DroMOOC

Motion Planning

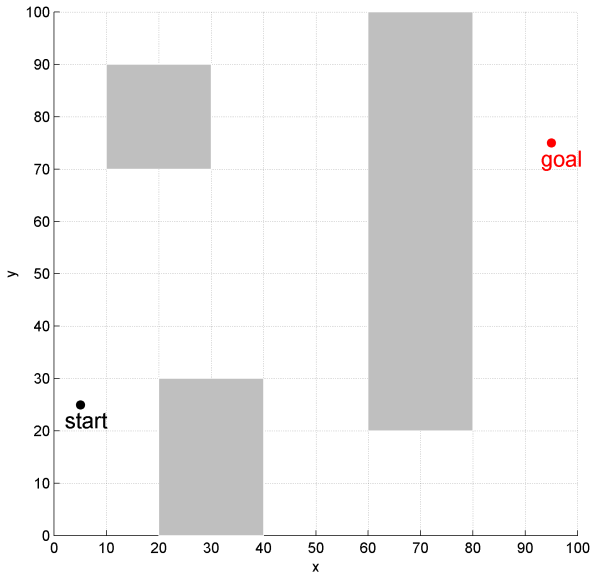
Basic level

Graph-based methods

Julien MARZAT

ONERA

Definitions



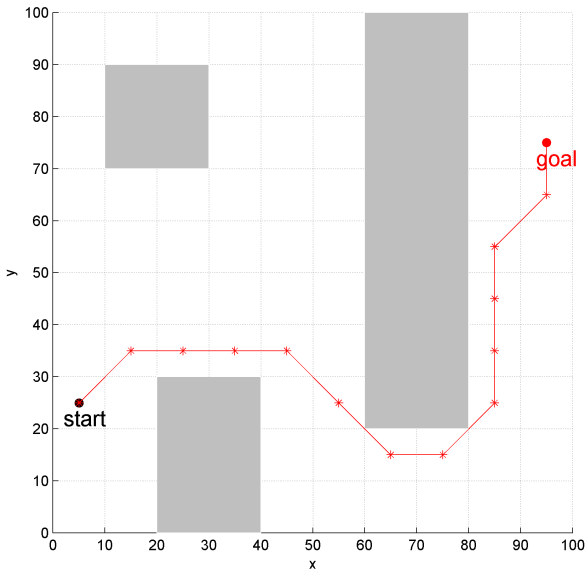
Configuration space

Robot states (position, orientation)
For UGVs and UAVs: $\mathbb{R}^2$, $\mathbb{R}^3$, $SO(3)$

Obstacles given by environment mapping (free / occupied cells)

Move robot from Start to Goal

Definitions



Path

Set of successive configurations $\{q_i\}$

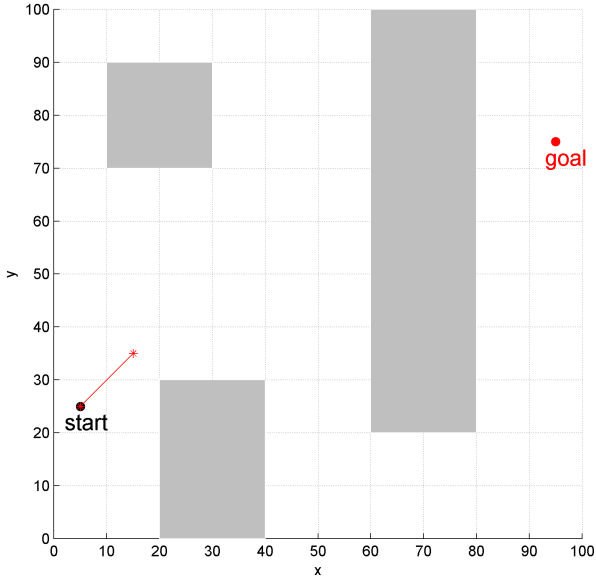
Trajectory

Timely-ordered configurations $\{t_i, q_i\}$

Motion

Trajectory with dynamical constraints $\{t_i, q_i\}$ s.t. $\dot{q}_i = f(q_i, u_i)$

Definitions



Path

Set of successive configurations $\{q_i\}$

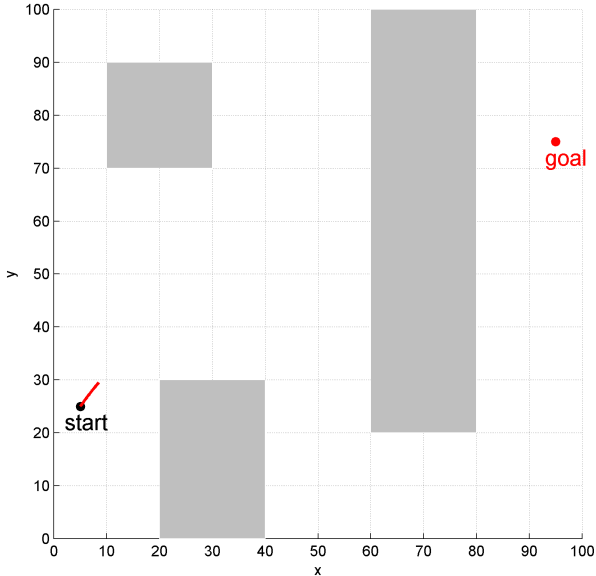
Trajectory

Timely-ordered configurations $\{t_i, q_i\}$

Motion

Trajectory with dynamical constraints $\{t_i, q_i\}$ s.t. $\dot{q}_i = f(q_i, u_i)$

Definitions



Path

Set of successive configurations $\{q_i\}$

Trajectory

Timely-ordered configurations $\{t_i, q_i\}$

Motion

Trajectory with dynamical constraints $\{t_i, q_i\}$ s.t. $\dot{q}_i = f(q_i, u_i)$

Graph-based methods

Graph model

Configuration space discretized as a grid
Build a graph from the grid

- Nodes (vertices) are grid points
- Parameterized number of neighbors
- Edges connect the nodes if feasible (collision check)

Path finding

Graph search to determine shortest path

- Uninformed search: Dijkstra
- Informed search: A*

Graph-based methods

Graph model

Configuration space discretized as a grid
Build a graph from the grid

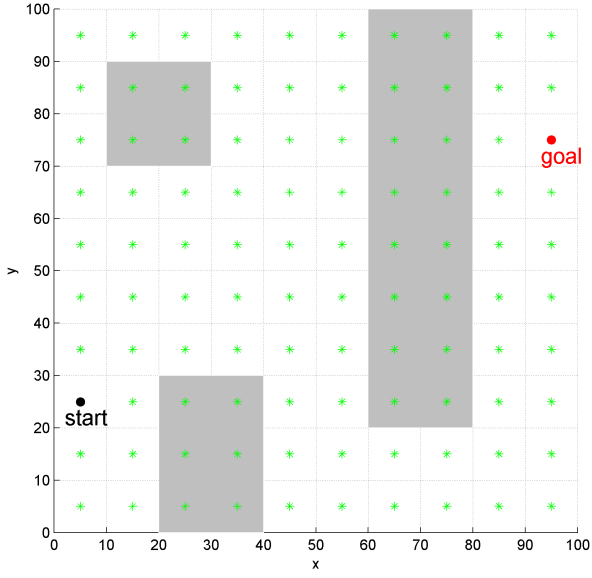
- Nodes (vertices) are grid points
- Parameterized number of neighbors
- Edges connect the nodes if feasible (collision check)

Path finding

Graph search to determine shortest path

- Uninformed search: Dijkstra
- Informed search: A*

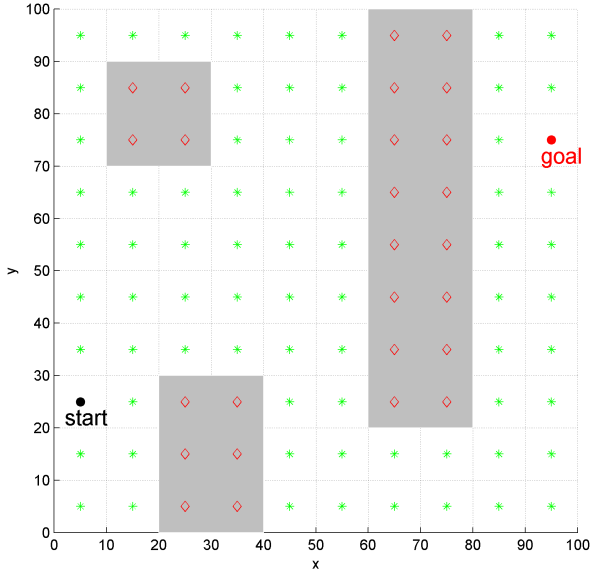
Graph construction



Graph generation

Space discretization

Graph construction

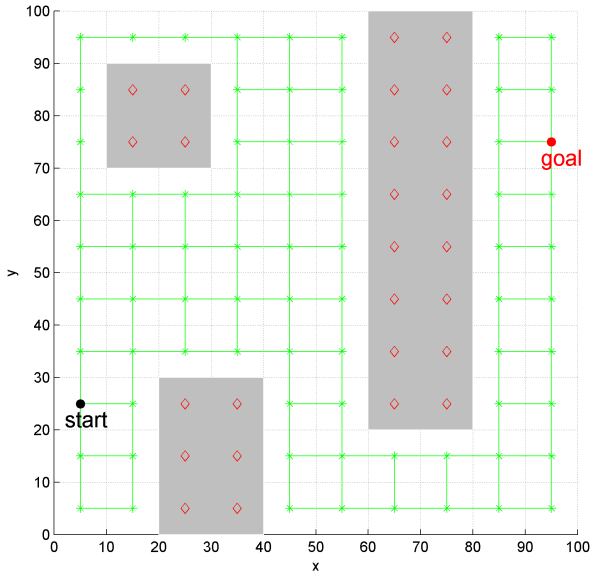


Graph generation

Space discretization

Collision checking

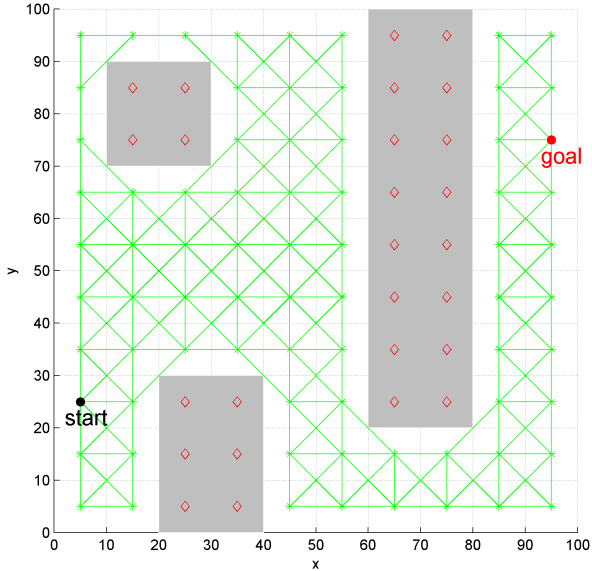
Graph construction



Graph generation

Space discretization
Collision checking
Neighbor definition: 4 neighbors

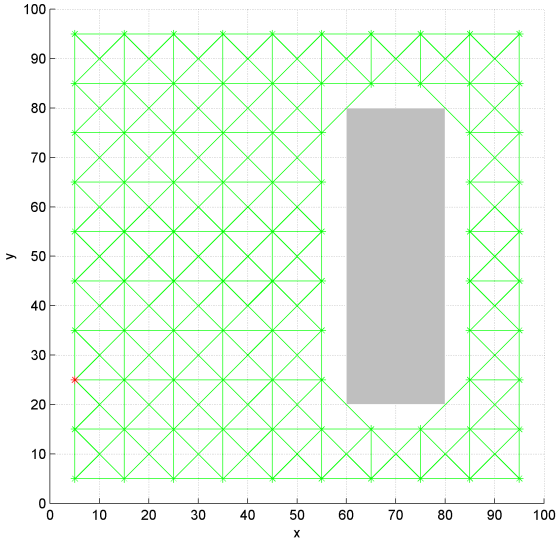
Graph construction



Graph generation

Space discretization
Collision checking
Neighbor definition: 8 neighbors

Dijkstra's algorithm

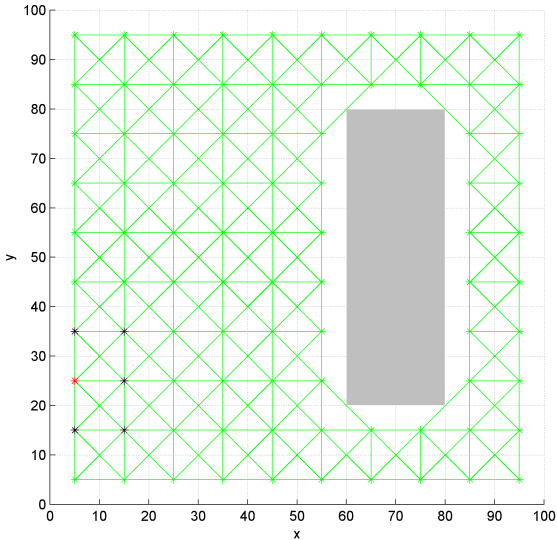


Algorithm

Initialization from starting node:

- Distance value 0, all others at ∞
- All nodes are marked as unvisited

Dijkstra's algorithm

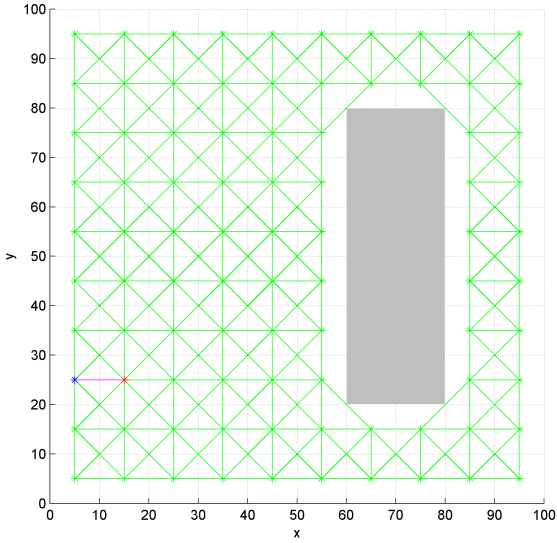


Algorithm

For all unvisited neighbors of current

- Calculate distance from start using path to current node
- Store preceding node on the path
- Node is marked visited when all its neighbors have been updated

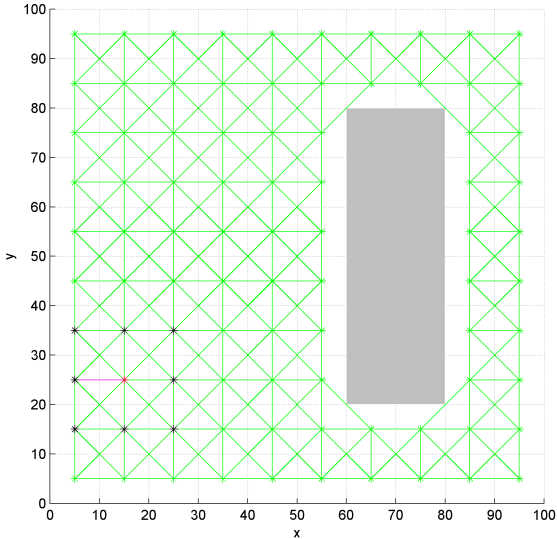
Dijkstra's algorithm



Algorithm

Set unvisited node with lowest distance as current node

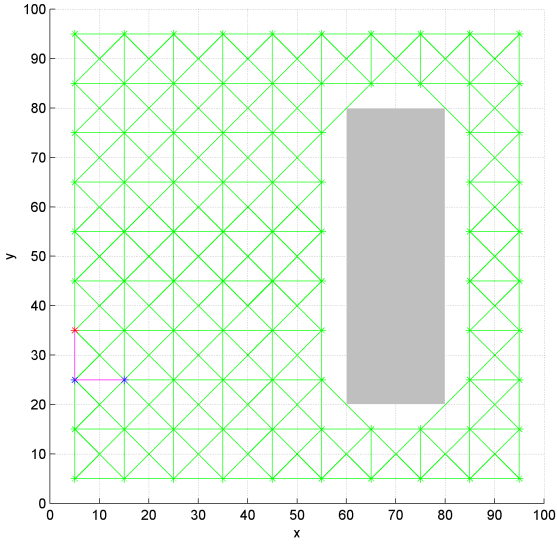
Dijkstra's algorithm



Algorithm

Update the distances of its neighbors,
Mark it as visited

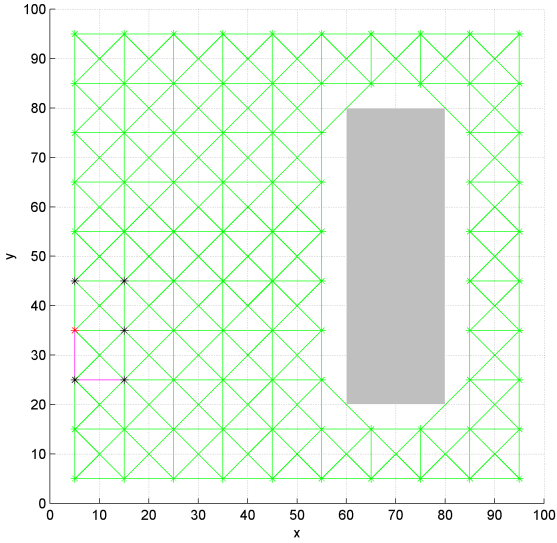
Dijkstra's algorithm



Algorithm

Repeat:
Pick unvisited node with lowest distance

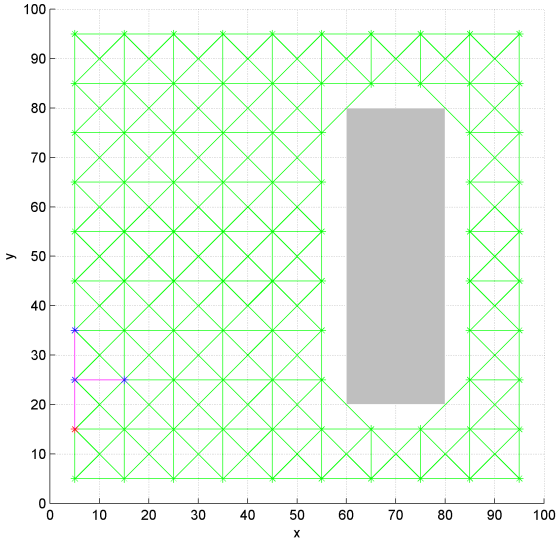
Dijkstra's algorithm



Algorithm

Repeat:
Pick unvisited node with lowest distance
Update its neighbors, mark it as visited

Dijkstra's algorithm



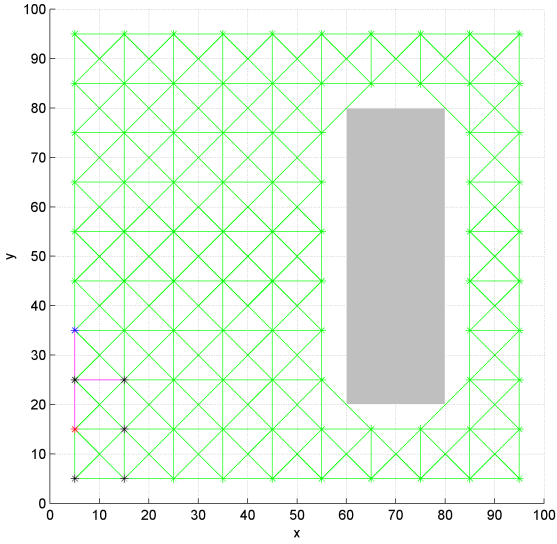
Algorithm

Repeat:

Pick unvisited node with lowest distance

Update its neighbors, mark it as visited

Dijkstra's algorithm



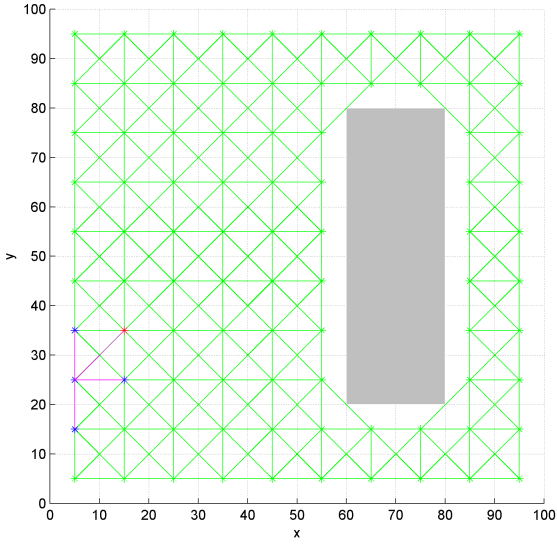
Algorithm

Repeat:

Pick unvisited node with lowest distance

Update its neighbors, mark it as visited

Dijkstra's algorithm



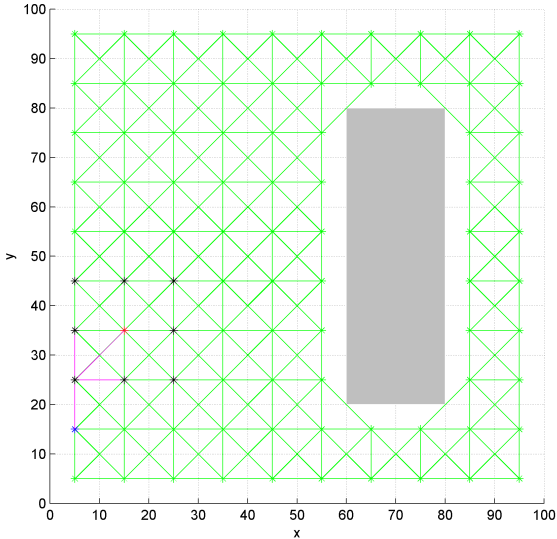
Algorithm

Repeat:

Pick unvisited node with lowest distance

Update its neighbors, mark it as visited

Dijkstra's algorithm



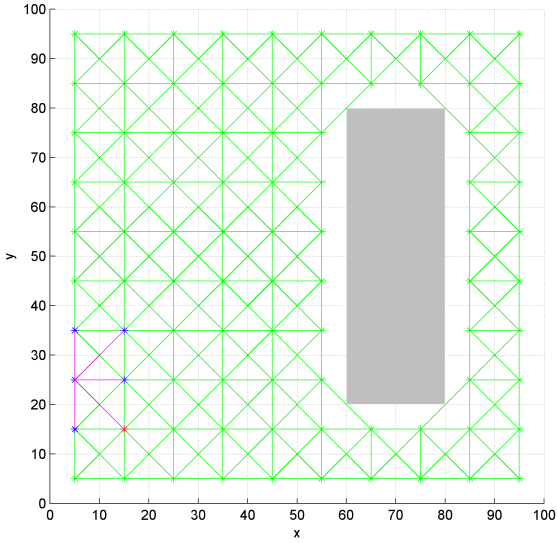
Algorithm

Repeat:

Pick unvisited node with lowest distance

Update its neighbors, mark it as visited

Dijkstra's algorithm



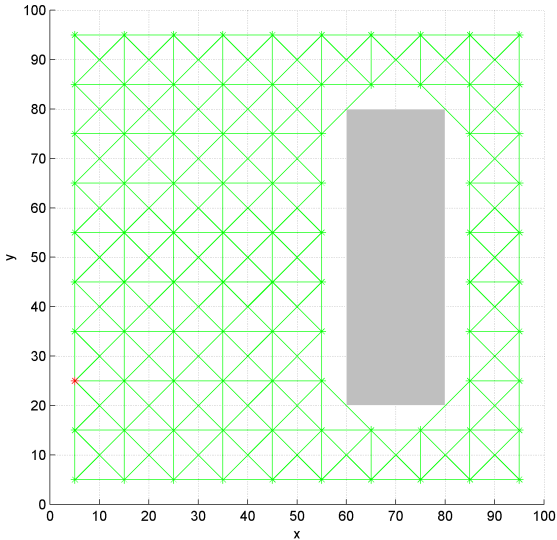
Algorithm

Repeat:

Pick unvisited node with lowest distance

Update its neighbors, mark it as visited

Dijkstra's algorithm



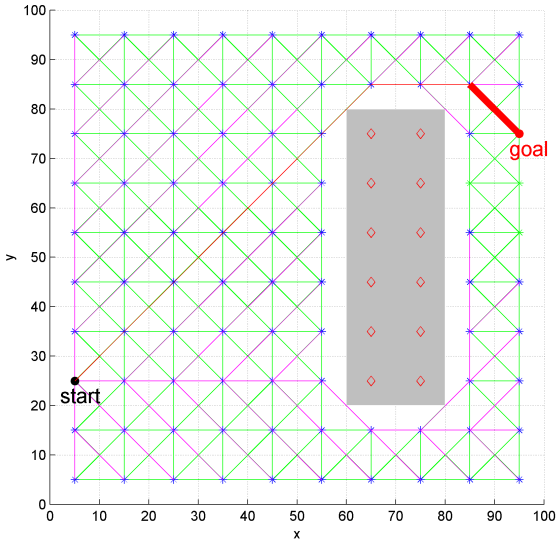
Algorithm

The procedure can be stopped when the goal node is reached

In the end

Shortest path from every node to start
Full exploration can be costly

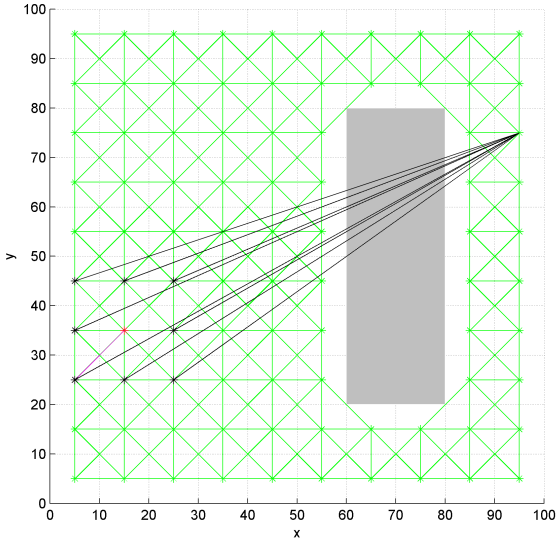
Dijkstra's algorithm



Extract path

Backward from goal node
Iterate through preceding nodes
until start node

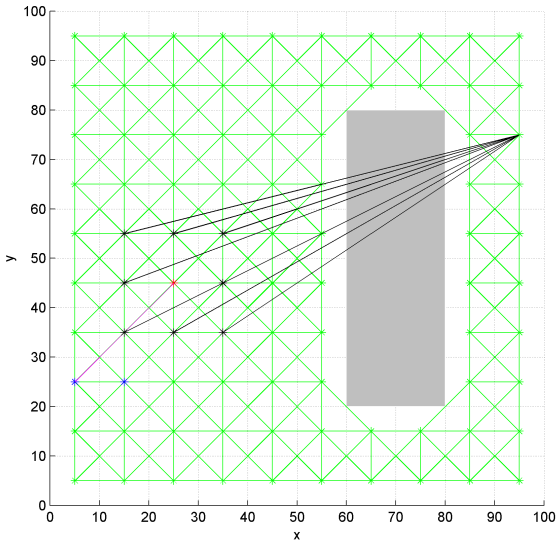
A* algorithm



Algorithm

Distance from start + heuristic to goal (e.g. Euclidean distance)
Optimal path found if heuristic always underestimate real cost.
Back to Dijkstra if heuristic = 0

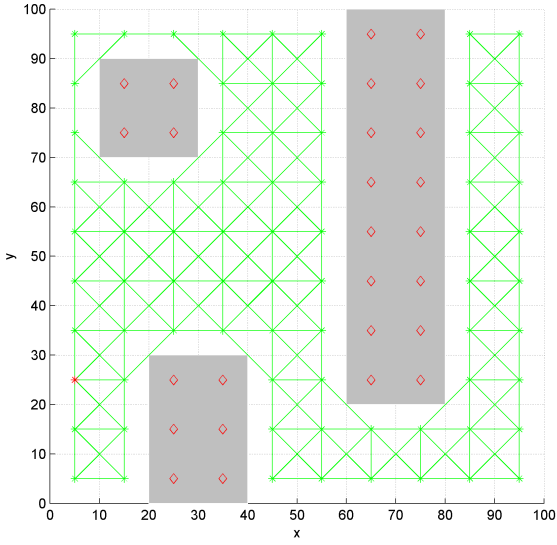
A* algorithm



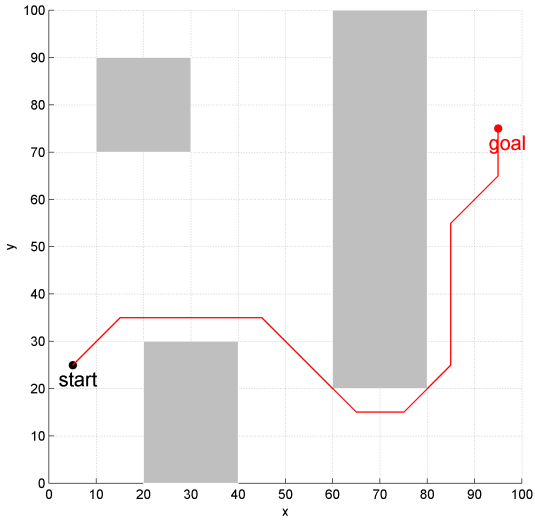
Algorithm

Distance from start + heuristic to goal (e.g. Euclidean distance)
Optimal path found if heuristic always underestimate real cost.
Back to Dijkstra if heuristic = 0

A* algorithm



A few words about smoothing



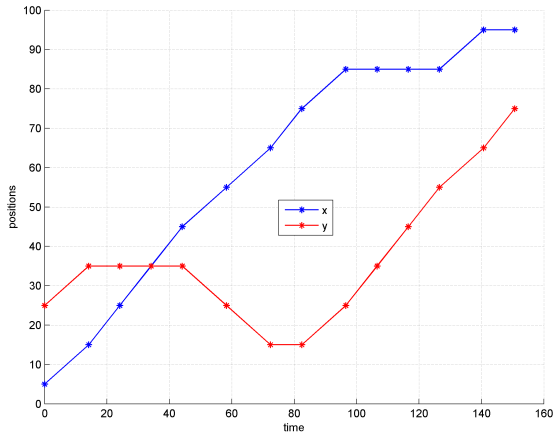
Smoothing procedure

Transform path into a smooth trajectory

Spline: continuous piecewise polynomial function defined between knots

Direct interpolation usually not feasible

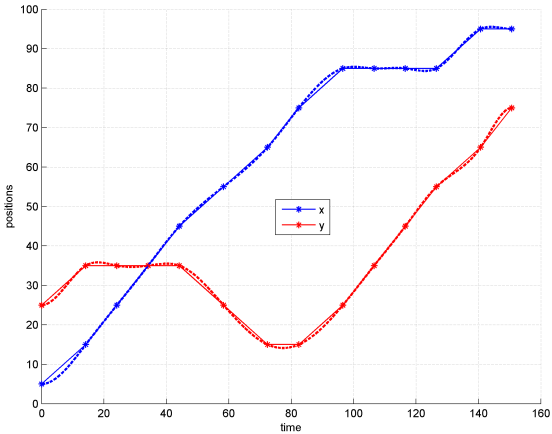
A few words about smoothing



Smoothing procedure

Assign time instants to the successive configurations
(e.g. with constant velocity model)

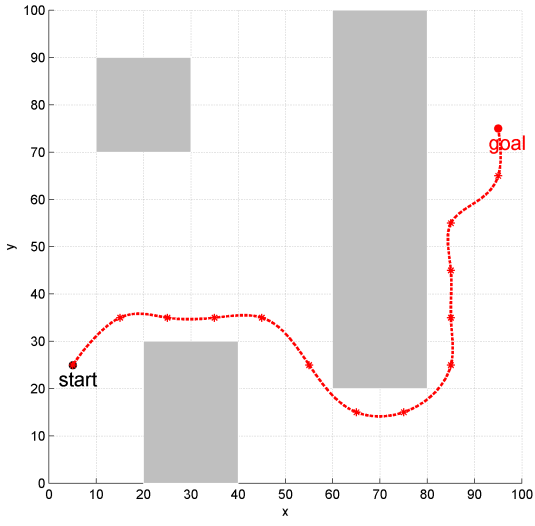
A few words about smoothing



Smoothing procedure

Interpolate a spline for each state, with time as intermediate variable

A few words about smoothing



Smoothing procedure

Go back to the configuration space with smoothed trajectory

Concluding remarks

Deterministic graph search

- Arbitrary discretization of the space, collision check to connect nodes
- Path finding techniques: Dijkstra (uninformed), A* (informed)
- For dynamic online replanning: D*, D* Lite

In any case

- Grid search in high dimensions or large spaces is costly
- A path is obtained, need to generate a trajectory
- Smoothing might be necessary (e.g. using splines)
- Handling dynamical constraints can be hard or costly