ECE276B: Planning & Learning in Robotics
Lecture 7: Anytime, Incremental, and Agent-centered Search

Instructor:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Zhichao Li: zhl355@eng.ucsd.edu

Ehsan Zobeidi: ezobeidi@eng.ucsd.edu

Ibrahim Akbar: iakbar@eng.ucsd.edu

UC San Diego

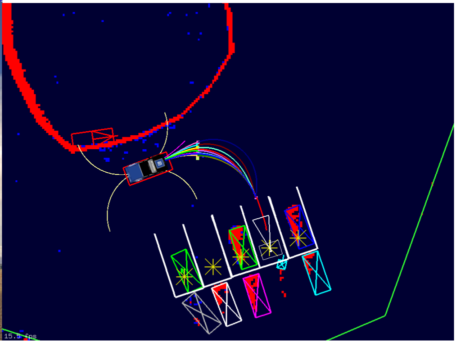
JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

Anytime, Incremental, and Agent-centered Search

- There are three important situations that happen in practice but our vanilla label correcting algorithms do not handle:
  1. How should we plan the best possible path in a given, fixed amount of time? (**Anytime Search**)
  2. How should we reuse a previous plan (rather than computing it from scratch) in a dynamic or partially known environment, where the edge costs c_{ij} change? (**Incremental Search**)
  3. How should we plan in really large environments, where it is impossible to compute the path all the way to the goal? (**Agent-centered Search**)

Anytime, Incremental, and Agent-centered Search

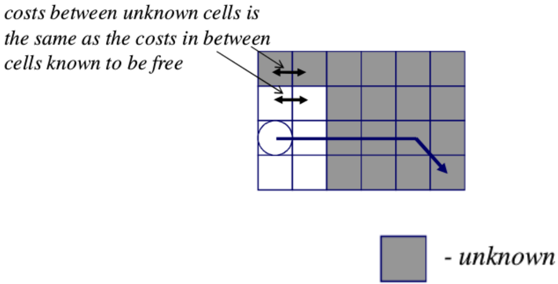


- CMU's autonomous car used anytime, incremental, agent-centered search (Anytime D*) in the DARPA Urban Challenge in 2007
- Likhachev and Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," IJRR'09
- Paper: `http://journals.sagepub.com/doi/pdf/10.1177/0278364909340445`
- Video: `https://www.youtube.com/watch?v=4hFhl0Oi8KI`
- Video: `https://www.youtube.com/watch?v=qXZt-B7iUyw`

Agent-centered Search

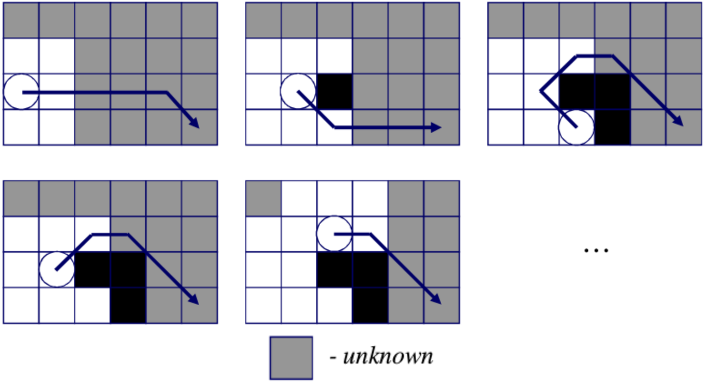
Planning in Large Unknown Environments

- **Freespace Assumption**: unknown space is free – costs between unknown cells are the same as between free cells
- Move the robot on a shortest potentially unblocked path and replan whenever new sensor information is received



Planning in Large Unknown Environments

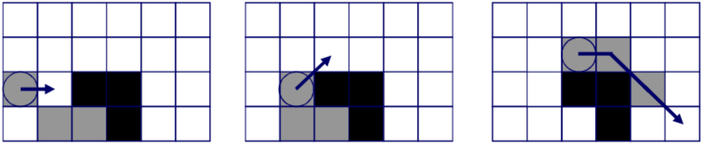
- A constantly updating map requires **a lot** of replanning!



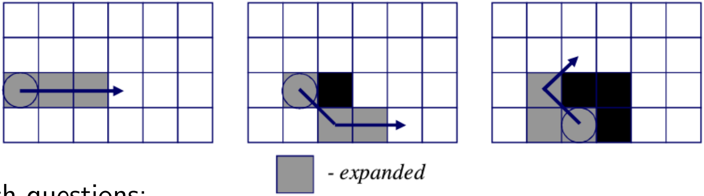
- Anytime incremental planning helps but what if the map is large and we cannot plan all the way to the goal even a single time?
- **Agent-centered Search**: places a strict limit on the amount of computation

Agent-centered Search

- Agent-centered search with a freespace assumption:
  1. Compute a partial path by expanding at most N nodes around the robot
  2. Move once, incorporate sensor information, and repeat
- Example in a known terrain:



- Example in an unknown terrain:



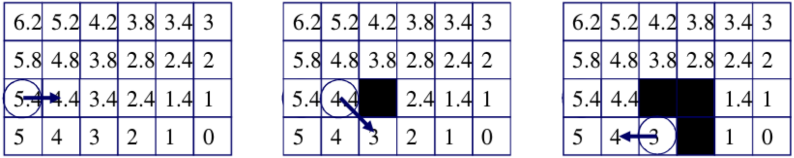
- Research questions:
  - how to compute a partial path
  - how to guarantee that the goal is eventually reached
  - how to provide bounds on the number of steps before reaching the goal

Learning Real-Time A* (LRTA*)

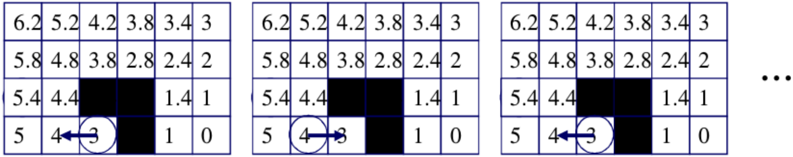
- Repeatedly move to the most promising adjacent cell using a heuristic:

$$s = \underset{j \in Children(s)}{\arg\min} \; c_{sj} + h_j$$

- Example: $h_i = \max\{|x_i - x_T|, |y_i - y_T|\} + 0.4 \min\{|x_i - x_T|, |y_i - y_T|\}$

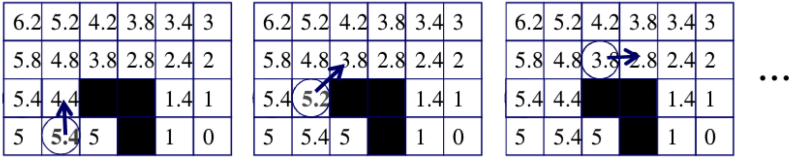


- Problem: this myopic behavior cannot overcome local minima!



Learning Real-Time A* (LRTA*)

- **Idea**: the heuristic needs to be updated over time!
- Repeatedly move to the most promising adjacent cell using **and updating** a heuristic:
  1. Update: $h_s = \min_{j \in Children(s)} c_{sj} + h_j$
  2. Move: $s = \underset{j \in Children(i)}{\arg\min} \; c_{sj} + h_j$

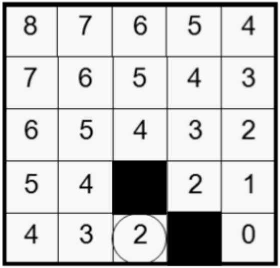


- The heuristic updates make h more informed while ensuring it remains admissible and consistent
- The robot is guaranteed to reach the goal in a finite number of steps if:
  - All edge costs are bounded from below: $c_{ij} \geq \Delta > 0$
  - The graph is finite size and there exists a finite-cost path to the goal
  - All actions are reversible – ensures that we do not get stuck in a local min

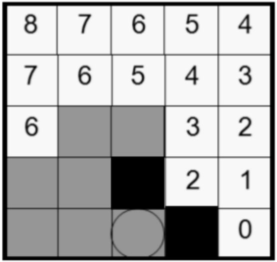
Learning Real-Time A* (LRTA*)

- LRTA* is related to limited-horizon A* ($N = 1$) because it makes a move towards the node j in OPEN with smallest $g_j + h_j = c_{sj} + h_j$ value
- LRTA* with $N \geq 1$ expands:
  1. Expand N nodes
  2. Update h-values of expanded nodes via Dynamic Programming (necessary to guarantee that the goal is reached):
     - Initialize: $h_i = \infty$ for all i in CLOSED
     - Repeat: $h_i = \min_{j \in Children(i)}(c_{ij} + h_j)$
  3. Move on the path to state $j^* = \arg\min_{j \in OPEN} g_j + h_j$. This node minimizes the cost to it plus the heuristic estimate of the remaining distance to the goal, i.e., it looks promising in terms of the whole path from the current robot state to the goal.

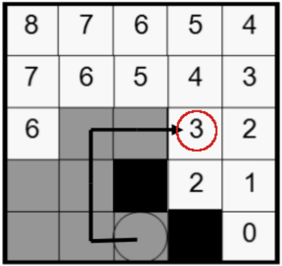
Example: Learning Real-Time A* (LRTA*)



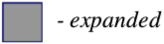
(a) 4-connected grid with Manhattan heuristic



(b) Expand $N = 7$ nodes



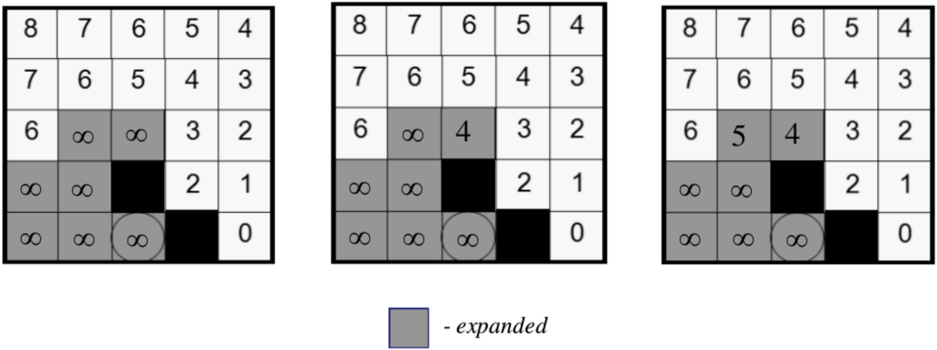
(c) Unexpanded node with smallest $f = 5 + 3$ value



Example: Learning Real-Time A* (LRTA*)

- Update h-values of expanded nodes via Dynamic Programming

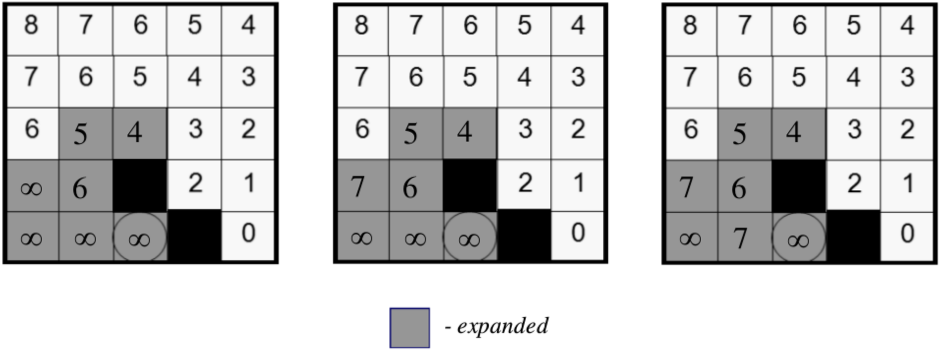
$$h_i = \min_{j \in Children(i)} (c_{ij} + h_j)$$



Example: Learning Real-Time A* (LRTA*)

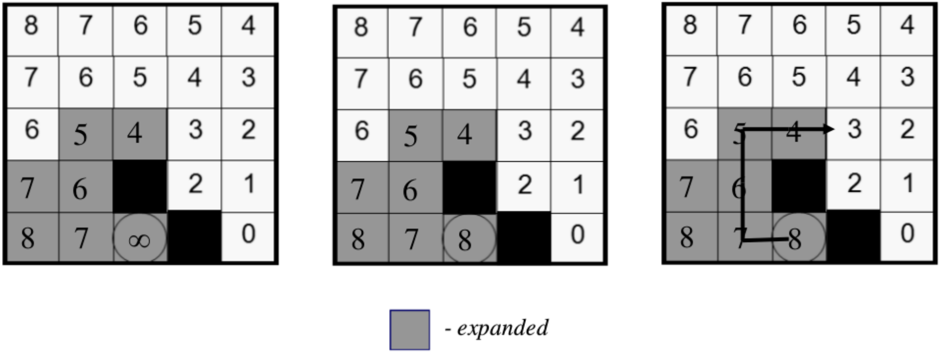
- Update h-values of expanded nodes via Dynamic Programming

$$h_i = \min_{j \in Children(i)} (c_{ij} + h_j)$$



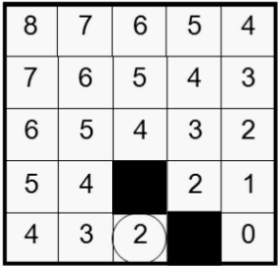
Example: Learning Real-Time A* (LRTA*)

- Repeat:
  1. Expand N nodes
  2. Update h-values of expanded nodes by Dynamic Programming
  3. Make one move along the shortest path to the unexpanded node in OPEN with smallest f value

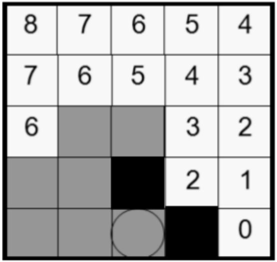


Real-time Adaptive A* (RTAA*)

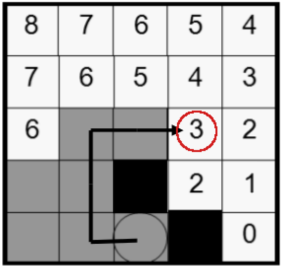
- RTAA* with $N \geq 1$ expands
  1. Expand N nodes
  2. Update h-values of expanded nodes i by $h_i = f_{j^*} - g_i$ where $j^* = \arg\min_{j \in OPEN} g_j + h_j$ (only a single pass through the nodes in CLOSED!)
  3. Move on the path to state $j^* = \arg\min_{j \in OPEN} g_j + h_j$
- **Proof of admissability**: $V^*_{i,\tau} \geq V^*_{s,\tau} - g_i \geq f_{j^*} - g_i = h_i$



(a) 4-connected grid with Manhattan heuristic



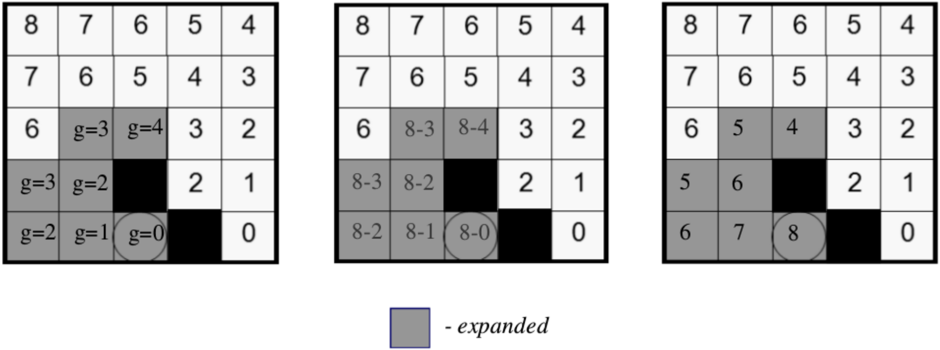
(b) Expand $N = 7$ states



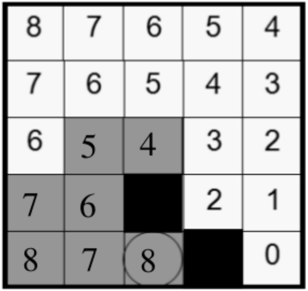
(c) Unexpanded state with smallest $f = 5 + 3$

Real-time Adaptive A* (RTAA*)

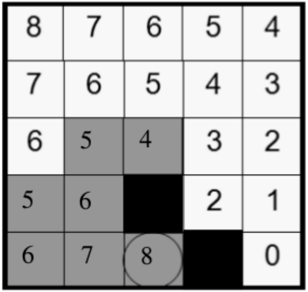
- Unexpanded state j^* with smallest $f_{j^*} = 8$
- Update h-values of expanded nodes: $h_i = f_{j^*} - g_i$



LRTA* vs RTAA*



(a) LRTA*



(b) RTAA*

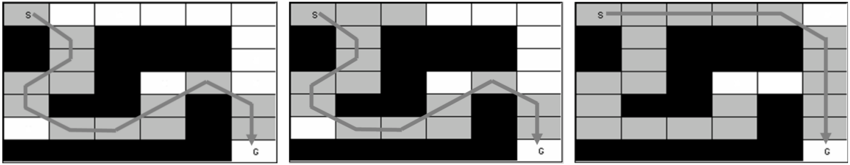
- Update of *h*-values in RTAA* is much faster but not as informed
- Both guarantee admissible and consistent heuristics
- Heuristics are monotonically increasing for both
- Both guarantee that the goal is reached in a finite number of steps (given the conditions listed previously)

Anytime Search

Anytime Search

- **Objective**: return the best plan possible within a fixed planning time
- **Idea**: run a series of weighted A* searches with decreasing ϵ:

$\varepsilon = 2.5$ | $\varepsilon = 1.5$ | $\varepsilon = 1.0$



13 expansions solution=11 moves | *15 expansions solution=11 moves* | *20 expansions solution=10 moves*

- This is inefficient because many labels (g-values) remain the same between search iterations yet we are recomputing them from scratch
- **Anytime Repairing A* (ARA*)**: an algorithm that is able to reuse the results from previous searches

Reusing Labels from a Previous Search

- **Idea**: mark nodes whose g-values have changed since the last expansion
- v-**value**: the g-value of a node at the time of its last expansion
  - $v_i = \infty$ for nodes that were never expanded
  - $g_j = \min_{i \in Parents(j)} v_i + c_{ij}$ for all nodes

Algorithm 1 A* that keeps track of inconsistent nodes

OPEN $\leftarrow \{s\}$, CLOSED $\leftarrow \{\}$, $\epsilon \geq 1$
$g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
$v_i = \infty$ for all $i \in \mathcal{V}$
ComputePath()

function ComputePath()
    while $f_\tau > \min_{i \in OPEN} f_i$ **do** ▷ τ is not the most promising node yet
        Remove i with smallest $f_i := g_i + \epsilon h_i$ from OPEN
        Insert i into CLOSED; $v_i = g_i$
        for $j \in$ Children(i) **do**
            if $g_j > (g_i + c_{ij})$ **then**
                $g_j \leftarrow (g_i + c_{ij})$
                if $j \notin$ CLOSED **then** insert j into OPEN
                otherwise insert j into INCONS

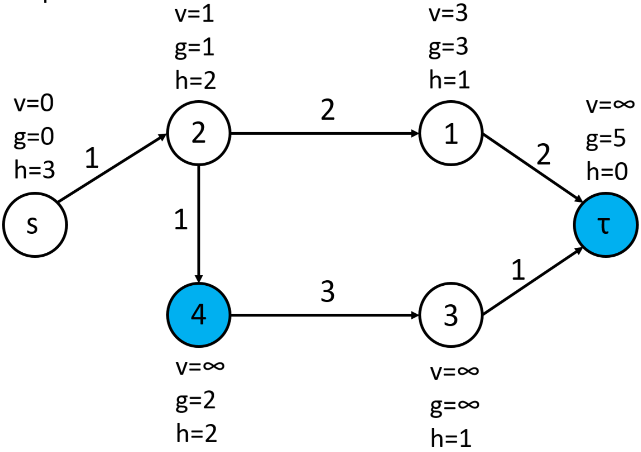
Ensure no node is expanded multiple times

Reusing Labels from a Previous Search

- **Consistent node**: a node i such that $v_i = g_i$
- **Overconsistent node**: a node i such that $v_i > g_i$
- All $i \in$ OPEN are overconsistent because $v_i = \infty > g_i$
- **Alternative view**: A* expands overconsistent nodes in the order of their f-values
- All you need to do to make A* reuse previous information is to initialize OPEN with all overconsistent nodes!
  - A* (consistent heuristic): OPEN is initialized with the OPEN set from a previous search since a consistent heuristic ensures that all nodes in CLOSED remain consistent
  - Weighted A* (ϵ-consistent heuristic): OPEN is initialized with the OPEN set from a previous search and all nodes in CLOSED whose g-values decreased after entering CLOSED (INCONS)

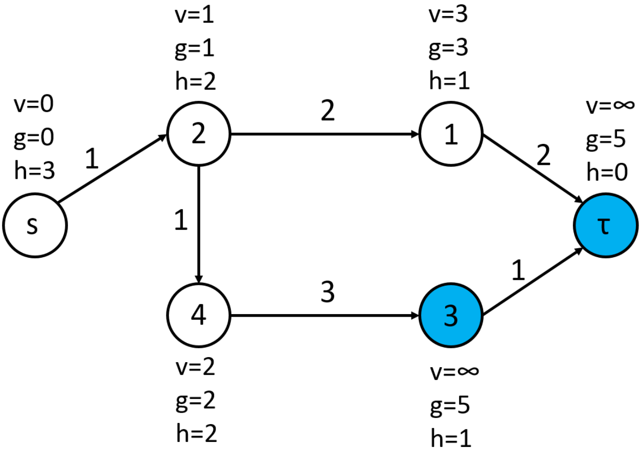
Example: Reusing Labels from a Previous Search

- OPEN contains all overconsistent nodes initially
- **Invariant** maintained throughout the search: $g_j = min_{i \in Parents(j)} v_i + c_{ij}$
- $OPEN = \{4, \tau\}$
- $CLOSED = \{\}$
- Next to expand: 4



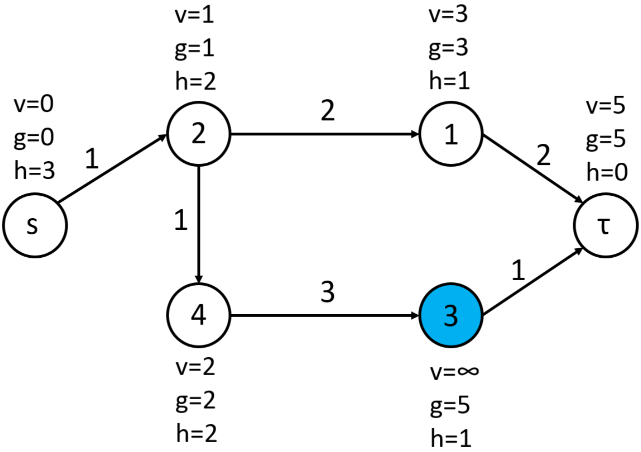
Example: Reusing Labels from a Previous Search

- *OPEN* $= \{3, \tau\}$
- *CLOSED* $= \{4\}$
- Next to expand: τ



Example: Reusing Labels from a Previous Search

- $\mathit{OPEN} = \{3\}$
- $\mathit{CLOSED} = \{4, \tau\}$
- Done



Anytime Repairing A* (ARA*)

- Efficient series of weighted A* searches with decreasing ϵ
- Need to keep track of all overconsistent nodes $= \mathit{OPEN} \cup \mathit{INCONS}$

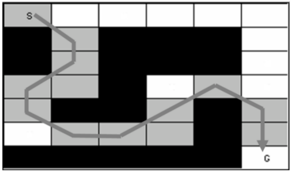
Algorithm 2 ARA*

Set ϵ to large value
OPEN $\leftarrow \{s\}$
$g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
$v_i = \infty$ for all $i \in \mathcal{V}$
while $\epsilon \geq 1$ **do**
    CLOSED $\leftarrow \{\}$; INCONS $\leftarrow \{\}$
    OPEN, INCONS $\leftarrow$ ComputePath()
    Publish current ϵ suboptimal solution
    Decrease ϵ
    OPEN $=$ OPEN $\cup$ INCONS $\triangleright$ Initialize OPEN with all overconsistent nodes

Repeated A* vs ARA*

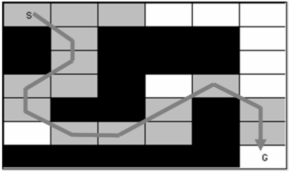
- A series of weighted A* searches (**no g-value reuse**)

$\varepsilon = 2.5$



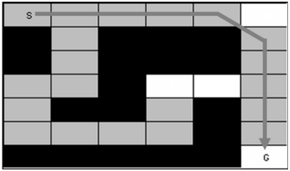
13 expansions
solution=11 moves

$\varepsilon = 1.5$



15 expansions
solution=11 moves

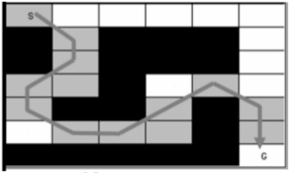
$\varepsilon = 1.0$



20 expansions
solution=10 moves

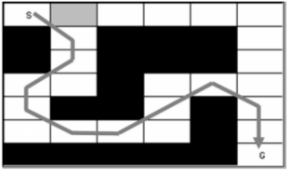
- Anytime Repairing A* (**ARA***)

$\varepsilon = 2.5$



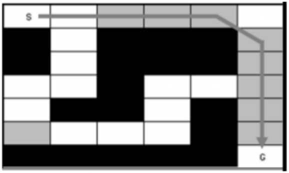
13 expansions
solution=11 moves

$\varepsilon = 1.5$



1 expansion
solution=11 moves

$\varepsilon = 1.0$



9 expansions
solution=10 moves

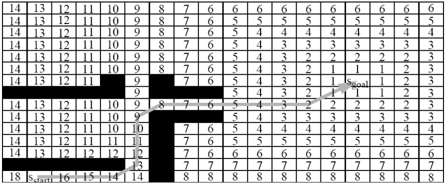
Incremental Search

Unknown, Dynamic Graphs

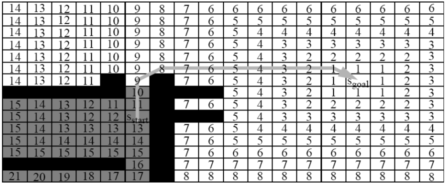
- So far, we have assumed that we know all edge costs and they don't change
- This is not the case in practice when the environment is partially unknown or changing
- **Naive solution**: recompute the path any time an edge cost changes
- **Lifelong Planning A*** (LPA*):
  - Assumes edge costs change over time but the robot has not actually moved yet
  - Recomputes the path from start to goal while reusing as much information as possible
- **D* and D* Lite**:
  - The robot starts moving on the path to goal and updates edge costs along the way as the sensors observe new obstacles or free areas
  - Recomputes the path from the current node to the goal while reusing as much information as possible
- Many other variations: Anytime D*, Field D*, Theta*, ...

Motivation for Incremental Search

- Optimal g-values for a backwards search:



(a) cost of least-cost path to τ initially



(b) cost of least-cost path to τ after a door turns out to be closed

- Can the g-values from the first search be re-used in the second search?
- Would the number of changed g-values be different for forward A*?

Map Changes and Underconsistent Nodes

- So far, ComputePath() only distinguishes consistent and overconsistent nodes, i.e., $v_i \geq g_i$
- Edge cost increases may introduce **underconsistent nodes** ($v_i < g_i$) which violates the ComputePath() **invariant**: $g_j = min_{i \in Parents(j)} v_i + c_{ij}$
- **Idea**:
  1. Fix all underconsistent nodes by setting $v_i = \infty$, which makes them either overconsistent or consistent
  2. Propagate the changes to maintain the **invariant**: $g_j = \min_{i \in Parents(j)} v_i + c_{ij}$
- **Additional f-value requirement**: For a consistent or overconsistent node i that can belong to some path from s to τ, we require that all underconsistent nodes j that could be on a path from s to i are expanded before i, i.e., $key_i > key_j$

  $key_i = [\min\{g_i, v_i\} + \epsilon h_i; \min\{g_i, v_i\}]$ (second value for tie breaking)

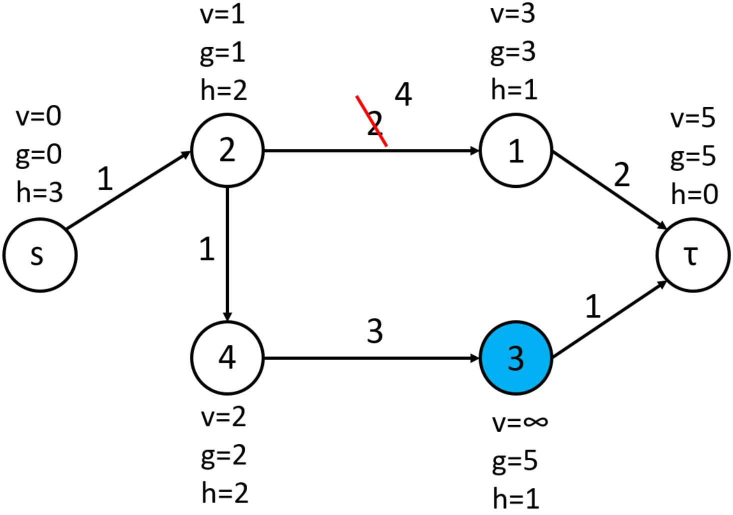
Lifelong Planning A*

Algorithm 3 LPA* ComputePath()

function $\textsc{UpdateMembership}(i)$
  if $v_i \neq g_i$ **then**
    if $i \notin$ CLOSED **then** Insert/Update i in OPEN with key_i
  else
    if $i \in$ OPEN **then** Remove i from OPEN
function $\textsc{ComputePath}()$
  while $key_\tau > \min_{j \in OPEN} key_j$ **or** $v_\tau < g_\tau$ **do**
    Remove i with smallest key_i from OPEN
    if $v_i > g_i$ (overconsistent) **then**
      $v_i = g_i$; Insert i into CLOSED
      for $j \in$ Children(i) **do**
        if $g_j > (g_i + c_{ij})$ **then**
          $g_j \leftarrow (g_i + c_{ij})$
          $\textsc{UpdateMembership}(j)$
    else (underconsistent)
      $v_i = \infty$; $\textsc{UpdateMembership}(i)$
      for $j \in$ Children(i) **and** $j \neq s$ **do**
        $g_j = \min_{k \in Parents(j)} v_k + c_{kj}$
        $\textsc{UpdateMembership}(j)$

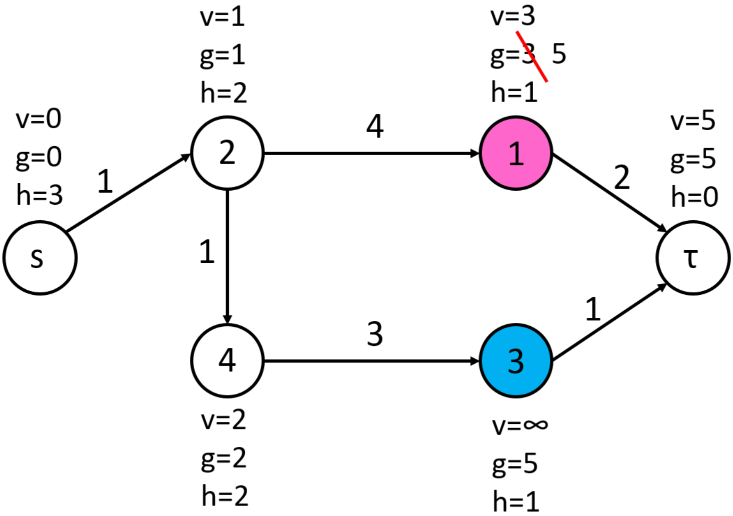
Example: Map Changes and Underconsistent Nodes

- Suppose that an edge cost changes
- Propagate the changes to maintain: $g_j = \min_{i \in Parents(j)} v_i + c_{ij}$



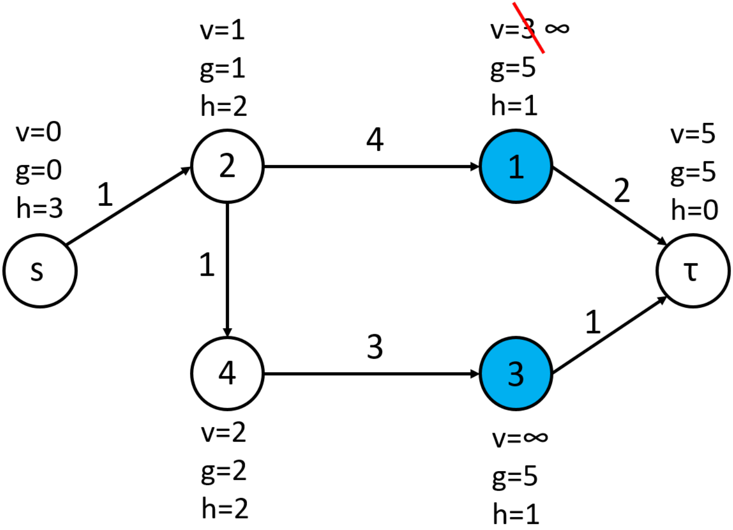
Example: Map Changes and Underconsistent Nodes

- This may introduce underconsistent nodes ($v_i < g_i$)
- OPEN $= \{1, 3\}$, CLOSED $= \{s, 2, 4, \tau\}$
- Next to expand: 1 (underconsistent)



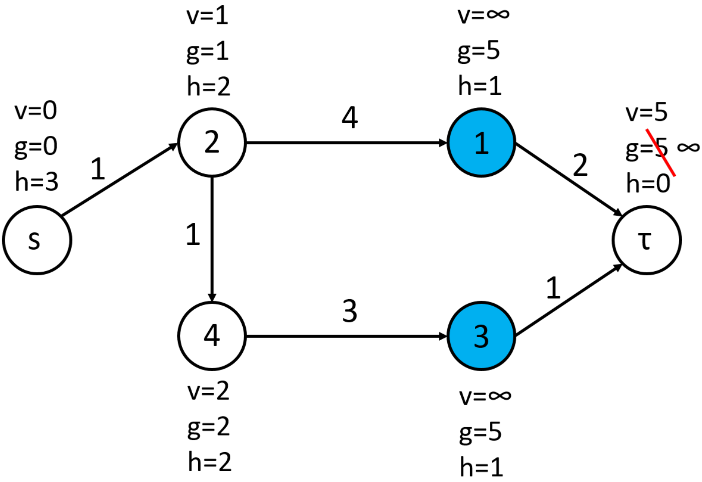
Example: Map Changes and Underconsistent Nodes

- Fix the underconsistent node by setting $v_1 = \infty$ and reinsert in OPEN



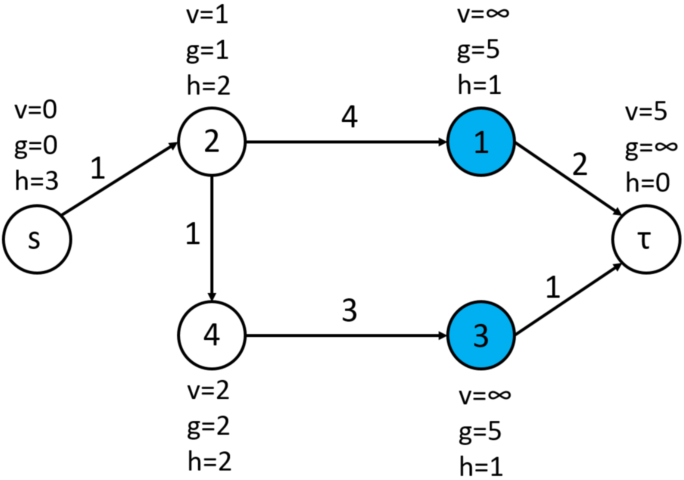
Example: Map Changes and Underconsistent Nodes

- Propagate the changes to maintain: $g_j = \min_{i \in Parents(j)} v_i + c_{ij}$



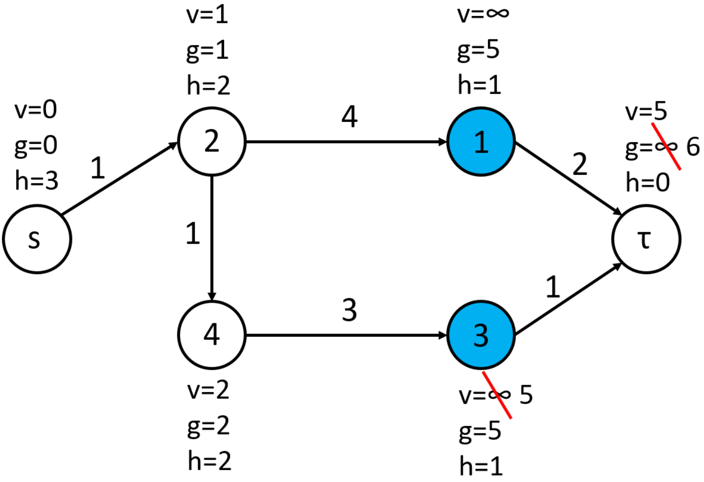
Example: Map Changes and Underconsistent Nodes

- OPEN $= \{1, 3\}$, CLOSED $= \{s, 2, 4, \tau\}$
- Next to expand: 3 (overconsistent)



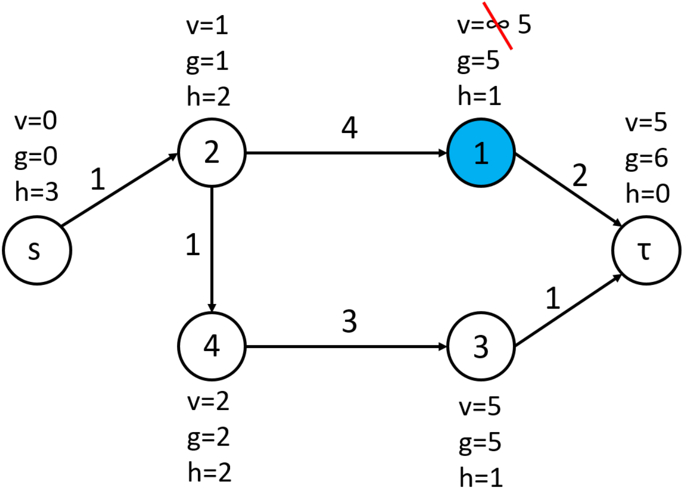
Example: Map Changes and Underconsistent Nodes

- Expand 3 and insert in CLOSED



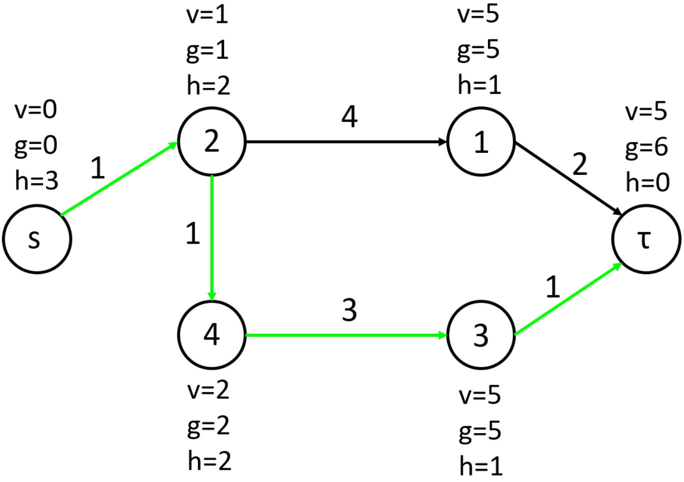
Example: Map Changes and Underconsistent Nodes

- OPEN $= \{1\}$, CLOSED $= \{s, 2, 4, \tau, 3\}$
- Next to expand: 1 (overconsistent)



Example: Map Changes and Underconsistent Nodes

- Done. Backtrack the optimal path.



D* Lite

- **Backward search**: The search is done backwards from the goal to the current state of the robot with all edges reversed because this way the root of the search tree remains the same and the g values are more likely to remain unchanged inbetween two calls to ComputePath()

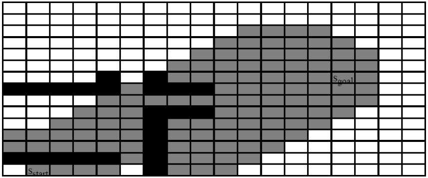
Algorithm 4 D* Lite

```
repeat
    ComputePath()                                   ▷ Modified to fix underconsistent nodes
    Publish optimal path
    Follow the path until the map is updated with new sensor information
    Update the corresponding edge costs
    Set τ to the current state of the agent
until τ is reached
```

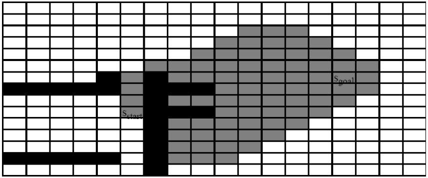
- Details in M. Likhachev, D. Ferguson, G. Gordon, A. Stenz, and S. Thrun, "Anytime search in dynamic graphs," Artificial Intelligence, 2012.

D* Lite (i.e., Incremental A*) vs A*

- Backward A* does not reuse g-values from previous searches:

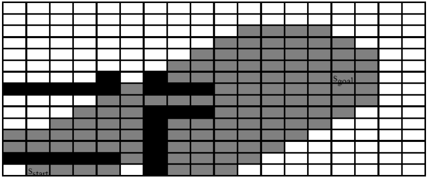


(a) initial search by backward A*

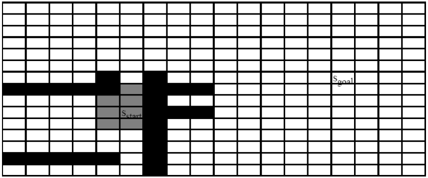


(b) second search by backward A*

- D* Lite reuses g-values from previous searches:



(a) initial search by D* Lite



(b) second search by D* Lite

Anytime and Incremental Planning

- Decrease ϵ and update edge costs at the same time
- Re-compute a path by reusing previous g values

Algorithm 5 Anytime D*

Set ϵ to large value
repeat
    ComputePath() $\triangleright$ Modified to fix underconsistent nodes
    Publish ϵ-suboptimal path
    Follow the path until the map is updated with new sensor information
    Update the corresponding edge costs
    Set τ to the current state of the agent
    if Significant changes were observed **then**
        Increase ϵ or replan from scratch
    else
        Decrease ϵ
until τ is reached