h(s)$

     $= \arg\min_{s \in succ(sstart)} g(s) + h(s)$



- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

necessary for the guarantee to reach the goal

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state* $s = argmin_{s' \in OPEN}\, g(s') + h(s')$

\- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$

state *s*:

- the state that minimizes cost to it plus heuristic estimate of the remaining distance

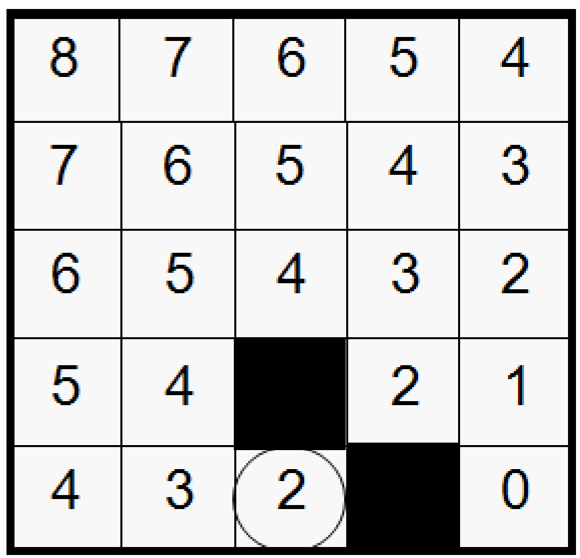
- the state that looks most promising in terms of the whole path from current robot state to goal

- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



4-connected grid (robot moves in 4 directions)

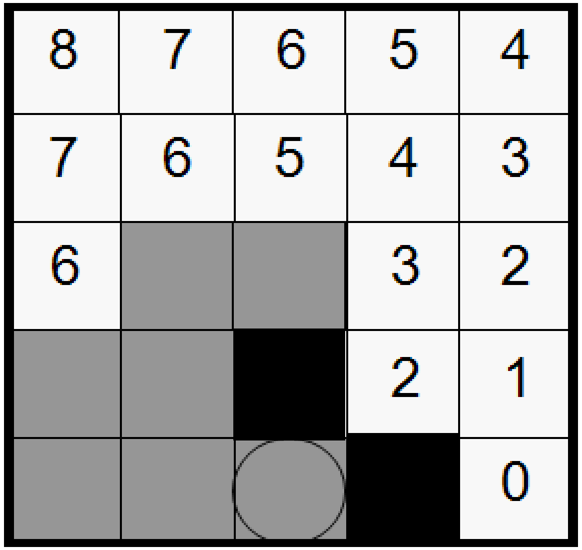
example borrowed from ICAPS'06 planning summer school lecture (Koenig & Likhachev)

- expanded

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$*



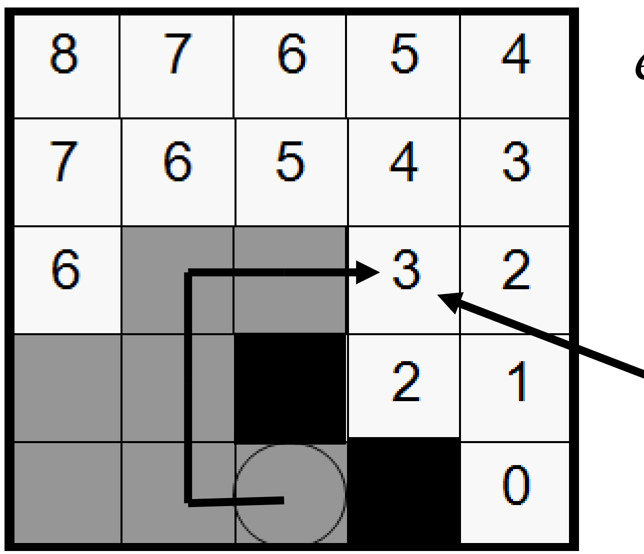
expand N=7 states

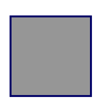
- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \arg\min_{s' \in OPEN} g(s') + h(s')$



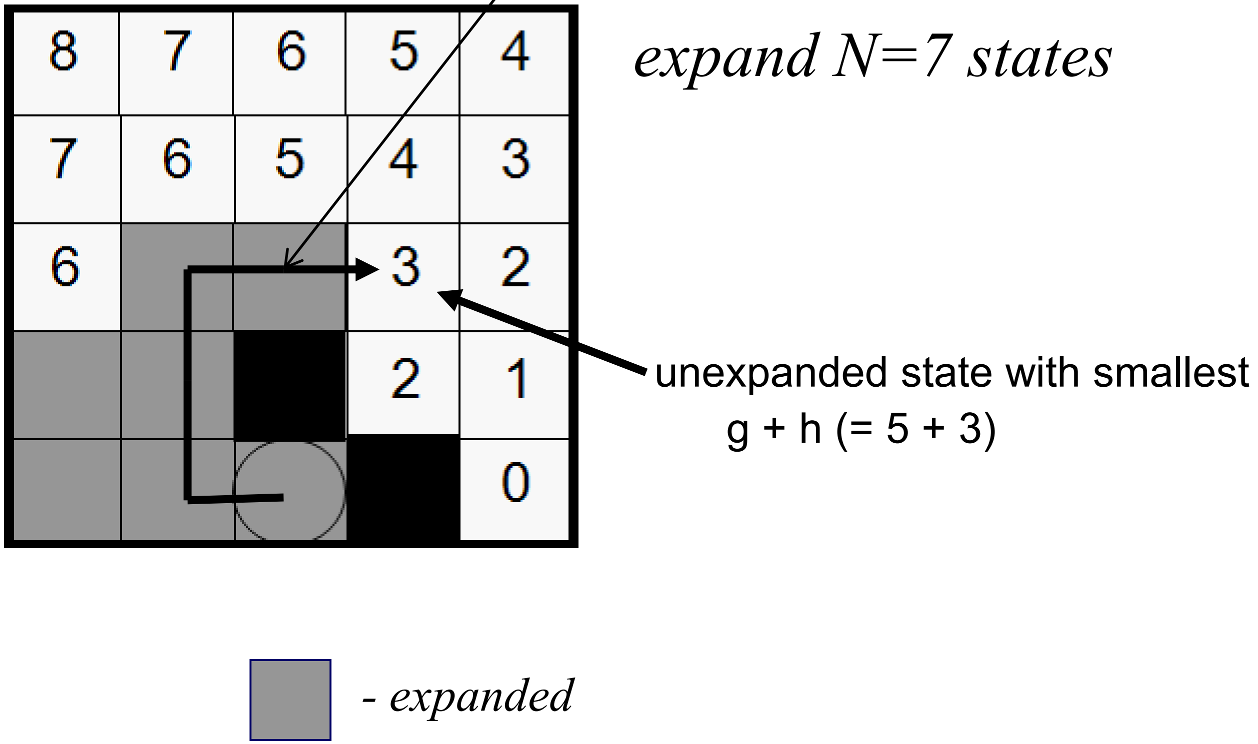

- expanded

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

How path is found?

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$

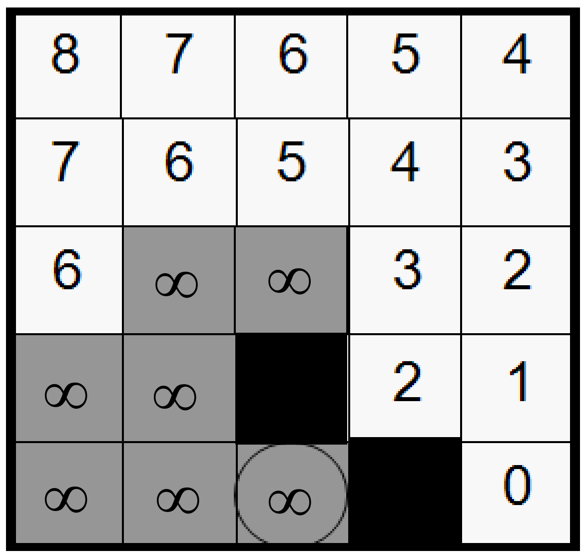


expand N=7 states

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



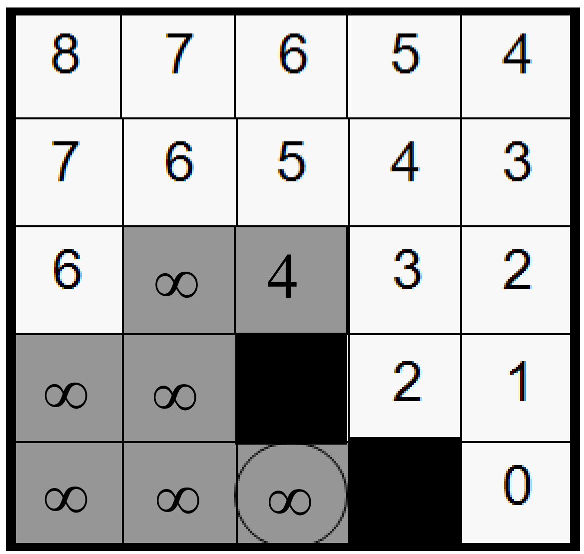
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s')+h(s'))$
until convergence

\- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$*



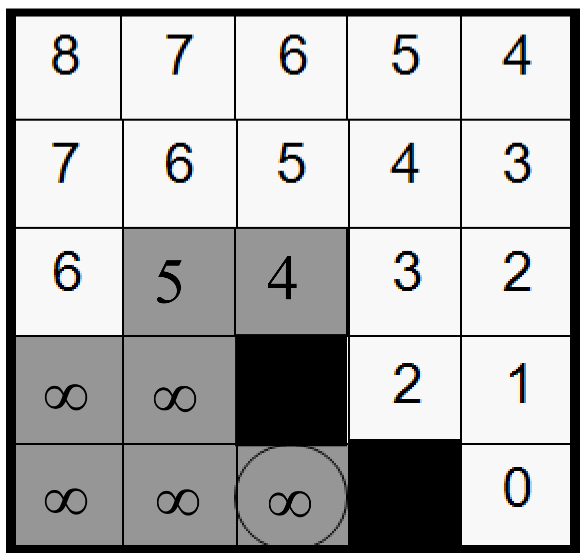
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s')+h(s'))$
until convergence

\- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



update h-values of expanded states via DP:

set h-values of expanded states to infinity

compute $h(s) = \min_{s' \in succ(s)} (c(s,s') + h(s'))$

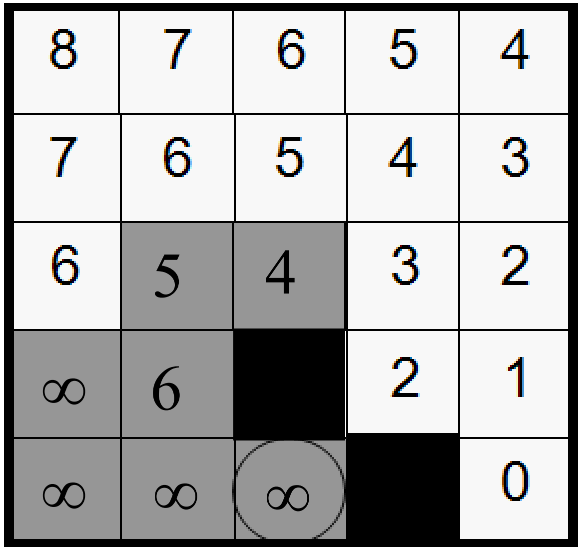
until convergence

- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state* $s = \arg\min_{s' \in OPEN} g(s') + h(s')$



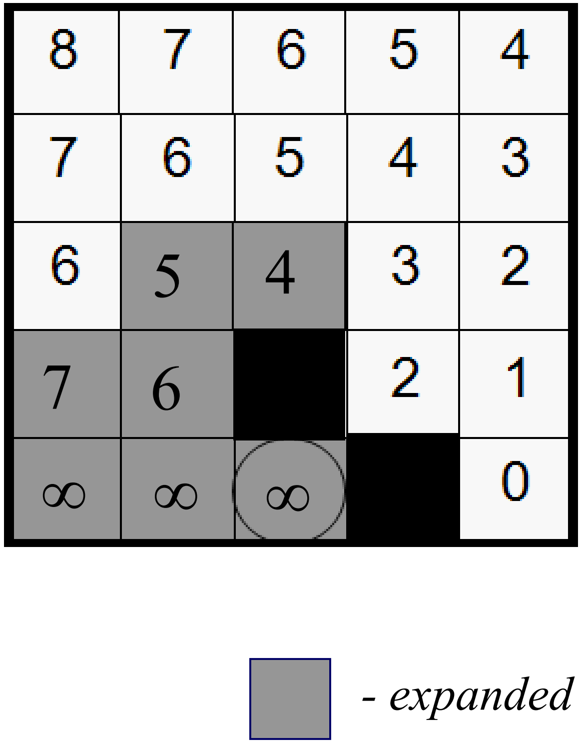
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s') + h(s'))$
until convergence

■ - *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



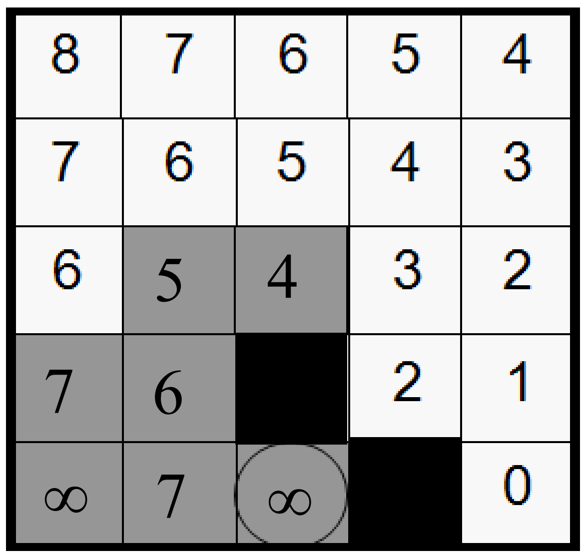
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s')+h(s'))$
until convergence

Does it matter in what order?

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states by Dynamic Programming (DP)*
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



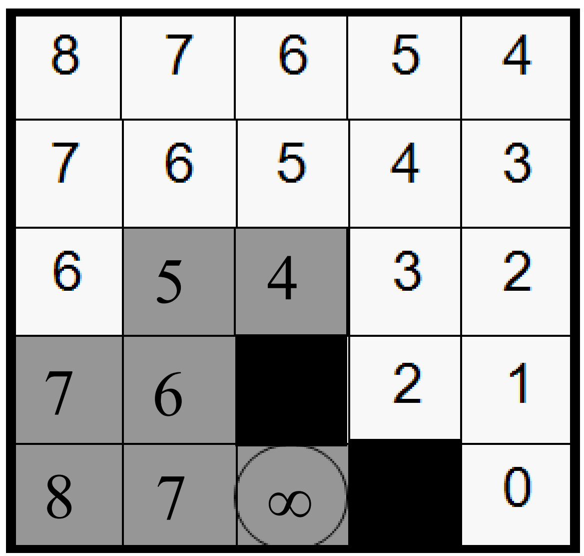
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s')+h(s'))$
until convergence

- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



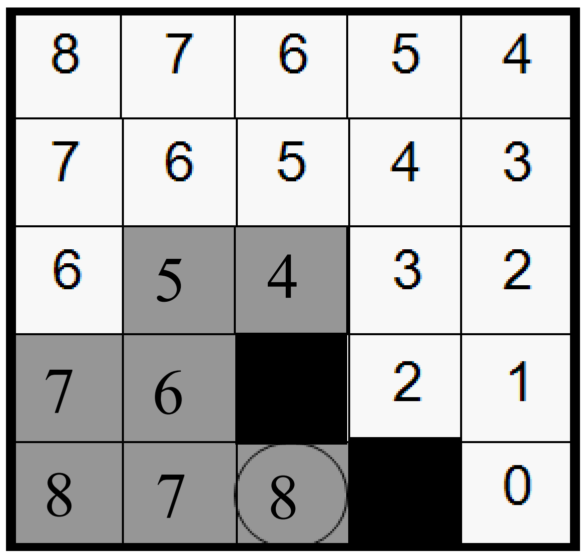
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s') + h(s'))$
until convergence

\- *expanded*

Learning Real-Time A* (LRTA*)

- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$*



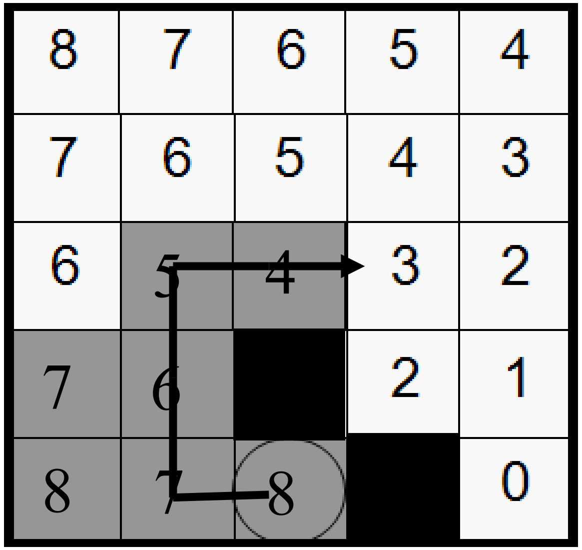
update h-values of expanded states via DP:
set h-values of expanded states to infinity
compute $h(s) = \min_{s' \in succ(s)} (c(s,s') + h(s'))$
until convergence

\- *expanded*

Learning Real-Time A* (LRTA*)

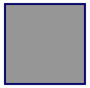
- LRTA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming (DP)*
3. *move on the path to state $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$*



make a move along the found path and repeat steps 1-3

Drawbacks compared to A?*

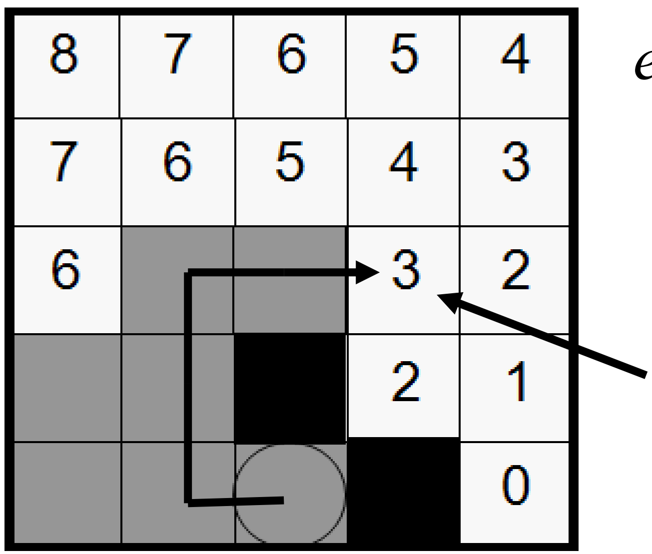


Real-time Adaptive A* (RTAA*)

- RTAA* with $N \geq 1$ expands: **LRTA***

one linear pass,
and even that can be lazy(postponed)

1. *expand N states*
2. *update h-values of expanded states u by $h(u) = f(s) - g(u)$,*
   where $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$
3. *move on the path to state $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$*



expand N=7 states

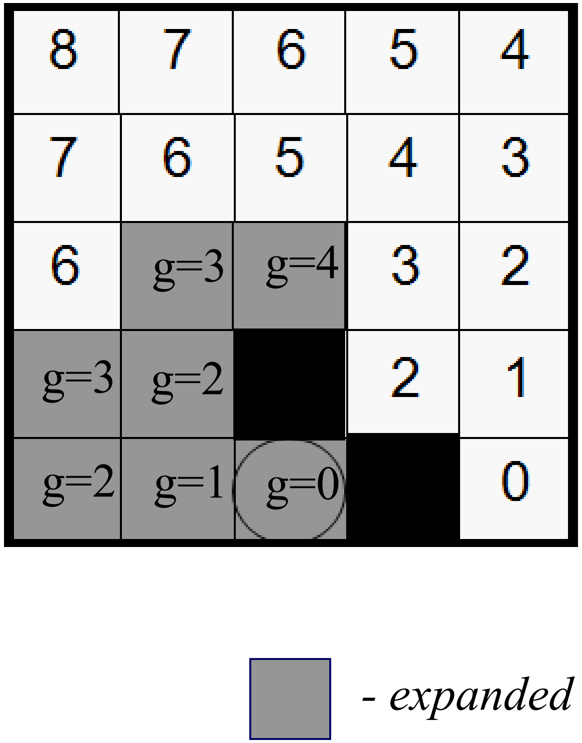
unexpanded state *s* with smallest $g + h$ (= 5 + 3)

\- *expanded*

Real-time Adaptive A* (RTAA*)

- RTAA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states u by h(u) = f(s) – g(u),*
   where $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$
3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



update all expanded states u:

h(u) = f(s) – g(u)

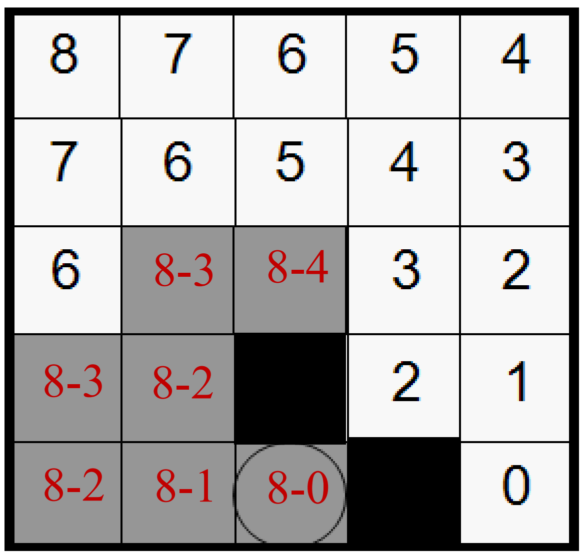
unexpanded state *s* with smallest
f(s) = 8

Real-time Adaptive A* (RTAA*)

- RTAA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states u by* $h(u) = f(s) - g(u)$,

     where $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



update all expanded states u:

$h(u) = f(s) - g(u)$

unexpanded state *s* with smallest *f(s)* = 8

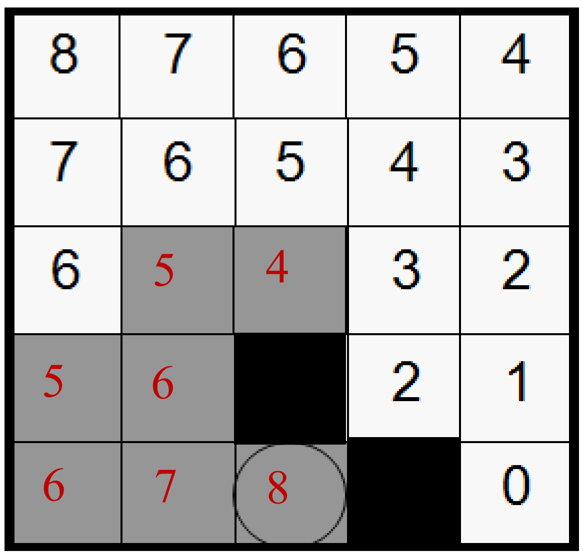
- *expanded*

Real-time Adaptive A* (RTAA*)

- RTAA* with $N \geq 1$ expands

1. *expand N states*
2. *update h-values of expanded states u by* $h(u) = f(s) - g(u)$,

   where $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$
3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$



update all expanded states u:

$h(u) = f(s) - g(u)$

unexpanded state *s* with smallest *f(s)* = 8

- *expanded*

Real-time Adaptive A* (RTAA*)

- RTAA* with $N \geq 1$ expands

  1. *expand N states*
  2. *update h-values of expanded states u by* $h(u) = f(s) - g(u)$*,*

     where $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$
  3. *move on the path to state* $s = \text{argmin}_{s' \in OPEN}\, g(s') + h(s')$

proof of admissibility:

h() – true cost-to-goal*

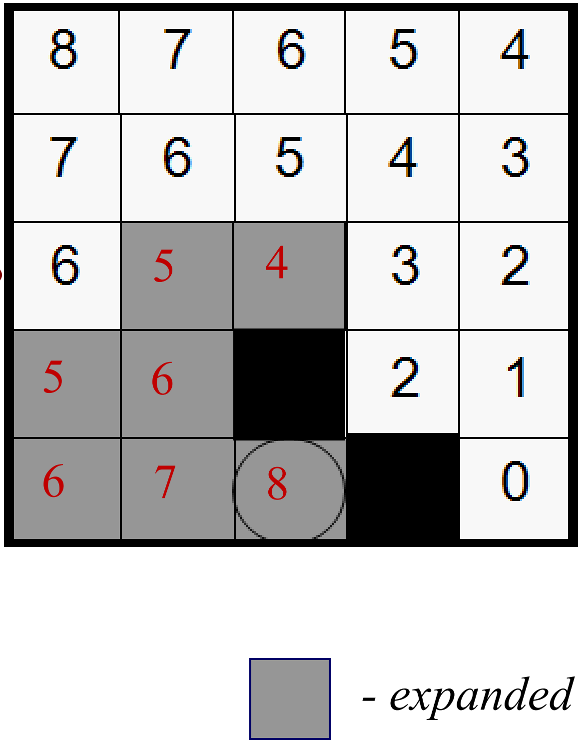
$g(u) + h^*(u) \geq h^*(s_{start})$

$h^*(u) \geq h^*(s_{start}) - g(u)$

because $f(s) \leq h^*(s_{start})$

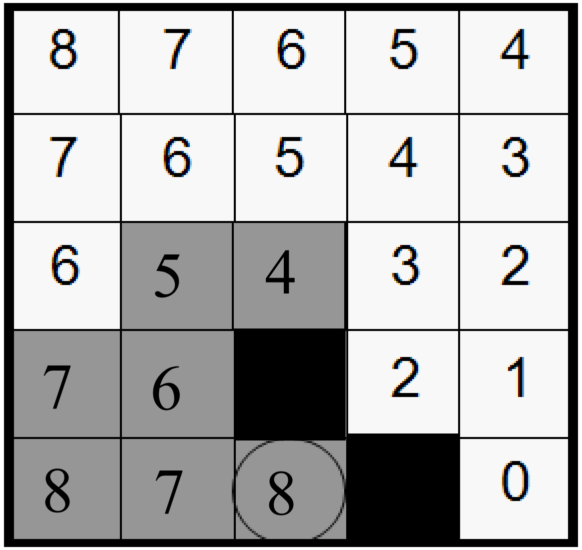
$h^*(u) \geq f(s) - g(u)$

$h^*(u) \geq h_{updated}(u)$

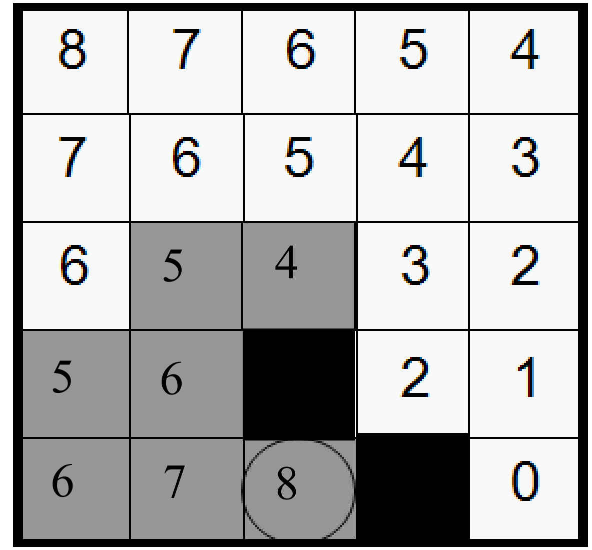


LRTA* vs. RTAA*

LRTA*



RTAA*



- Update of *h*-values in RTAA* is much faster but not as informed
- Both guarantee adimssibility and consistency of heuristics
- For both, heuristics are monotonically increasing
- Both guarantee to reach the goal in a finite number of steps (given the conditions listed previously)

What You Should Know…

- What is Real-time Heuristic Search and what are the challenges associated with it
- Operation of LRTA*
- Operation of RTAA*
- Pros/cons of LRTA* vs. A*