Aerial Robotics
Path Planning I

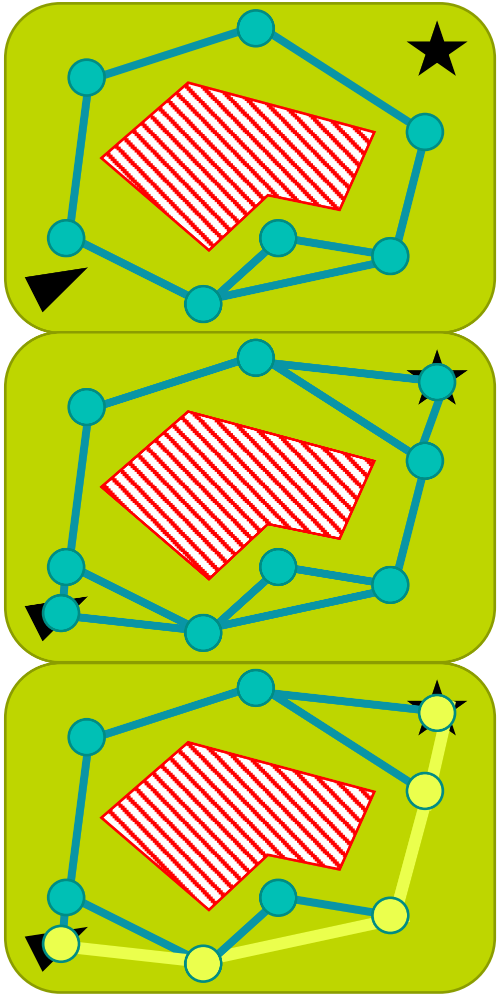
Prof. Arthur Richards

Roadmap approaches

- Identify discrete set of locations and connecting paths – the *map*

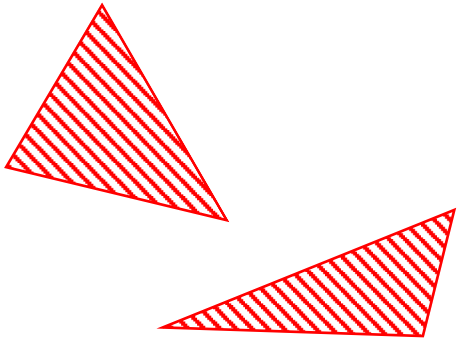
- Connect our start and goal points to that *graph*

- Use *graph search* to find shortest path
  - Will also look at 'visit all' scenario



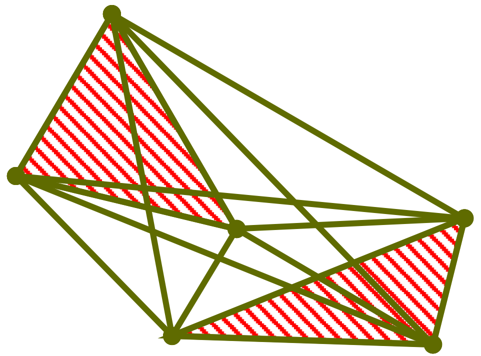
Visibility Graph

- Start with obstacles



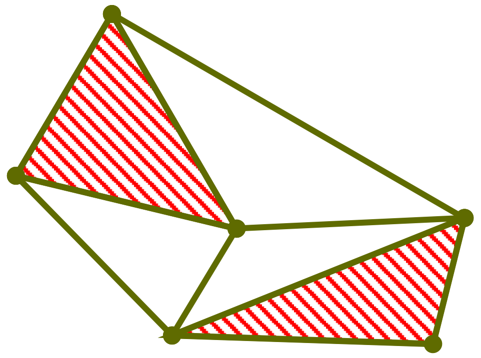
Visibility Graph

- Start with obstacles
- Evaluate all pairs of vertices



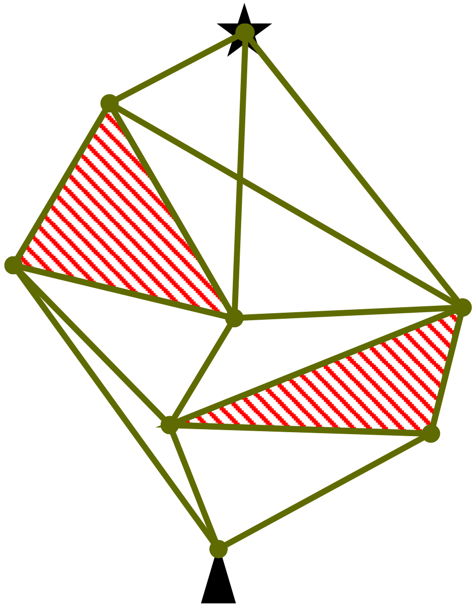
Visibility Graph

- Start with obstacles
- Evaluate all pairs of vertices
- Remove lines that go through obstacles



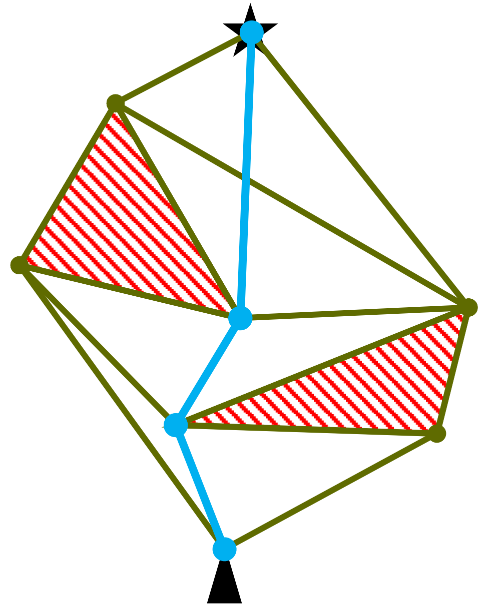
Visibility Graph

- Start with obstacles
- Evaluate all pairs of vertices
- Remove lines that go through obstacles
- Connect the start and goal



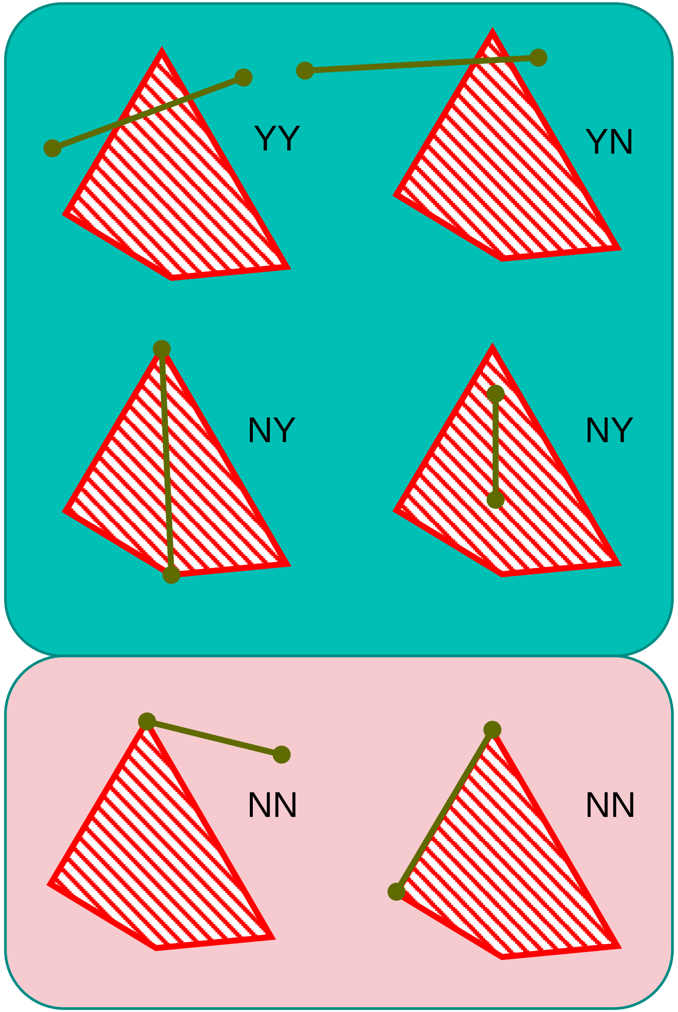
Visibility Graph

- Start with obstacles
- Evaluate all pairs of vertices
- Remove lines that go through obstacles
- Connect the start and goal
- Graph search



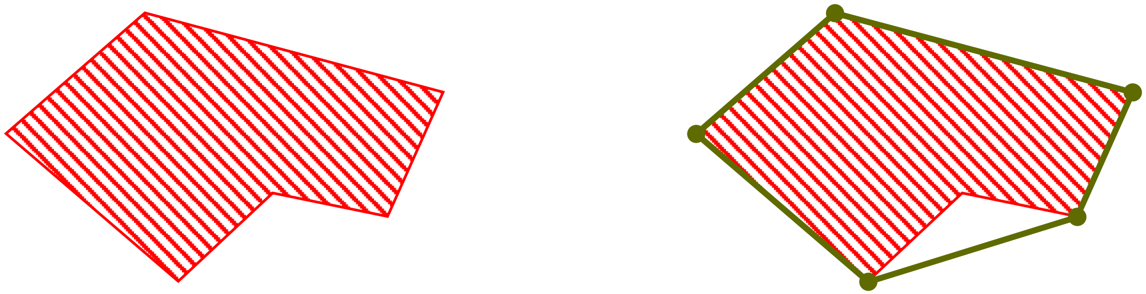
Visibility Testing

- Sounds easy, right?
- Does line A-B go through the obstacle?
  - Does AB cross any edge of the obstacle?
  - Is midpoint of AB strictly inside obstacle?
- Must be fast! Need O(N3) checks for N vertices



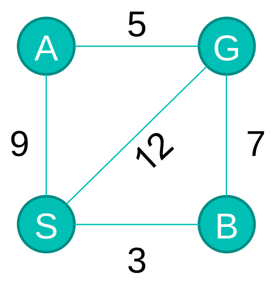
Reduced Visibility Graph

- You can skip concave obstacle corners with no loss of quality



Graph Search: Dijkstra’s Shortest Path

- Other algorithms are available – A*, Floyd-Warshall, etc.
- Lots of code available for free
- Example graph:

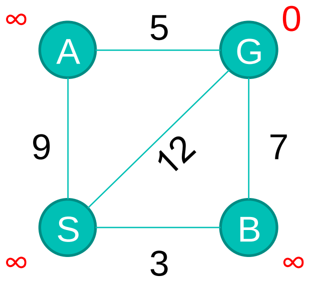


Or in matrix form:

	S	A	B	G
S	0	9	3	12
A	9	0	∞	5
B	3	∞	0	7
G	12	5	7	0

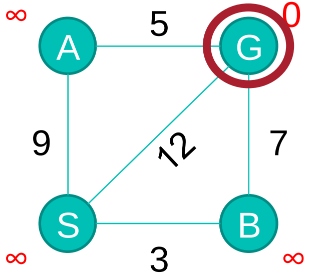
Dijkstra’s Algorithm

- Start: label each node with an upper bound of the cost-to-go, i.e. cost to reach G from that node
  - Zero for G, obviously
  - Infinity everywhere else
    - Lazy but it’s the only bound without doing any more work… be patient!



Dijkstra’s Algorithm

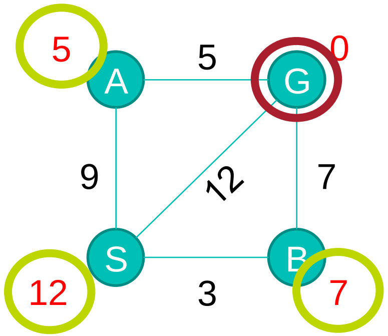
- Pick the node with lowest current cost
- Here, G



Dijkstra's Algorithm

- Using the cost to go and the link cost, update the cost-to-go of each of G's neighbours

- Only update if we're improving the cost-to-go
  – For A: 0+5<∞ so A becomes 5
  – For B: 0+7<∞ so B becomes 7
  – For S: 0+12<∞ so S becomes 12

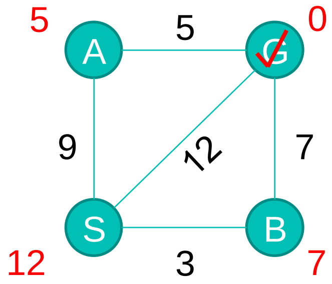


Dijkstra’s Algorithm

- Using the cost to go and the link cost, update the cost-to-go of each of G’s neighbours

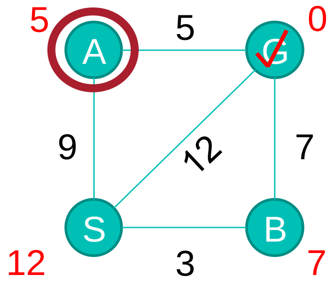
- Only update if we’re improving the cost-to-go
  - For A: 0+5<∞ so A becomes 5
  - For B: 0+7<∞ so B becomes 7
  - For S: 0+12<∞ so S becomes 12

- Mark G as ‘visited’ – no returns



Dijkstra's Algorithm

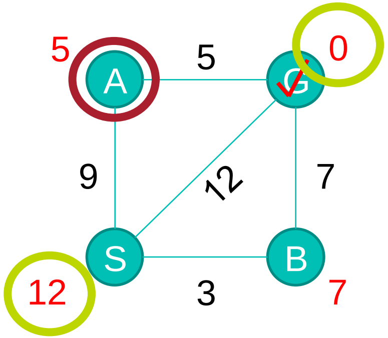
- Now repeat: choose unvisited node with lowest cost - A



Dijkstra's Algorithm

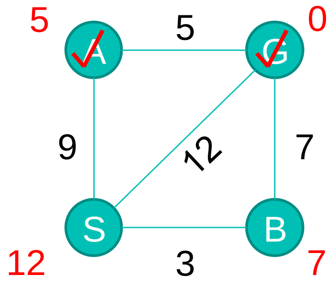
- Now repeat: choose unvisited node with lowest cost – A

- Update its neighbours if it offers improvement
  - G: 5+5=10>0 → no update
  - S: 5+9=14>12 → no update



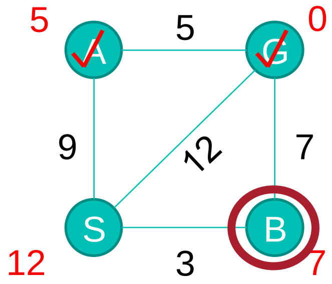
Dijkstra's Algorithm

- Now repeat: choose unvisited node with lowest cost – A
- Update its neighbours if it offers improvement
  - G: 5+5=10>0 → no update
  - S: 5+9=14>12 → no update
- A visited



Dijkstra’s Algorithm

- Repeat again: unvisited node with lowest cost is B

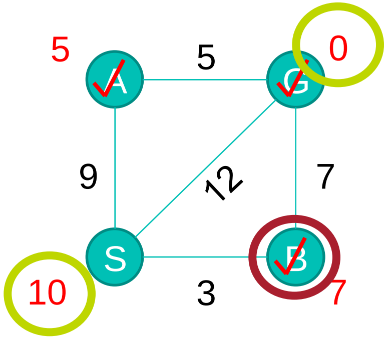


Dijkstra's Algorithm

- Repeat again: unvisited node with lowest cost is B

- Update neighbours
  - G: 7+7=14>0 → no update
  - S: 7+3=10<12 → S becomes 10

- B visited



Dijkstra's Algorithm

- And that's it! When the start node becomes the next node to expand, you can stop
  - Note: you'd get the same answer working forwards from the start to the goal