ECE276B: Planning & Learning in Robotics
Lecture 8: Anytime, Incremental, and Agent-Centered Search

Nikolay Atanasov

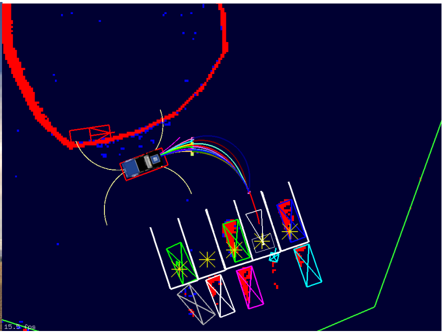
natanasov@ucsd.edu

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 vanilla label correcting algorithms do not handle:
  1. planning in very large environments, where it is impossible to compute a path all the way to the goal (**Agent-Centered Search**),
  2. planning the best possible path in a given fixed amount of time (**Anytime Search**)
  3. reusing a previous plan rather than recomputing it from scratch in a dynamically changing or partially known environment (**Incremental Search**)

Agent-Centered, Anytime, and Incremental Search

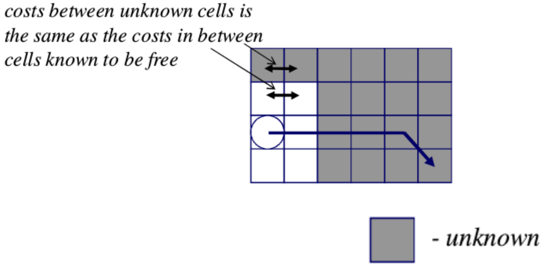


- CMU's autonomous car used agent-centered, anytime, incremental search based on the **anytime D* algorithm** in the DARPA Urban Challenge in 2007
- Likhachev and Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," IJRR, 2009, `http://journals.sagepub.com/doi/pdf/10.1177/0278364909340445`
- Video: `https://www.youtube.com/watch?v=4hFhl0Oi8KI`
- Video: `https://www.youtube.com/watch?v=qXZt-B7iUyw`

Outline

Planning in Large Unknown Environments

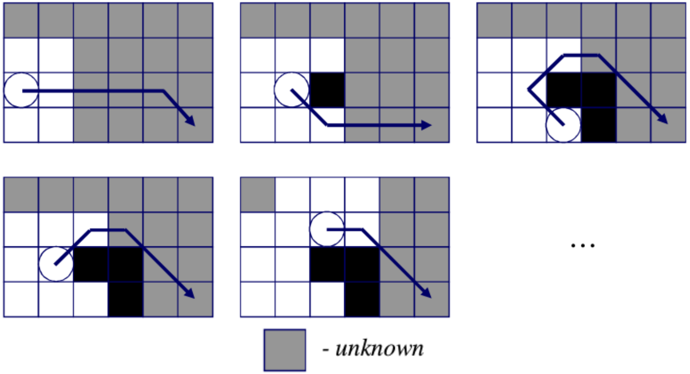
- **Freespace Assumption**: unknown space is free, i.e., costs between unknown cells are the same as between free cells
- Move the agent along a shortest potentially free path and replan whenever new sensor information is received



costs between unknown cells is the same as the costs in between cells known to be free
- unknown

Planning in Large Unknown Environments

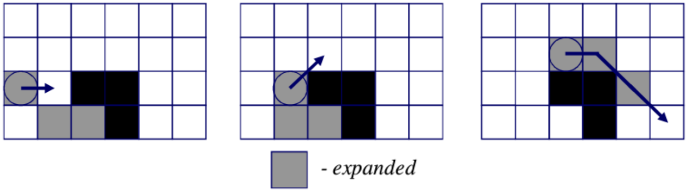
- A constantly updating map requires **a lot** of replanning!



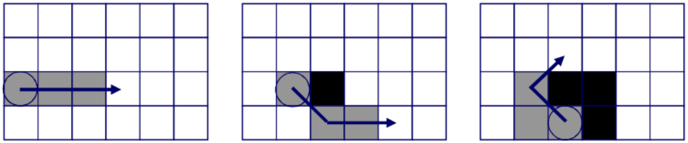
- **Agent-Centered Search**: places a strict limit on the amount of computation during replanning

Agent-Centered Search

- Agent-centered search with free-space assumption:
  1. compute a partial path by expanding at most N nodes around the agent
  2. move once, incorporate sensor information, and repeat
- Example in a known terrain:



- Example in an unknown terrain:



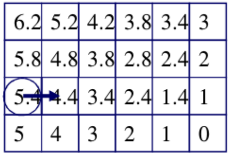
- Questions:
  - How to compute a partial path?
  - How to guarantee that the goal is eventually reached?
  - How to provide bounds on the number of steps to reach the goal?

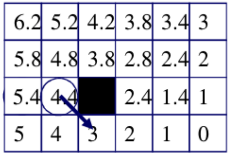
Agent-Centered Search

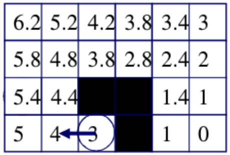
- Consider repeatedly moving from the start node s to the most promising adjacent node $j \in \mathit{Children}(s)$ using a heuristic h_j:

$$s = \operatorname*{arg\,min}_{j \in \mathit{Children}(s)} c_{sj} + h_j$$

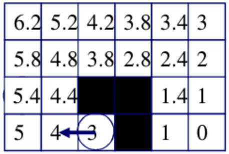
- Example: $h_i = \max\{|x_i - x_\tau|, |y_i - y_\tau|\} + 0.4 \min\{|x_i - x_\tau|, |y_i - y_\tau|\}$

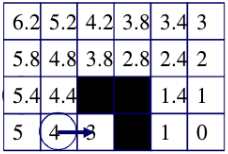


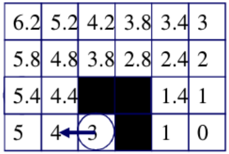




- Problem: this idea cannot overcome local minima in the heuristic function



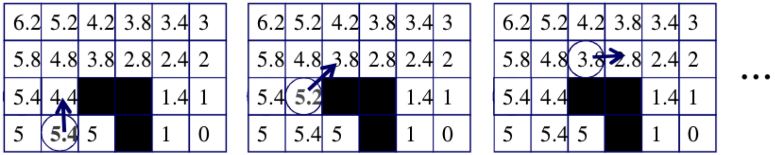




...

Learning Real-Time A* (LRTA*)

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

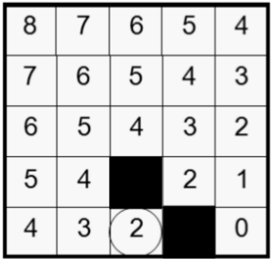


- The heuristic updates make h more informed while ensuring it remains admissible and consistent
- The agent is guaranteed to reach the goal in a finite number of steps if:
  - all edge costs are bounded from below: $c_{ij} \geq \delta > 0$
  - the graph is finite and there exists a finite-cost path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

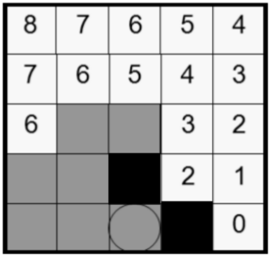
Learning Real-Time A* (LRTA*)

- LRTA* is related to A* with $N = 1$ node expansions because it makes a move towards the node j in OPEN with smallest $g_j + h_j = c_{sj} + h_j$ value
- LRTA* with $N \geq 1$ node expansions:
  1. Expand N nodes
  2. Update h-values of expanded nodes via dynamic programming (necessary to guarantee that the goal is eventually reached):
     - Initialize: $h_i = \infty$ for all i in CLOSED
     - Repeat: $h_i = \min_{j \in Children(i)}(c_{ij} + h_j)$ until convergence of h_i for all $i \in$ CLOSED
  3. Move on the path to node $j^* = \underset{j \in OPEN}{\arg\min}(g_j + h_j)$
     - Node j^* minimizes the cost-to-arrive 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 agent state to the goal.

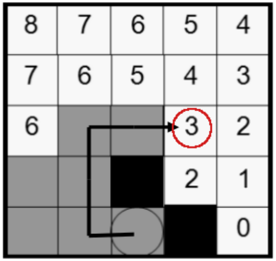
Example: Learning Real-Time A* (LRTA*)



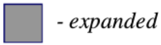
(a) 4-connected grid with Manhattan heuristic



(b) Expand $N = 7$ nodes



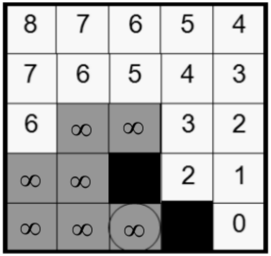
(c) Unexpanded node with smallest $f = 5 + 3$ value

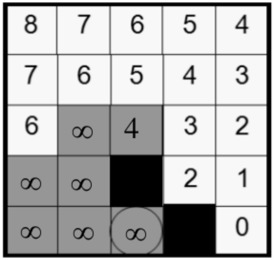


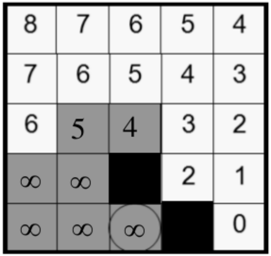
Example: Learning Real-Time A* (LRTA*)

- Update h-values of expanded nodes via dynamic programming

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





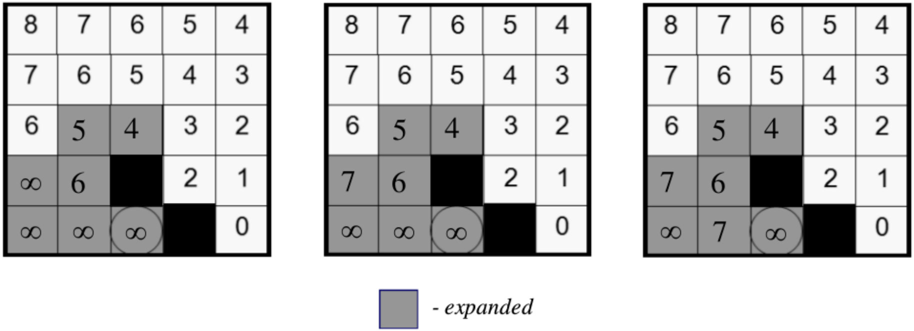


▨ - *expanded*

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

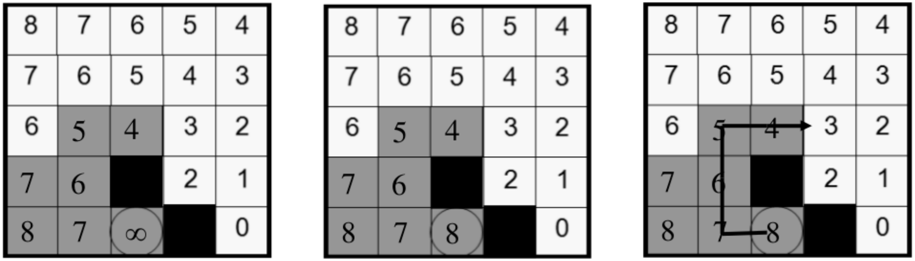
- Update h-values of expanded nodes via dynamic programming

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



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

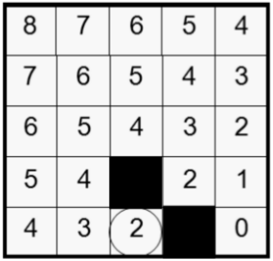
- Repeat:
  1. Expand N nodes
  2. Update h-values of expanded nodes by dynamic programming
  3. Make one move along the shortest path to the node in OPEN with the smallest f value



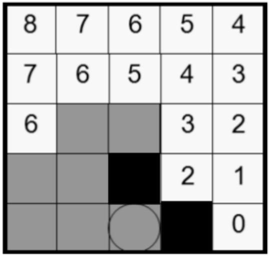
▨ - *expanded*

Real-time Adaptive A* (RTAA*)

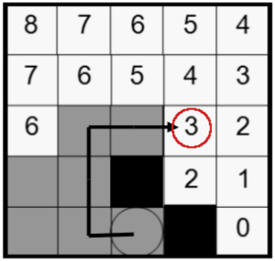
- RTAA* with $N \geq 1$ expands
  1. Expand N nodes
  2. Update h-values of expanded nodes i by $h_i = f_{j^*} - g_i$ where $j^* = \arg\min_{j \in OPEN}(g_j + h_j)$ (only a single pass through the nodes in CLOSED)
  3. Move on the path to node $j^* = \arg\min_{j \in OPEN} g_j + h_j$
- **Proof of admissability**: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



(a) 4-connected grid with Manhattan heuristic



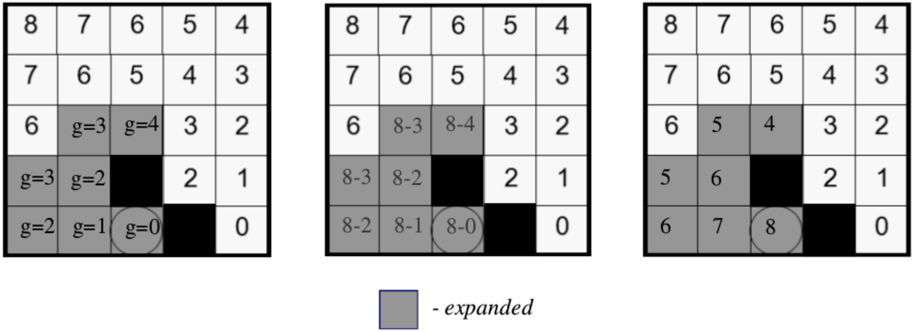
(b) Expand $N = 7$ states



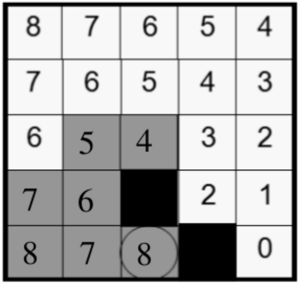
(c) Unexpanded state with smallest $f = 5 + 3$

Real-time Adaptive A* (RTAA*)

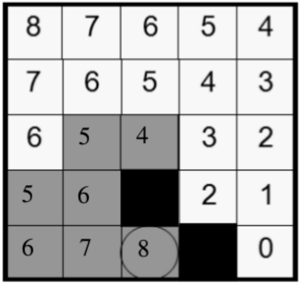
- Unexpanded node j^* with smallest $f_{j^*} = 8$
- Update h-values of expanded nodes: $h_i = f_{j^*} - g_i$



LRTA* vs RTAA*



(a) LRTA*



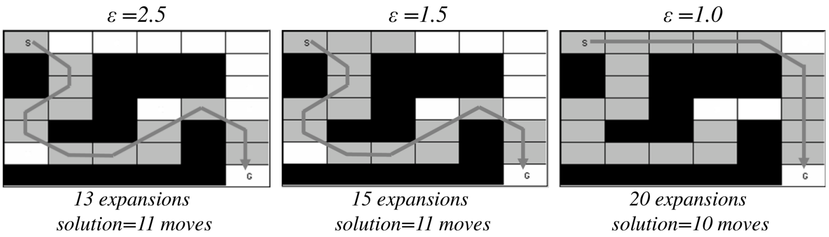
(b) RTAA*

- Update of *h*-values in RTAA* is much faster but not as informed
- Both algorithms guarantee admissible and consistent heuristic over time
- Heuristics are monotonically increasing for both
- Both guarantee that the goal is reached in a finite number of steps (assuming: bounded edge cost, finite graph, finite-cost path to goal, and reversible actions)

Outline

Anytime Search

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



- 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 information from a previous search

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
- **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 f-value order
- All we need to do to make A* reuse previous information is to initialize OPEN with all overconsistent nodes!

Reusing Labels from a Previous Search

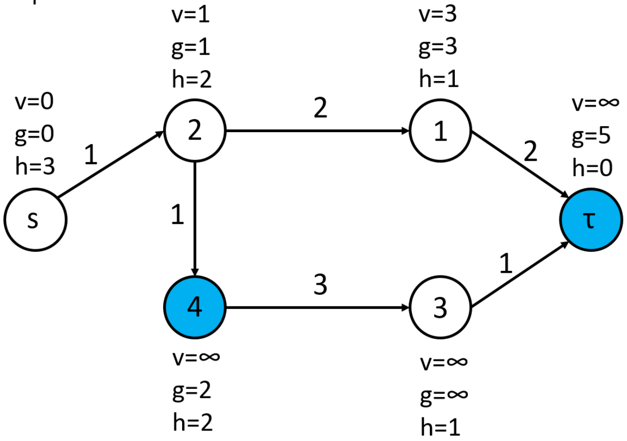
- A* (consistent heuristic): OPEN is initialized with the OPEN list from a previous search since a consistent heuristic ensures that all nodes in CLOSED remain consistent
- Weighted A* (ϵ-consistent heuristic): OPEN is initialized with the OPEN list from a previous search and an INCONS list of all nodes in CLOSED whose g-values decreased after entering CLOSED

Algorithm Weighted A* that keeps track of inconsistent nodes

```
 1: OPEN ← {$s$}, CLOSED ← {}, $\epsilon \geq 1$
 2: $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
 3: $v_i = \infty$ for all $i \in \mathcal{V}$
 4: ComputePath()
 5:
 6: function ComputePath()
 7:     while $f_\tau > \min_{i \in OPEN} f_i$ do          ▷ $\tau$ is not the most promising node yet
 8:         Remove $i$ with smallest $f_i := g_i + \epsilon h_i$ from OPEN
 9:         Insert $i$ into CLOSED; $v_i = g_i$
10:         for $j \in$ Children($i$) do
11:             if $g_j > (g_i + c_{ij})$ then
12:                 $g_j \leftarrow (g_i + c_{ij})$                              } Ensure no node is
13:                 if $j \notin$ CLOSED then insert or update $j$ in OPEN    } expanded multiple times
14:                 else insert $j$ into INCONS                               }
```

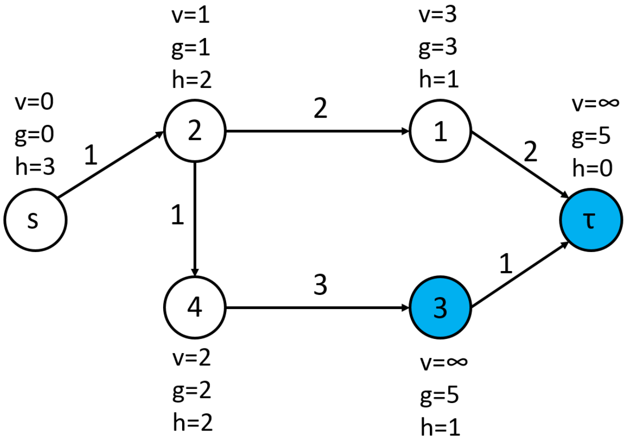
Example: Reusing Labels from a Previous Search

- 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



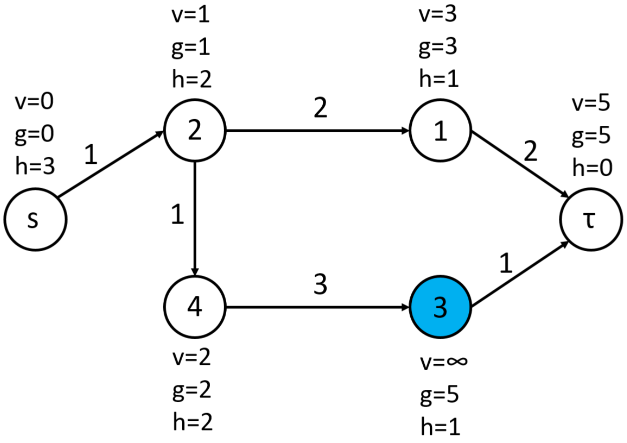
Example: Reusing Labels from a Previous Search

- $OPEN = \{3, \tau\}$
- $CLOSED = \{4\}$
- 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 $=$ *OPEN* $\cup$ *INCONS*

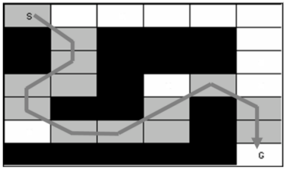
Algorithm ARA*

```
1:  Set $\epsilon$ to large value
2:  OPEN $\leftarrow \{s\}$
3:  $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
4:  $v_i = \infty$ for all $i \in \mathcal{V}$
5:  while $\epsilon \geq 1$ do
6:      CLOSED $\leftarrow \{\}$; INCONS $\leftarrow \{\}$
7:      OPEN, INCONS $\leftarrow$ ComputePath()
8:      Publish current $\epsilon$ suboptimal solution
9:      Decrease $\epsilon$
10:     OPEN $=$ OPEN $\cup$ INCONS          ▷ Initialize OPEN with all overconsistent nodes
```

Repeated A* vs ARA*

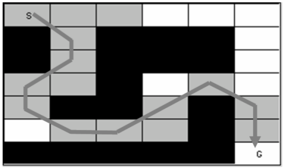
- A series of weighted A* searches (**no g-value reuse**)

$\varepsilon = 2.5$



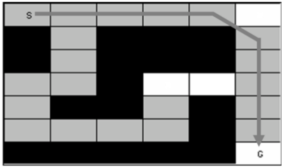
13 expansions
solution=11 moves

$\varepsilon = 1.5$



15 expansions
solution=11 moves

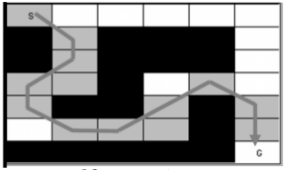
$\varepsilon = 1.0$



20 expansions
solution=10 moves

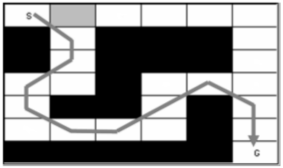
- Anytime Repairing A* (**ARA***)

$\varepsilon = 2.5$



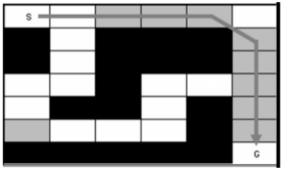
13 expansions
solution=11 moves

$\varepsilon = 1.5$



1 expansion
solution=11 moves

$\varepsilon = 1.0$



9 expansions
solution=10 moves

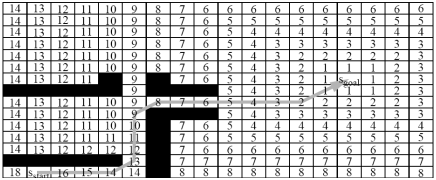
Outline

Unknown Dynamic Graphs

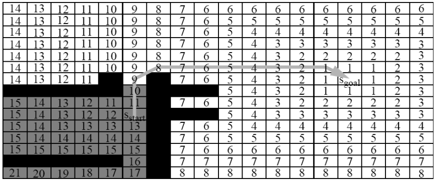
- So far, we assumed that all edge costs are known and do not change
- In practice, the environment may be partially known or changing
- **Naive idea**: recompute the path any time an edge cost changes
- **Lifelong Planning A*** (LPA*):
  - Assumes edge costs change but the agent has not actually moved yet
  - Recomputes the path from start to goal while reusing prior information
- **D* and D* Lite**:
  - The agent starts moving along the path to goal and updates edge costs as the sensors observe new obstacles or free areas
  - Recomputes the path from the current node to the goal while reusing prior information
- 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**: $g_j = min_{i \in Parents(j)} v_i + c_{ij}$
- **Incremental search approach**:
  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 (over)consistent 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., $key_i > key_j$

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

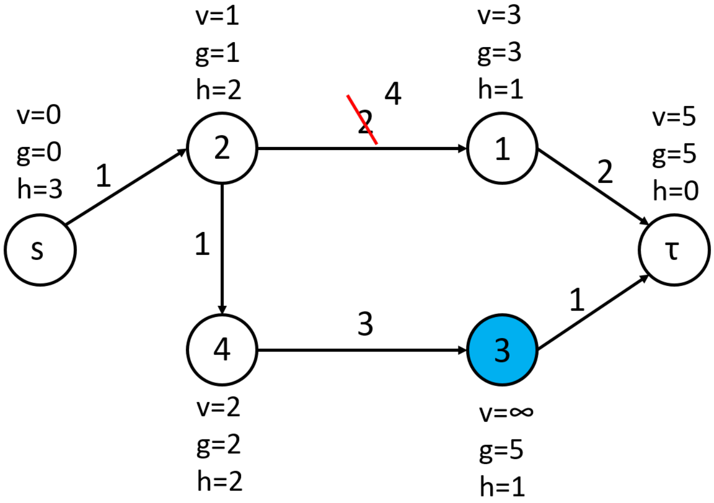
Lifelong Planning A*

Algorithm LPA* ComputePath()

```
1:  function ComputePath()
2:      while $key_\tau > \min_{j \in OPEN} key_j$ or $v_\tau < g_\tau$ do
3:          Remove $i$ with smallest $key_i$ from OPEN
4:          if $v_i > g_i$ (overconsistent) then
5:              $v_i = g_i$; Insert $i$ into CLOSED
6:              for $j \in$ Children($i$) do
7:                  if $g_j > (g_i + c_{ij})$ then
8:                      $g_j \leftarrow (g_i + c_{ij})$
9:                      UpdateMembership($j$)
10:         else (underconsistent)
11:             $v_i = \infty$; UpdateMembership($i$)
12:             for $j \in$ Children($i$) and $j \neq s$ do
13:                 $g_j = \min_{k \in Parents(j)} v_k + c_{kj}$
14:                 UpdateMembership($j$)
15:
16: function UpdateMembership($i$)
17:     if $v_i \neq g_i$ then
18:         if $i \notin$ CLOSED then insert/update $i$ in OPEN with $key_i$
19:     else
20:         if $i \in$ OPEN then remove $i$ from OPEN
```

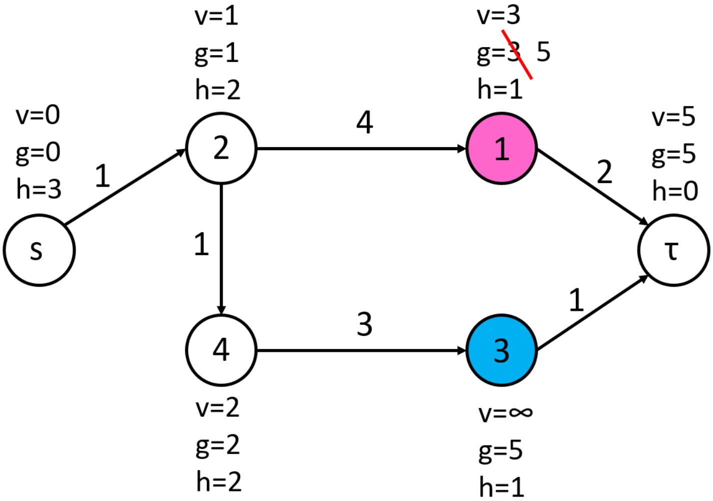
Example: Map Changes and Underconsistent Nodes

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



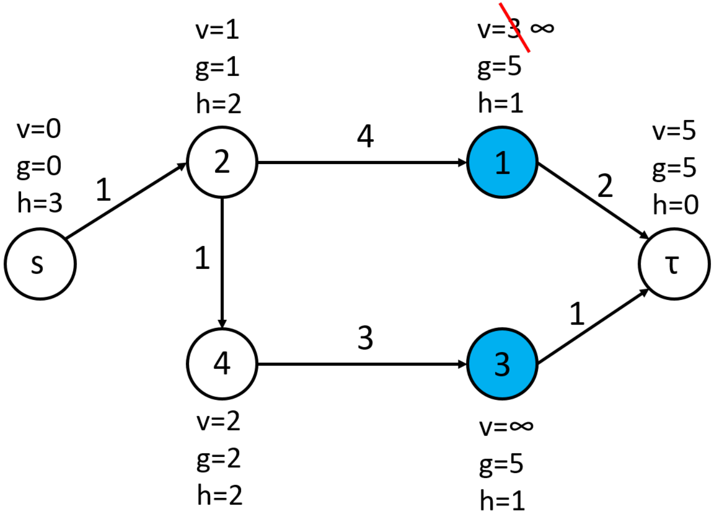
Example: Map Changes and Underconsistent Nodes

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



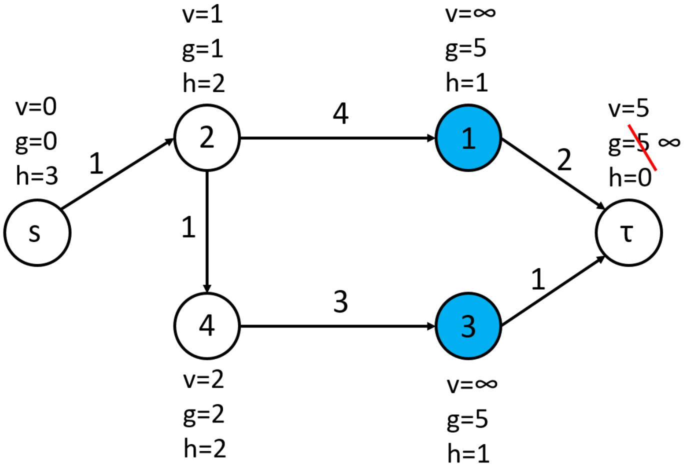
Example: Map Changes and Underconsistent Nodes

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



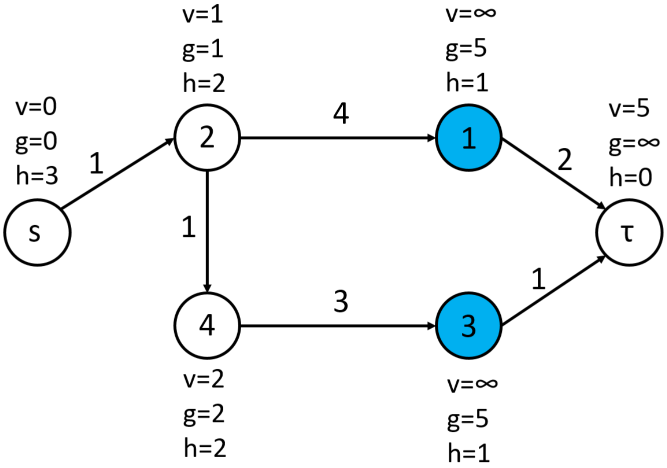
Example: Map Changes and Underconsistent Nodes

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



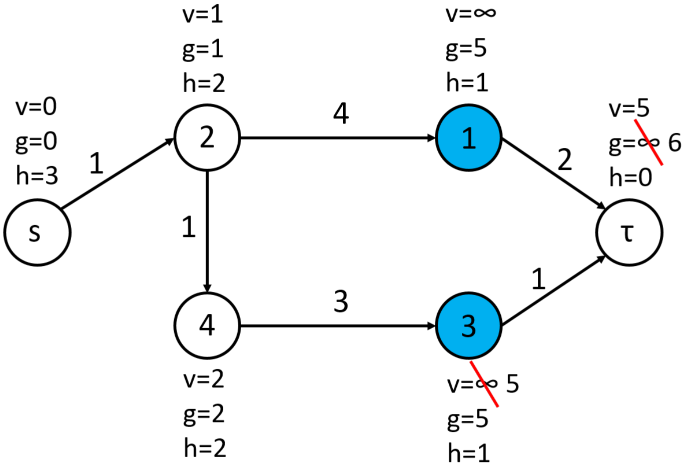
Example: Map Changes and Underconsistent Nodes

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



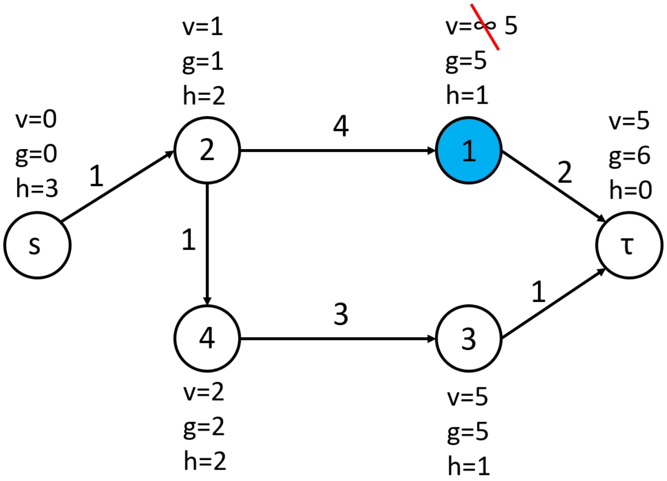
Example: Map Changes and Underconsistent Nodes

- Expand 3 and insert in CLOSED



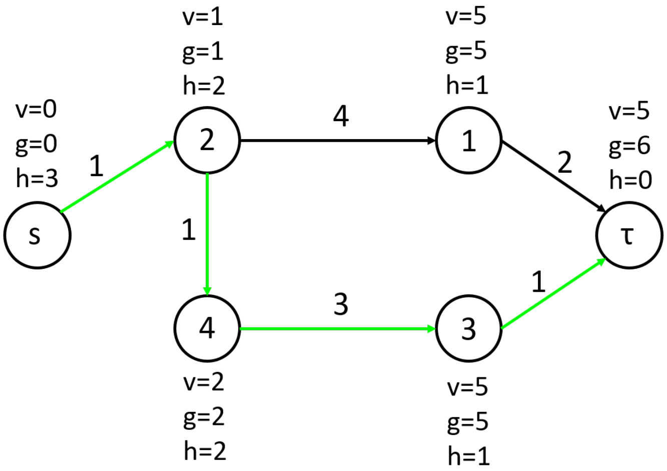
Example: Map Changes and Underconsistent Nodes

- 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** from goal τ to current agent state with all edges reversed
- The root of the search tree remains the same and the g values are more likely to remain unchanged in-between two calls to ComputePath()

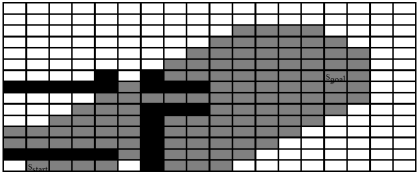
Algorithm D* Lite

```
1: repeat
2:     ComputePath( )                                 ▷ Modified to fix underconsistent nodes
3:     Publish optimal path
4:     Follow the path until the map is updated with new sensor information
5:     Update the corresponding edge costs
6:     Set $\tau$ to the current state of the agent
7: until $\tau$ is reached
```

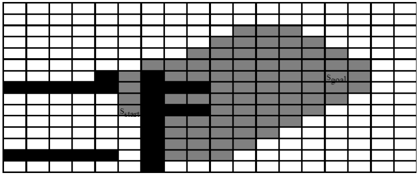
- Details in M. Likhachev, D. Ferguson, G. Gordon, A. Stenz, and S. Thrun, "Anytime search in dynamic graphs," Artificial Intelligence, 2012.

D* Lite (Incremental A*) vs A*

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

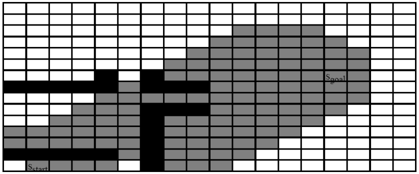


(a) initial search by backward A*

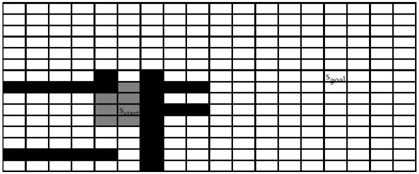


(b) second search by backward A*

- 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 Anytime D*

```
1: Set $\epsilon$ to large value
2: repeat
3:     ComputePath( )                          ▷ Modified to fix underconsistent nodes
4:     Publish $\epsilon$-suboptimal path
5:     Follow the path until the map is updated with new sensor information
6:     Update the corresponding edge costs
7:     Set $\tau$ to the current state of the agent
8:     if significant changes were observed then
9:         Increase $\epsilon$ or replan from scratch
10:    else
11:        Decrease $\epsilon$
12: until $\tau$ is reached
```