7630 – Autonomous Robotics

Introduction to Path and Trajectory Planning

Cédric Pradalier

Today

Introduction

Objectives

- Notions on path planning techniques
- Focus on graph-based planning
- Deterministic and stochastic methods

Recommended Reading

- Planning Algorithms, by Steven M. LaValle, 2006, Cambridge University Pres: available online on `http://planning.cs.uiuc.edu/`
- Robot Motion Planning, Jean-Claude Latombe, Kluwer Academic Publishers, 1991.
- `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html`

Outline

- State-space and obstacle representation
- Global motion planning
  - Reminder about potential fields and optimal control
  - Deterministic graph search (Dijkstra, A*, Lattices)
  - Probabilistic/random/sampling approaches

Outline

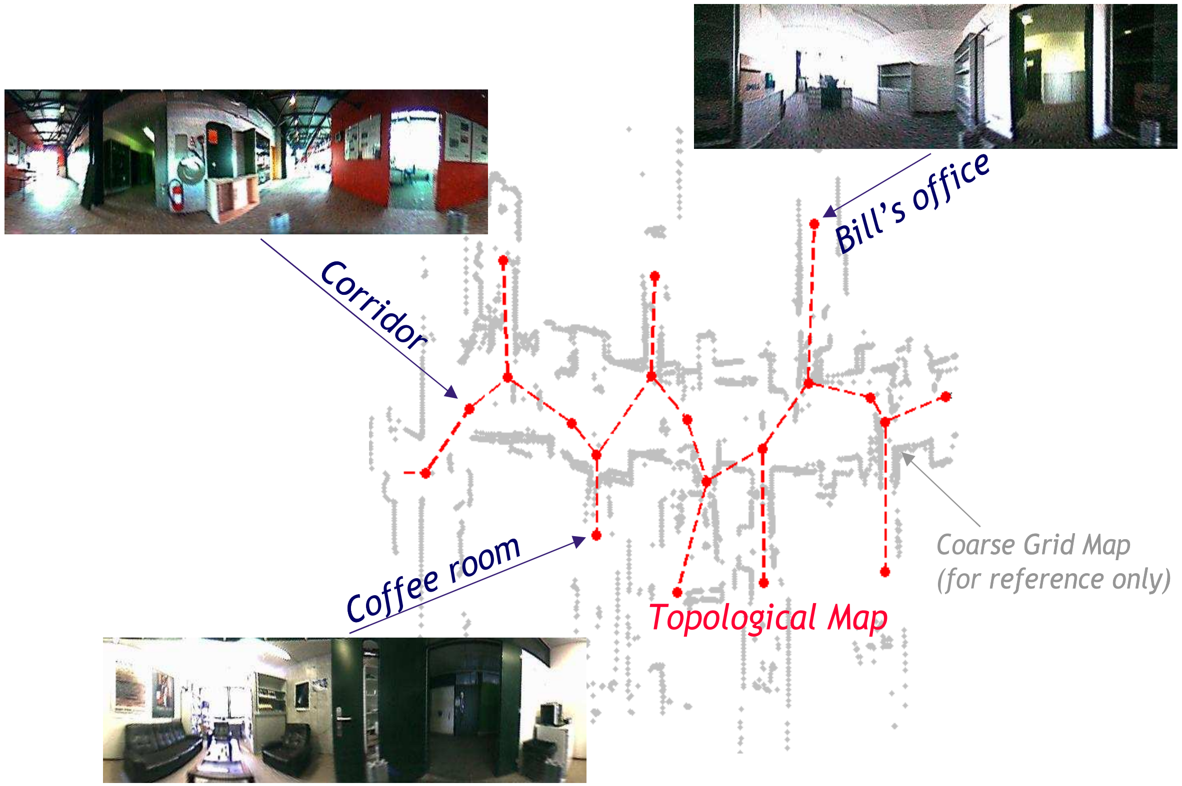
State-space and obstacle representation

Global Motion Planning

Conclusions

The Planning Problem (1/2)

- The problem: find a path in the work space (physical space) from an initial position to a goal position avoiding all collisions with obstacles
- Assumption: there exists a good enough map of the environment for navigation.
  - Topological
  - Metric
  - Hybrid methods



The Planning Problem (2/2)

- We can generally distinguish between
  - (global) path planning and
  - (local) obstacle avoidance.

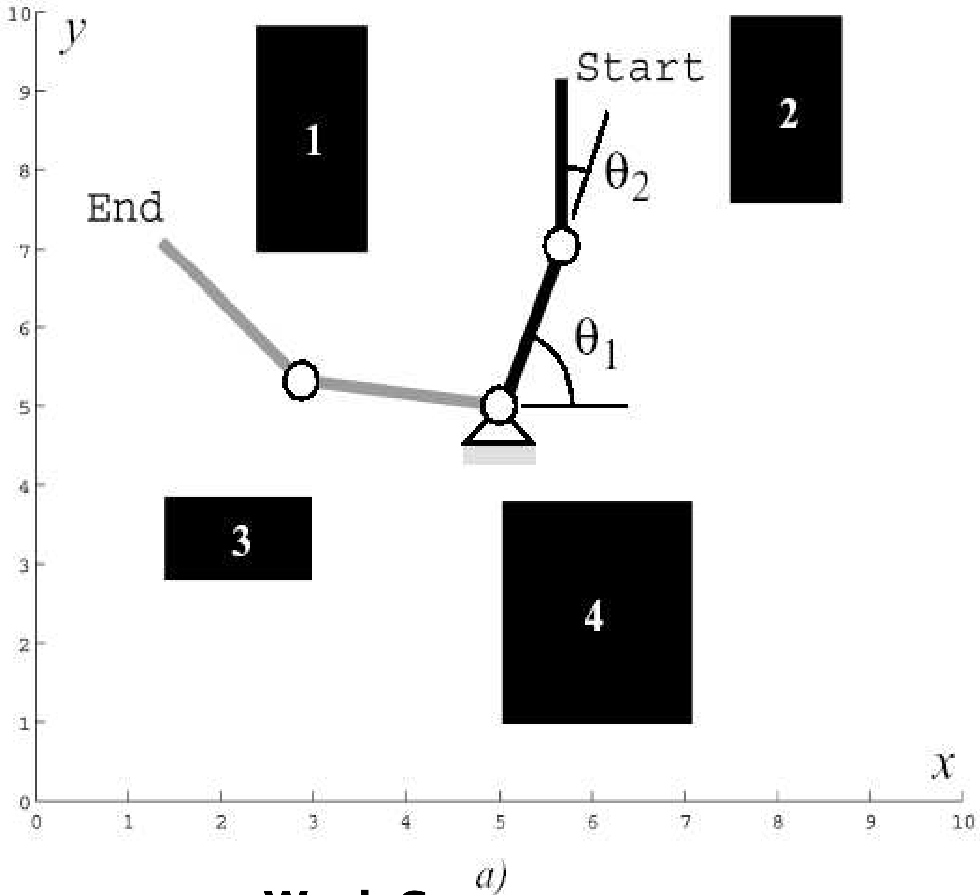
- First step:
  - Transformation of the map into a representation useful for planning
  - This step is planner-dependent

- Second step:
  - Plan a path on the transformed map

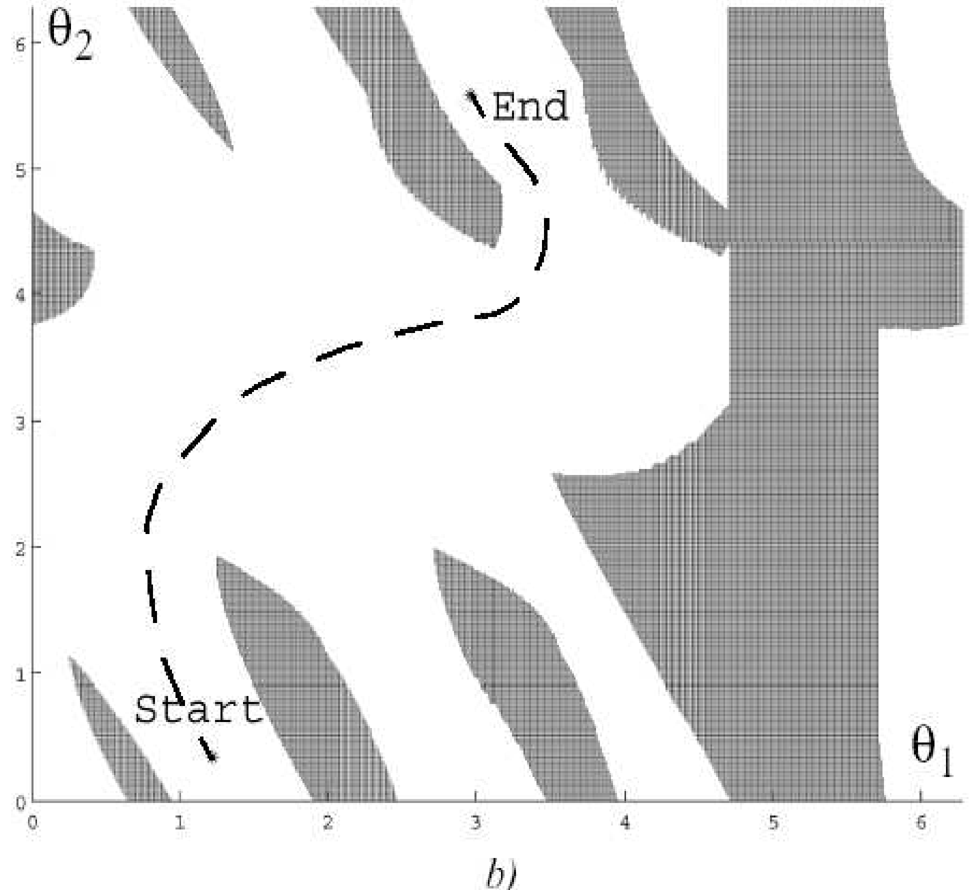
- Third step:
  - Send motion commands to controller
  - This step is planner-dependent (e.g. Model based feed forward, path following)

Work Space (Map) → Configuration Space

- State or configuration q can be described with k values q_i



Work Space



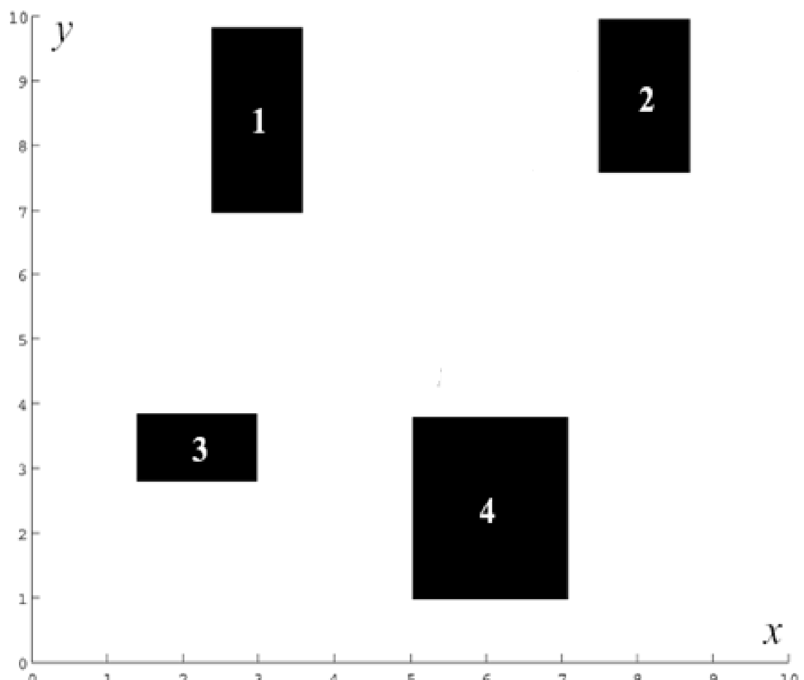
Configuration Space:
the dimension of this
space is equal to the Degrees of Freedom (DoF)
of the robot

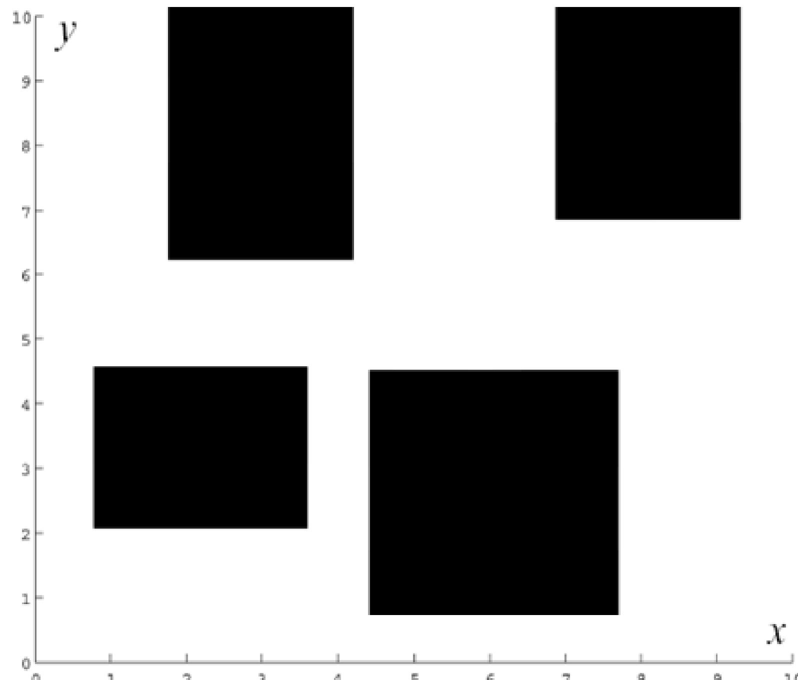
Configuration Space for a Mobile Robot

- Mobile robots operating on a flat ground have 3 DoF: (x, y, θ)

- For simplification, in path planning mobile roboticists often assume that the robot is holonomic and that it is a point. In this way the configuration space is reduced to 2D (x,y)

- Because we have reduced each robot to a point, we have to inflate each obstacle by the size of the robot radius to compensate.



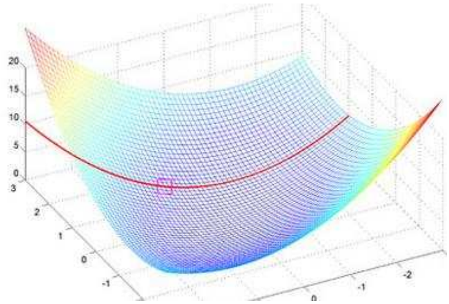


Outline

Path Planning: Overview of Algorithms

1. Optimal Control

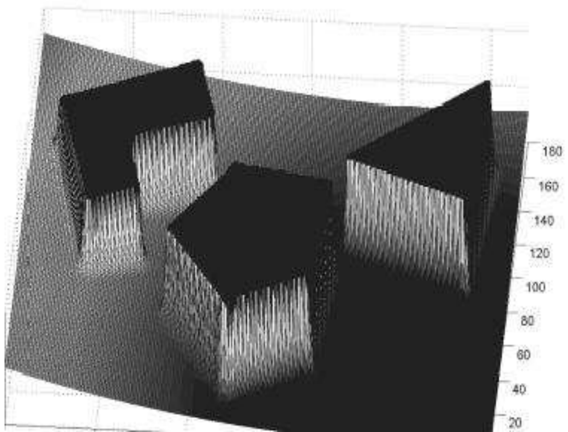
- Solves for the truly optimal solution
- Becomes intractable for even moderately complex and/or nonconvex problems



Source:
http://mitocw.udsm.ac.tz

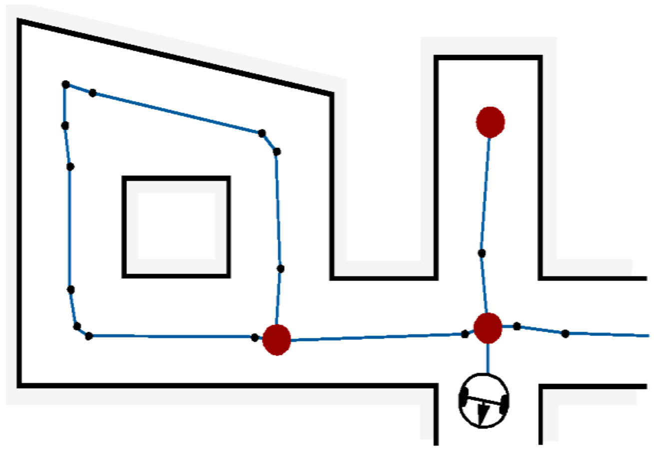
2. Potential Field

- Imposes a mathematical function over the state/configuration space
- Many physical metaphors exist
- Often employed due to its simplicity and similarity to optimal control solutions

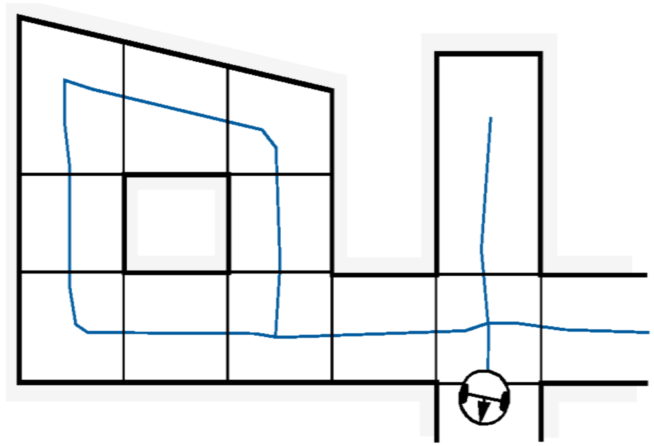


3. Graph Search

- Identify a set of edges and connect them to nodes within the free space

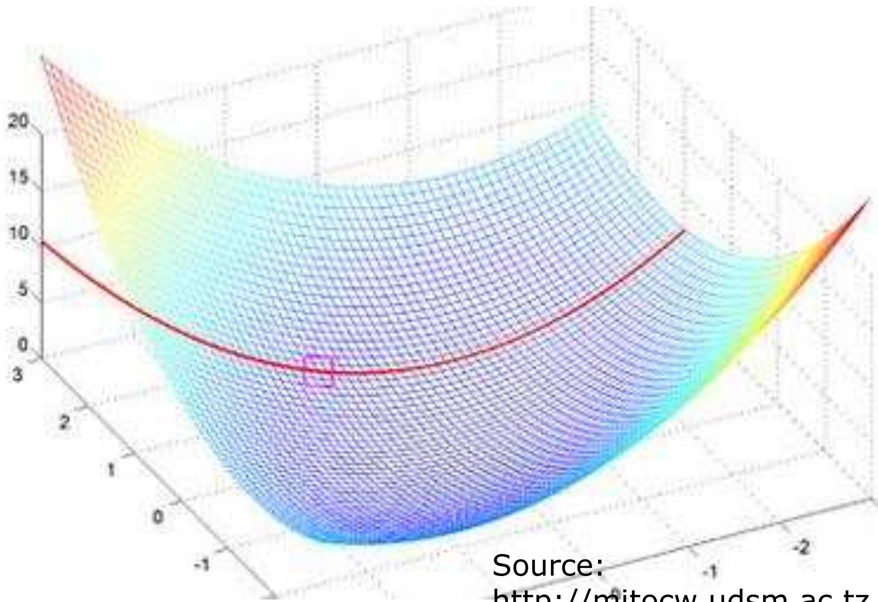


- Where to put the nodes?



Optimal Control based Path Planning Strategies

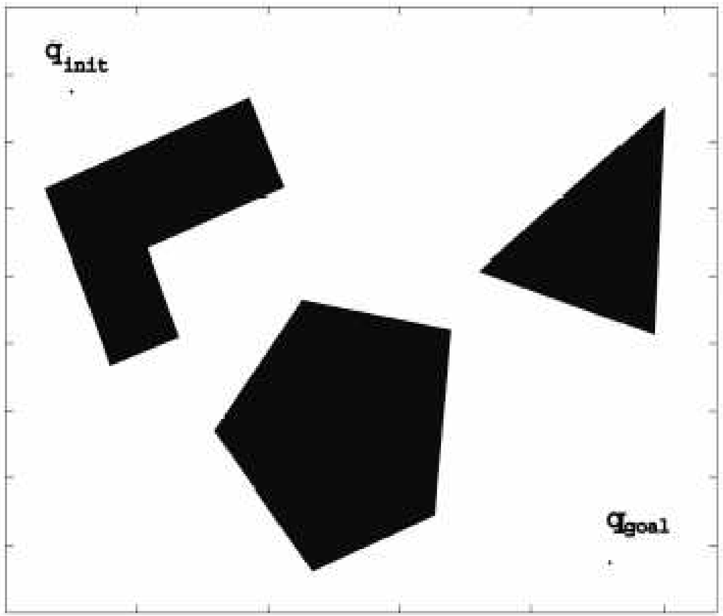
- Overview
  - Solves a two-point boundary problem in the continuum
  - Not treated in this course
- Limitations
  - Becomes very hard to solve as problem dimensionality increases
  - Prone to local optima
- Algorithms
  - Pontryagin maximum principle
  - Hamilton-Jacobi-Bellman



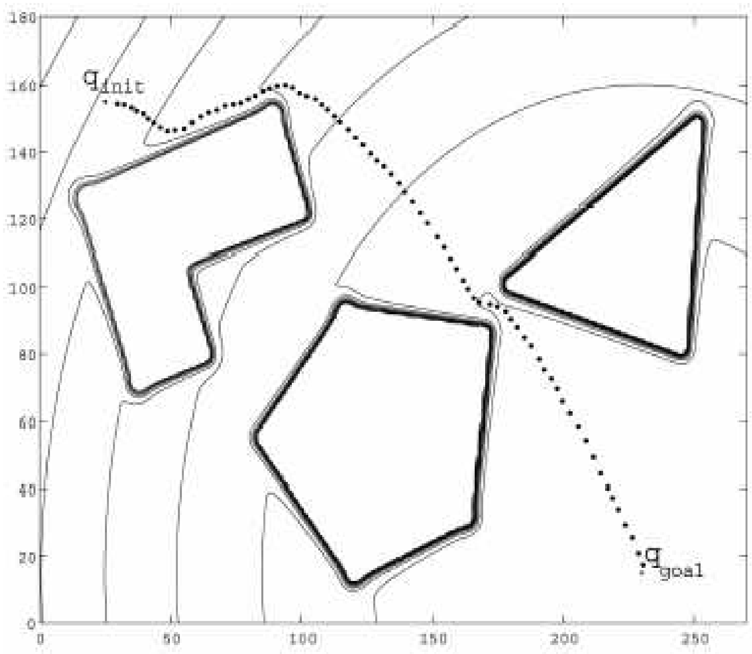
Source:
http://mitocw.udsm.ac.tz

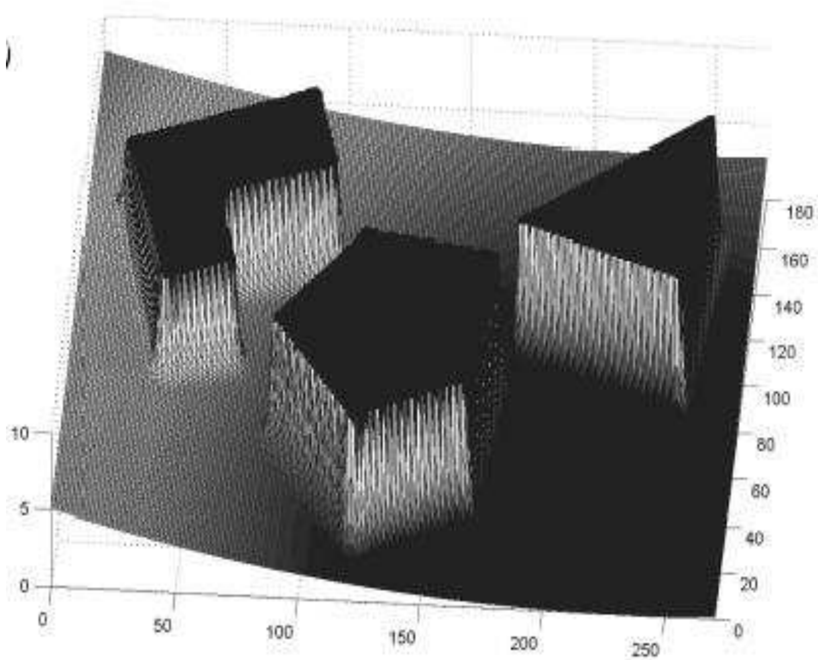
Potential Field Path Planning Strategies

Khatib



- Robot is treated as a *point under the influence* of an artificial potential field.
- Operates in the continuum
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacle are repulsive forces





C Khatib

Potential Field Path Planning: Potential Field Generation

- Generation of potential field function *U(q)*
  - attracting (goal) and repulsing (obstacle) fields
  - summing up the fields
  - functions must be differentiable
- Generate artificial force field *F(q)*

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

- Set robot speed (v_x, v_y) proportional to the force *F*(*q*) generated by the field
  - the force field drives the robot to the goal
  - robot is assumed to be a point mass (non-holonomics are hard to deal with)
  - method produces both a plan *and* the corresponding control

Potential Field Path Planning: Attractive Potential Field

- Parabolic function representing the Euclidean distance $\rho_{goal} = \left\| q - q_{goal} \right\|$ to the goal

$$
\begin{aligned}
U_{att}(q) &= \frac{1}{2} k_{att} \cdot \rho^2_{goal}(q) \\
&= \frac{1}{2} k_{att} \cdot (q - q_{goal})^2
\end{aligned}
$$

- Attracting force converges linearly towards 0 (goal)

$$
\begin{aligned}
F_{att}(q) &= -\nabla U_{att}(q) \\
&= k_{att} \cdot (q - q_{goal})
\end{aligned}
$$

Potential Field Path Planning: Repulsing Potential Field

- Should generate a barrier around all the obstacle
  - strong if close to the obstacle
  - no influence if fare from the obstacle

$$U_{rep}(q) = \begin{cases} \frac{1}{2}k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

  - $\rho(q)$ minimum distance to the object
  - Field is positive or zero and *tends to infinity* as q gets closer to the object

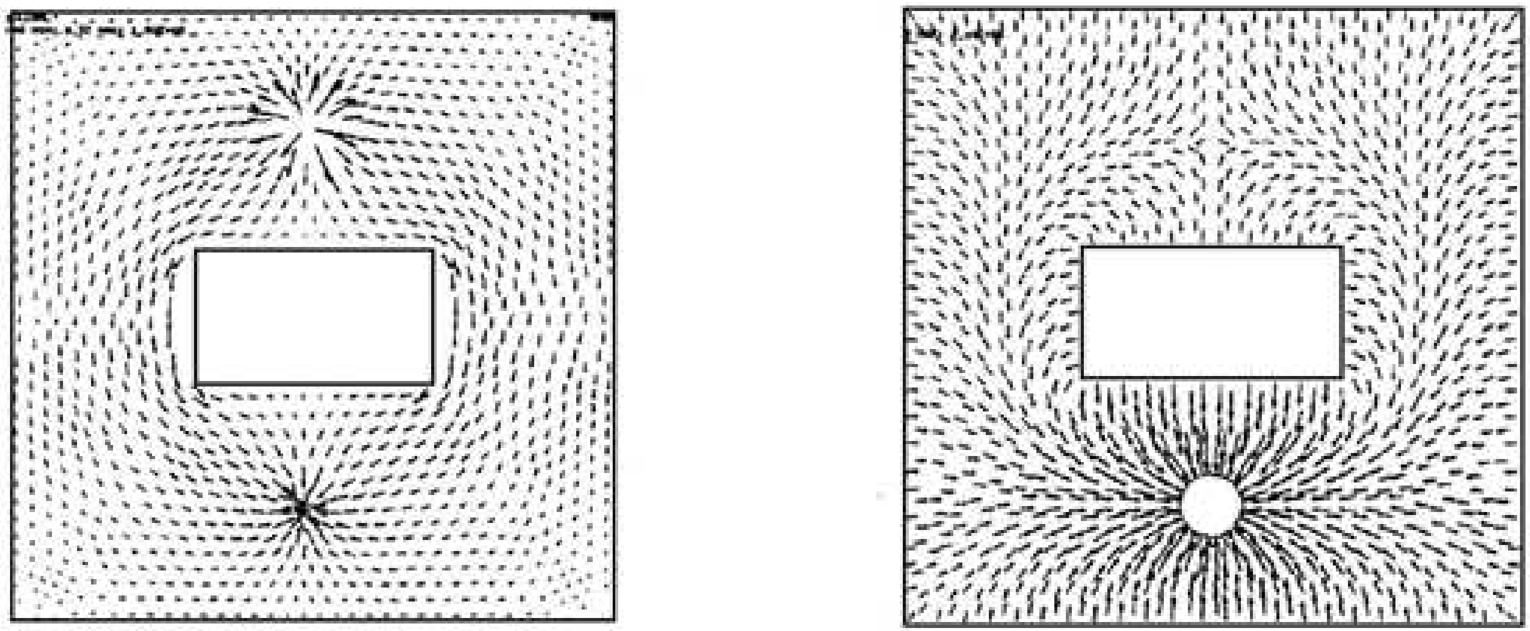
$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2(q)}\frac{q - q_{obst.}}{\rho(q)} & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

Potential Field Path Planning:

- Notes:
  - Local minima problem exists
  - problem is getting more complex if the robot is not considered as a point mass
  - If objects are non-convex there exists situations where several minimal distances exist $\rightarrow$ can result in oscillations

Potential Field Path Planning: Using Harmonic Potentials

- Hydrodynamics / Electrostatics analogy
  - robot is moving similar to a fluid particle following a stream
- Ensures that there are no local minima



Neumann

Dirichlet

C A. Masoud

  - Neumann Boundary Conditions
    - → equipotential lines orthogonal on object boundaries (as in image above!)
    - → short but dangerous paths
  - Dirichlet Boundary Conditions
    - → equipotential lines parallel to object boundaries
    - → long but safe paths

Graph Search

- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation

- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded

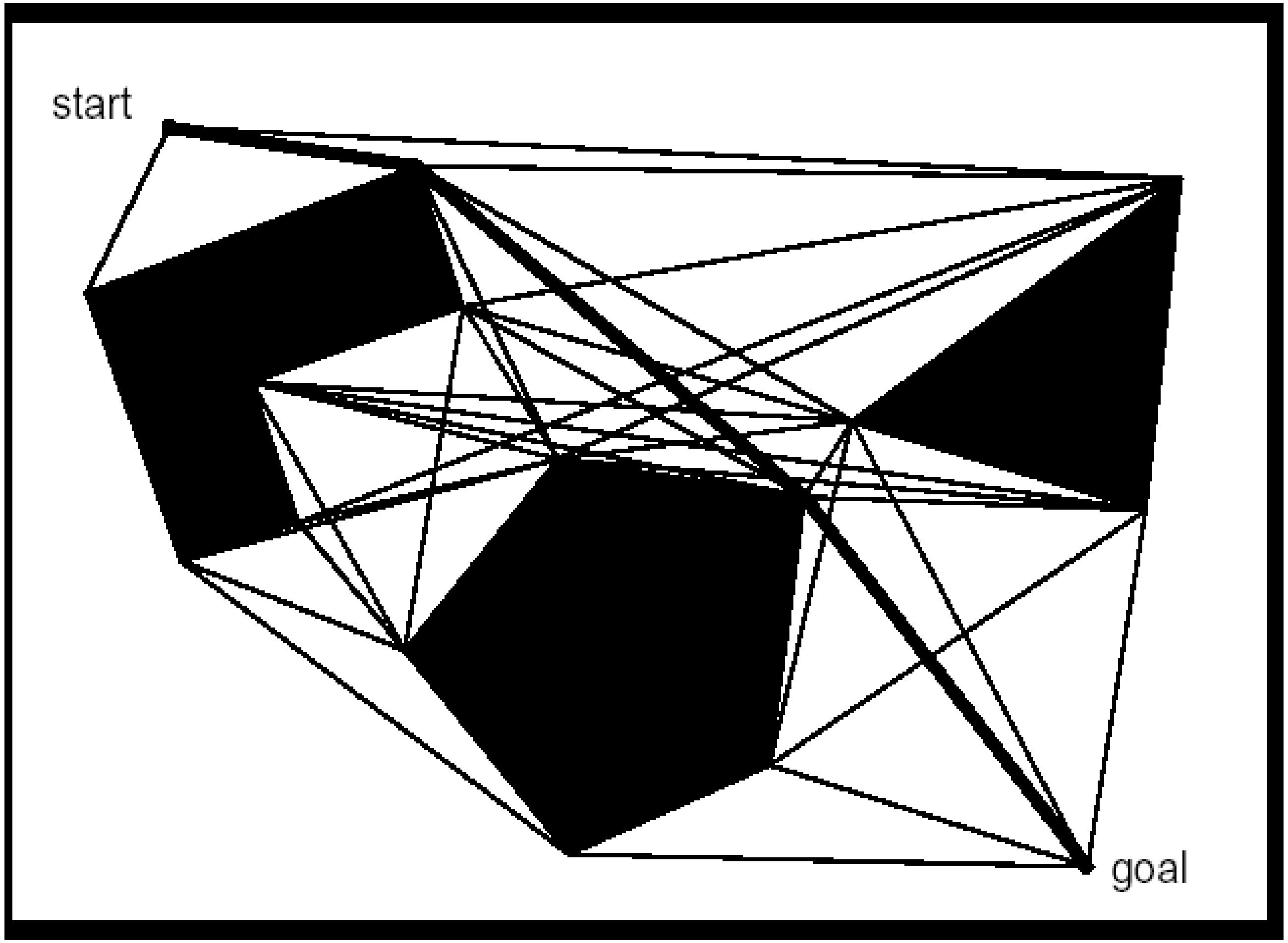
- Algorithms
  - (Preprocessing steps)
  - Breath first
  - Depth first
  - Dijkstra
  - A* and variants
  - D* and variants

C wikipedia.org

Graph Construction (Preprocessing Step)

- Methods
  - Visibility graph
  - Voronoi diagram
  - Cell decomposition
  - ...

Graph Construction: Visibility Graph (1/2)



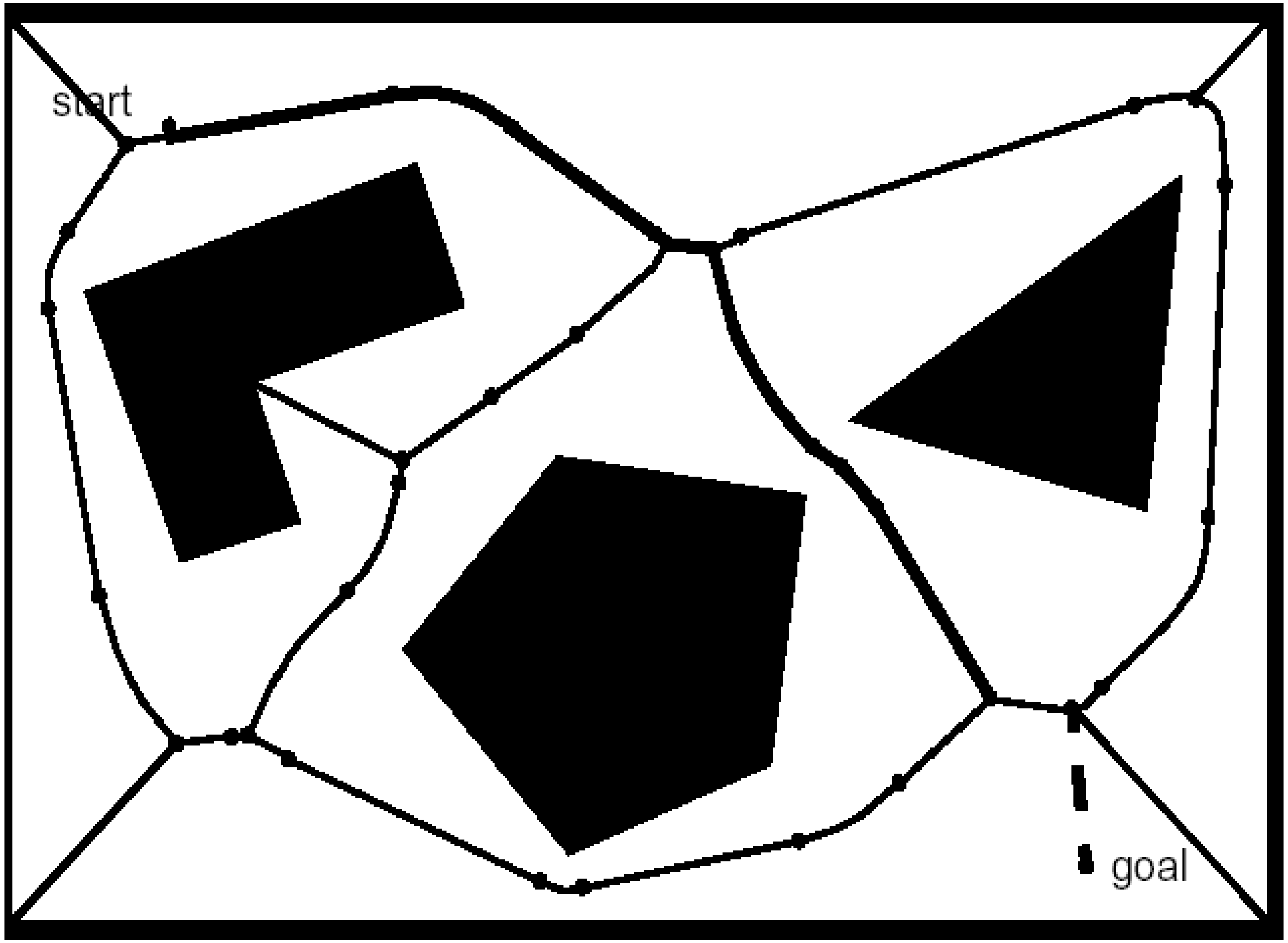
- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

Graph Construction: Visibility Graph (2/2)

- Pros
  - The found path is optimal because it is the shortest length path
  - Implementation simple when obstacles are polygons

- Cons
  - The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than robot's radius
  - Number of edges and nodes increases with the number of polygons
  - Thus it can be inefficient in densely populated environments

Graph Construction: Voronoi Diagram (1/2)



- In contrast to the Visibility Graph approach, the Voronoi Diagram tends to maximize the distance between robot and obstacles
- Easily executable: Maximize the minimal sensor readings
- Works also for map-building: Move on the Voronoi edges: 1D Mapping

Graph Construction: Voronoi Diagram (2/2)

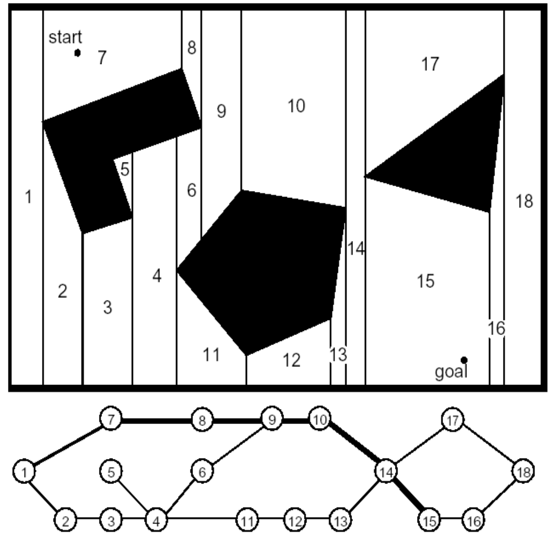
- Pros
  - Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules

- Cons
  - Because the Voronoi diagram tends to keep the robot as far as possible from obstacles, any short range sensor will be in danger of failing

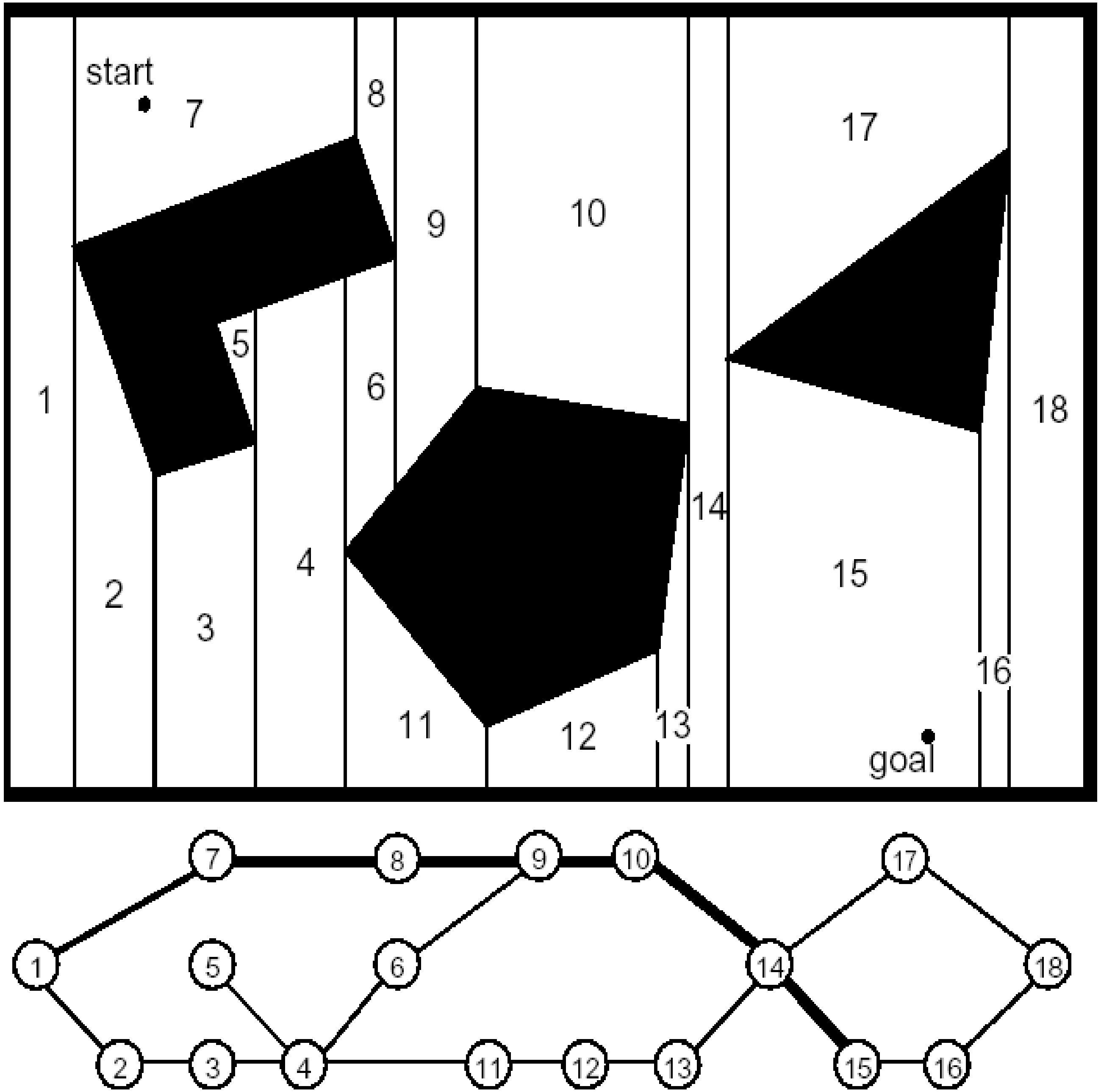
- Peculiarities
  - when obstacles are polygons, the Voronoi map consists of straight and parabolic segments

Graph Construction: Cell Decomposition (1/4)

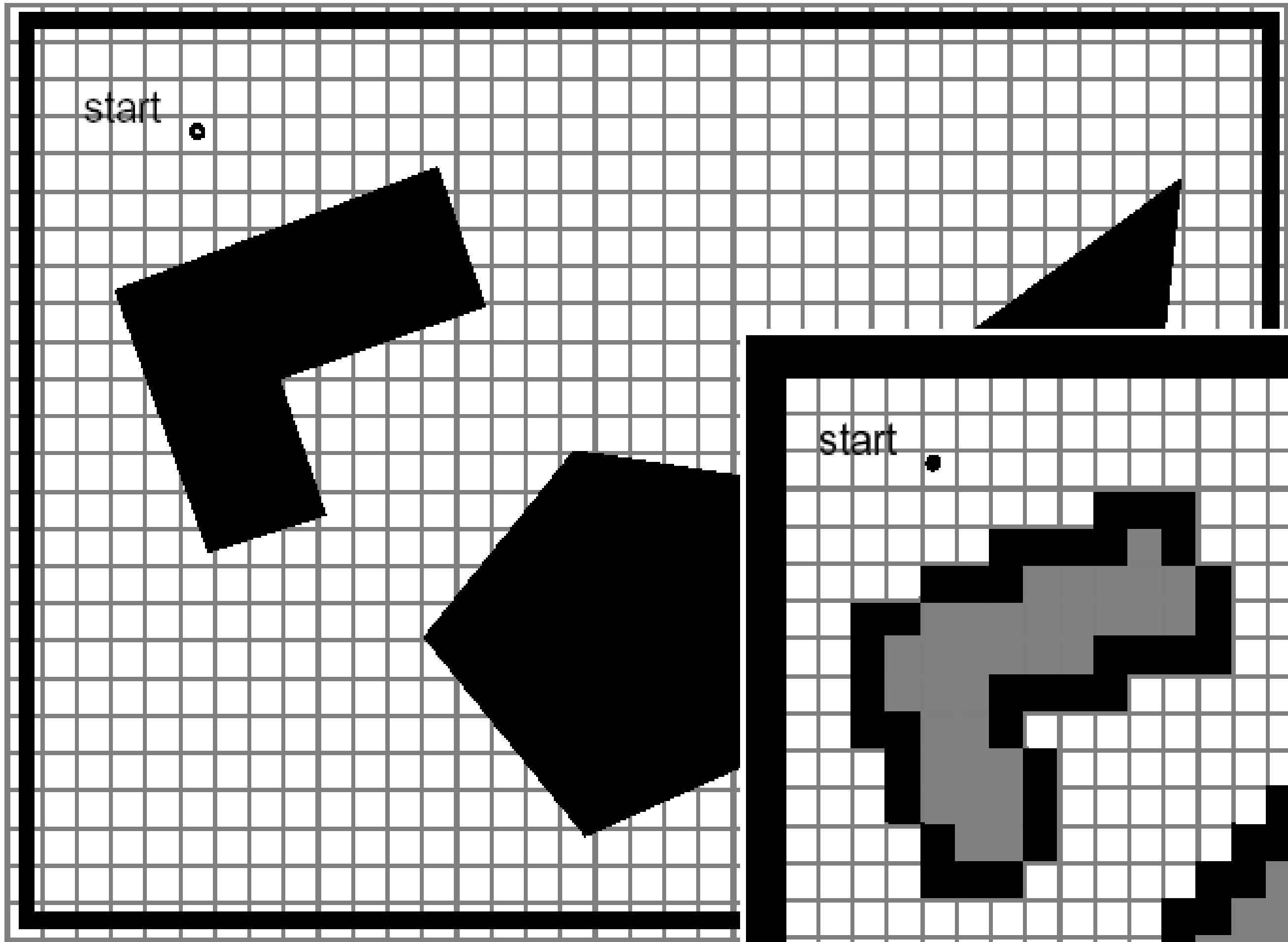
- Divide space into simple, connected regions called cells
- Determine which open cells are adjacent and construct a connectivity graph
- Possible cell decompositions:
  - Exact cell decomposition
  - Approximate cell decomposition:
    - Fixed cell decomposition
    - Adaptive cell decomposition

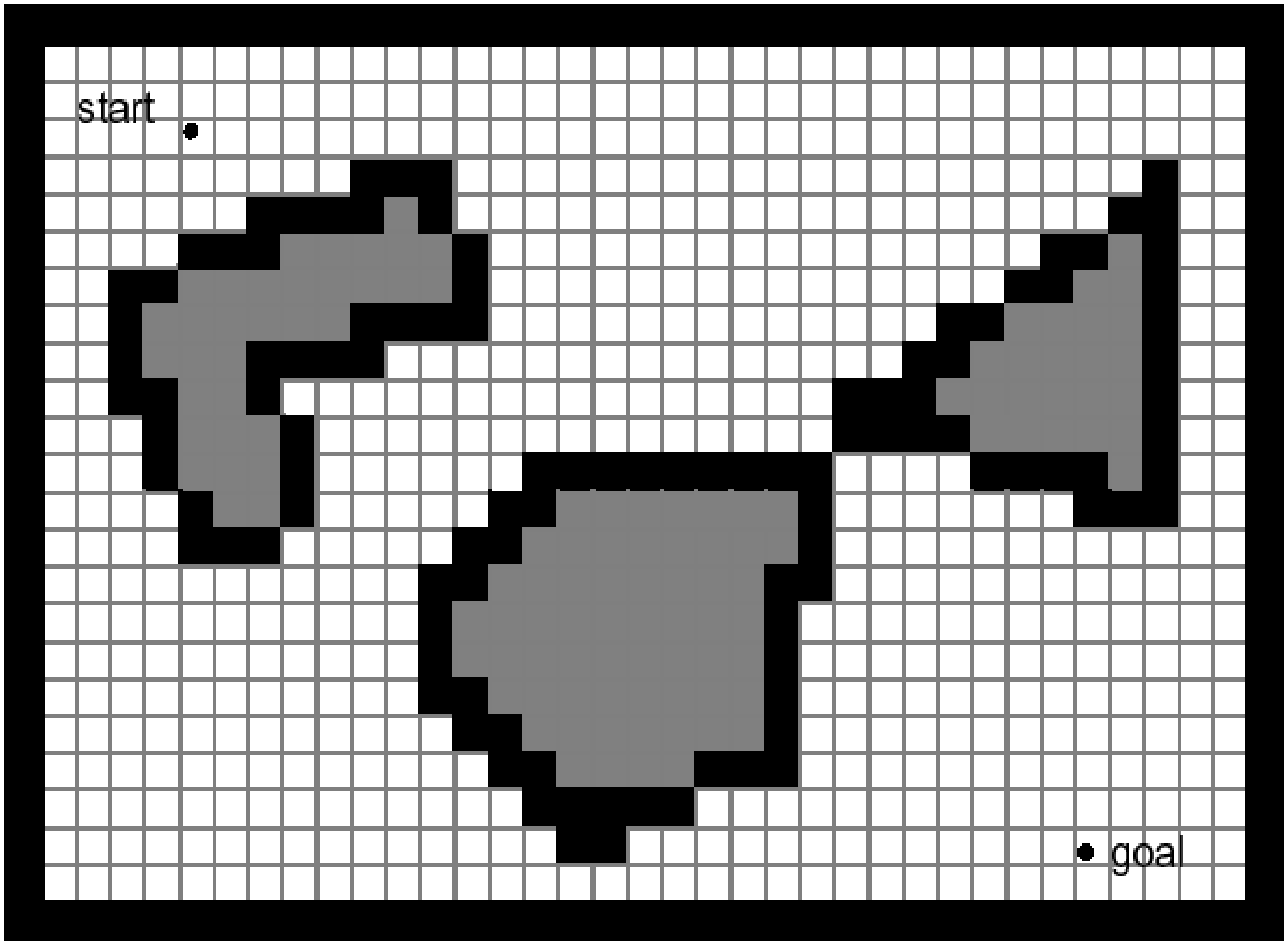


Graph Construction: Exact Cell Decomposition (2/4)

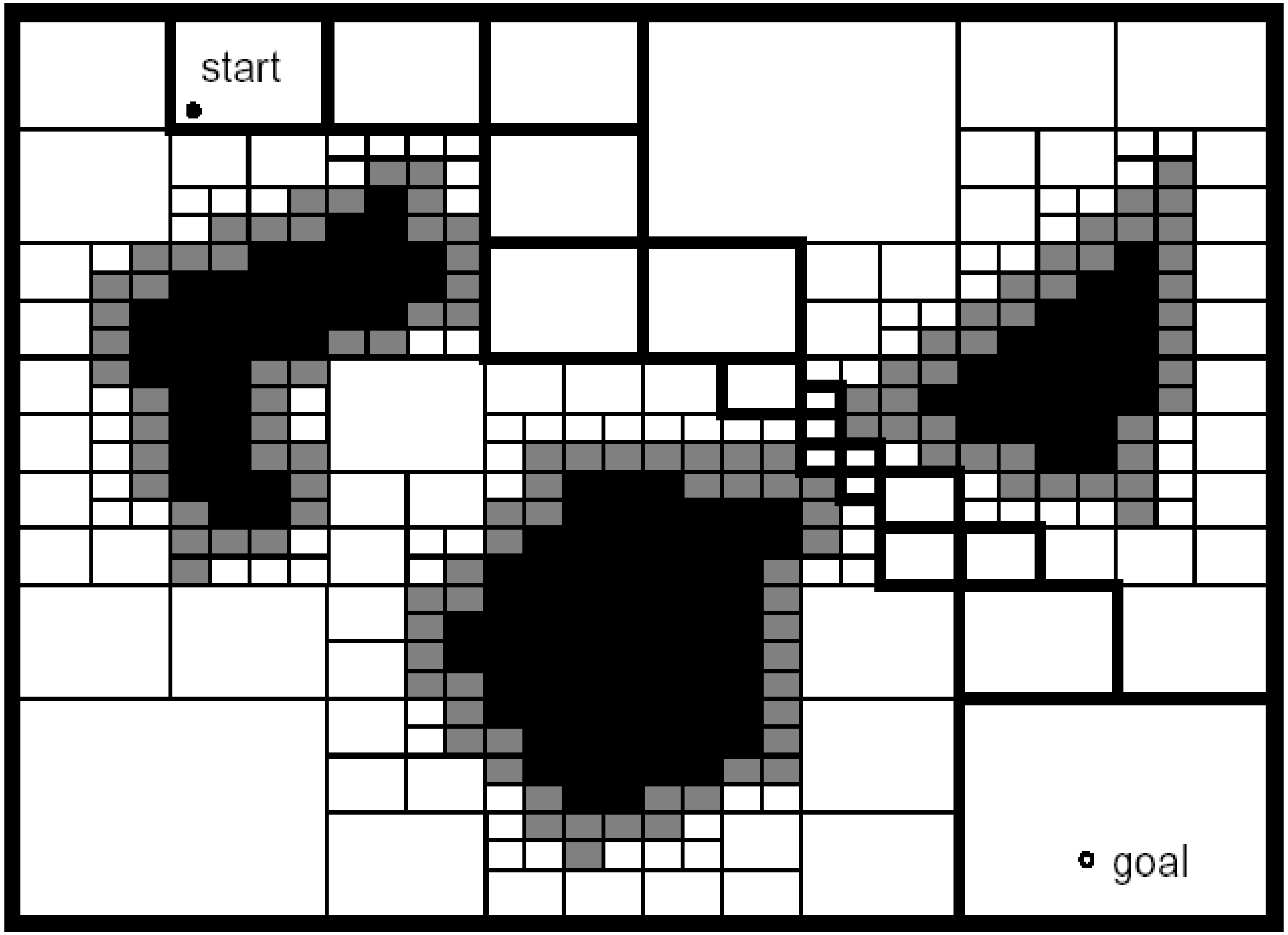


Graph Construction: Approximate Cell Decomposition (3/4)





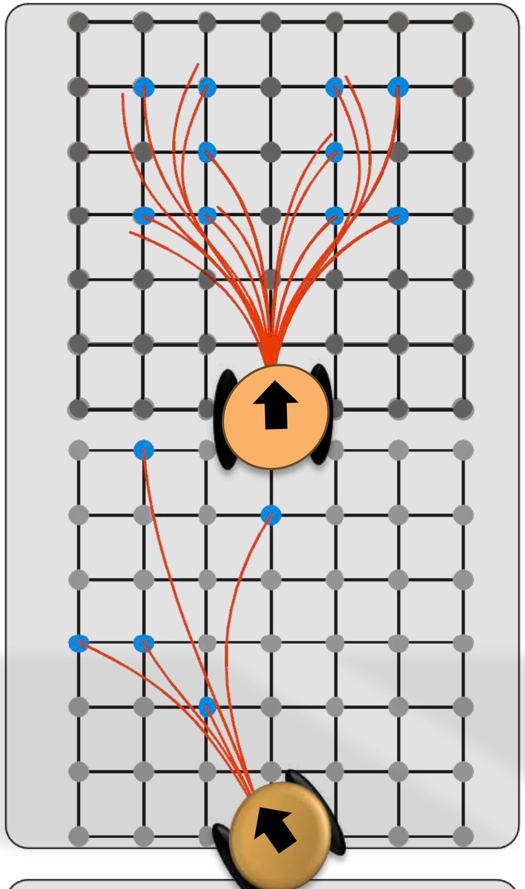
Graph Construction: Adaptive Cell Decomposition (4/4)



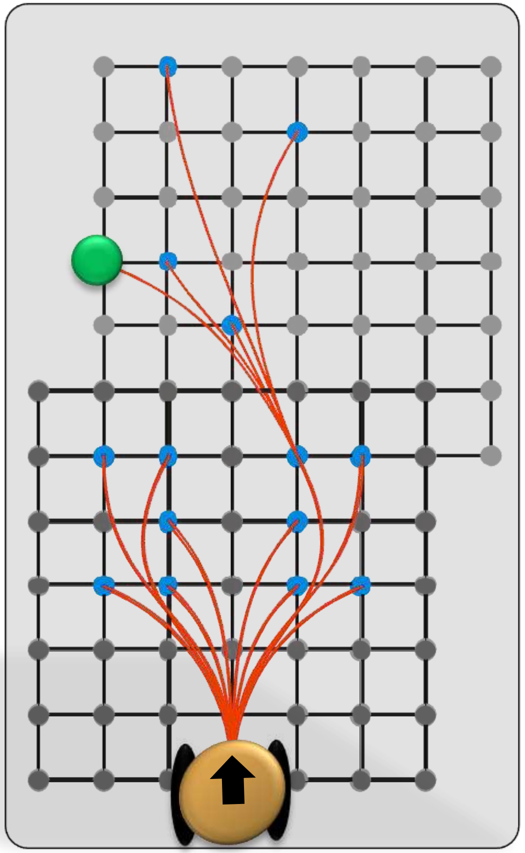
Graph Construction: State Lattice Design (1/2)

- Enforces edge feasibility

Offline:
Motion Model



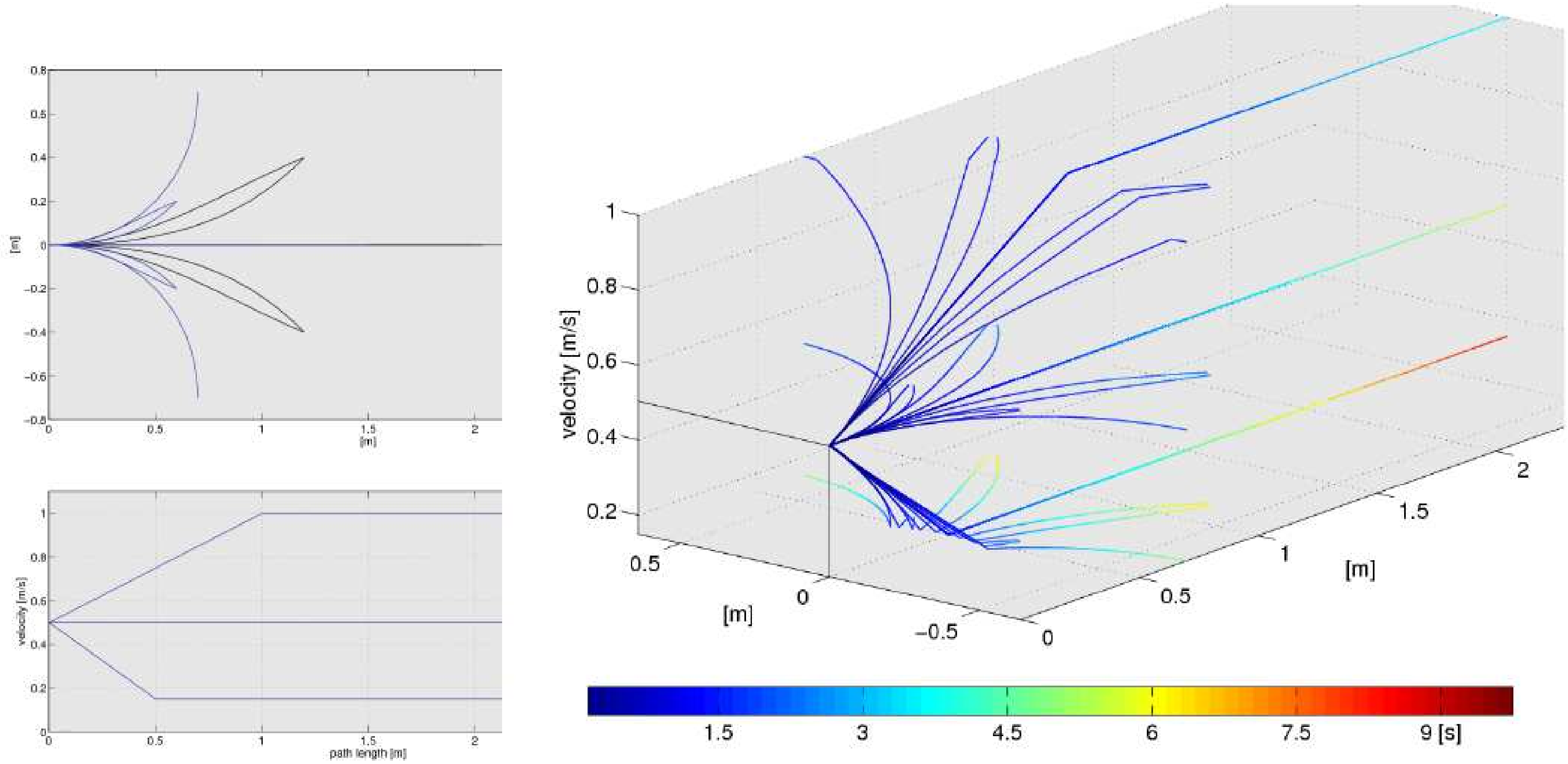
Offline:
Lattice Gen.



Online:
Incremental Graph
Constr.

Graph Construction: State Lattice Design (2/2)

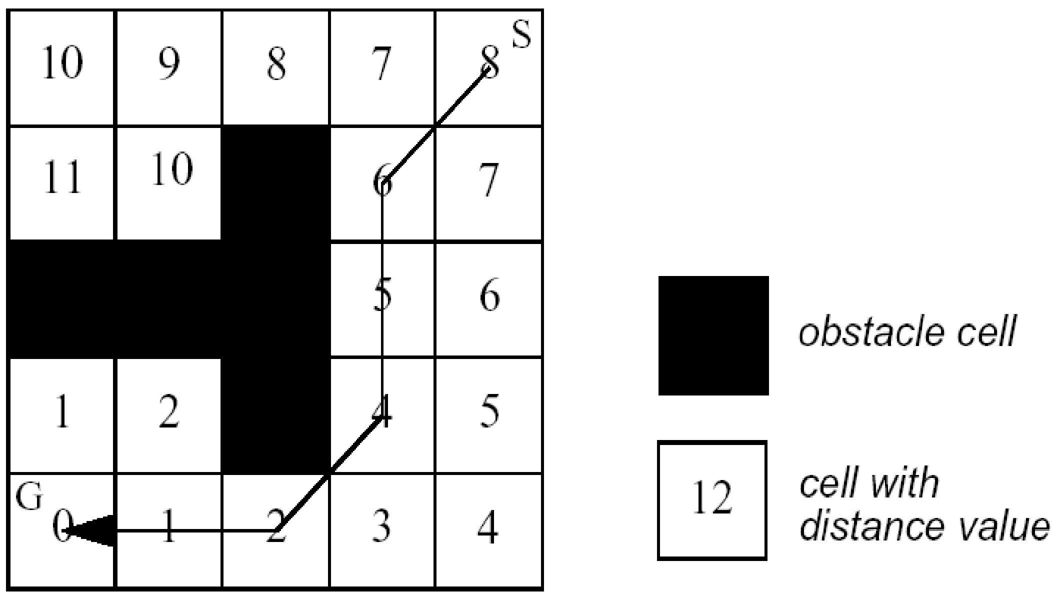
- State lattice encodes only kinematically feasible edges



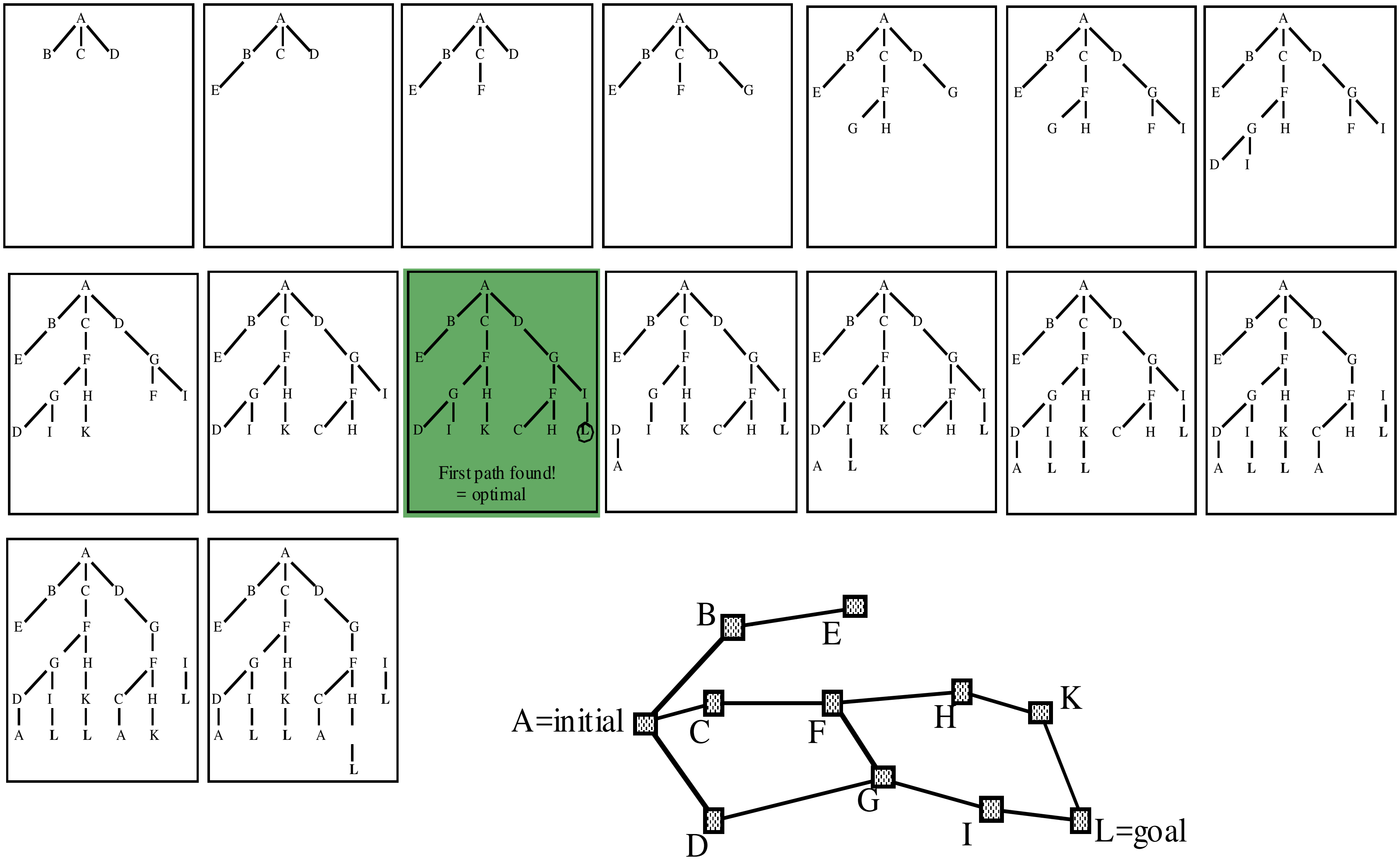
Graph Search

- Methods
  - Breath First
  - Depth First
  - Dijkstra
  - A* and variants
  - D* and variants
  - ...

- Discriminators
  - $f(n) = g(n) + \varepsilon\, h(n)$
  - $g(n') = g(n) + c(n,n')$



Graph Search Strategies: Breadth-First Search



Graph Search Strategies: Breadth-First Search

- Corresponds to a wavefront expansion on a 2D grid
- Use of a FIFO queue
  - First-found solution is optimal if all edges have equal costs
- Dijkstra‘s search is an „g(n)-sorted“ HEAP variation of breadth first search
  - First-found solution is guaranteed to be optimal no matter the (positive!) cell cost

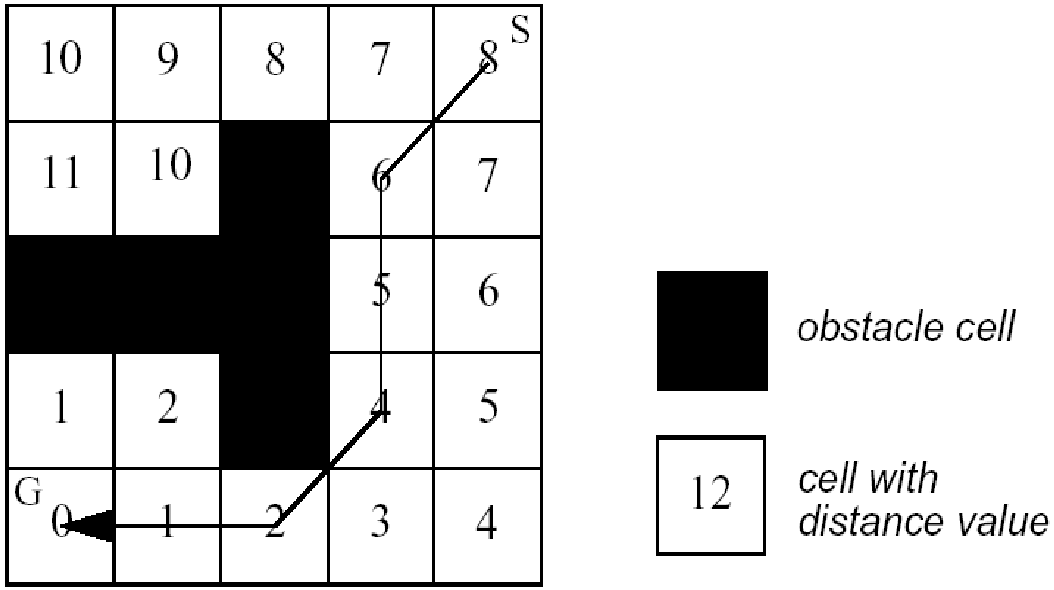
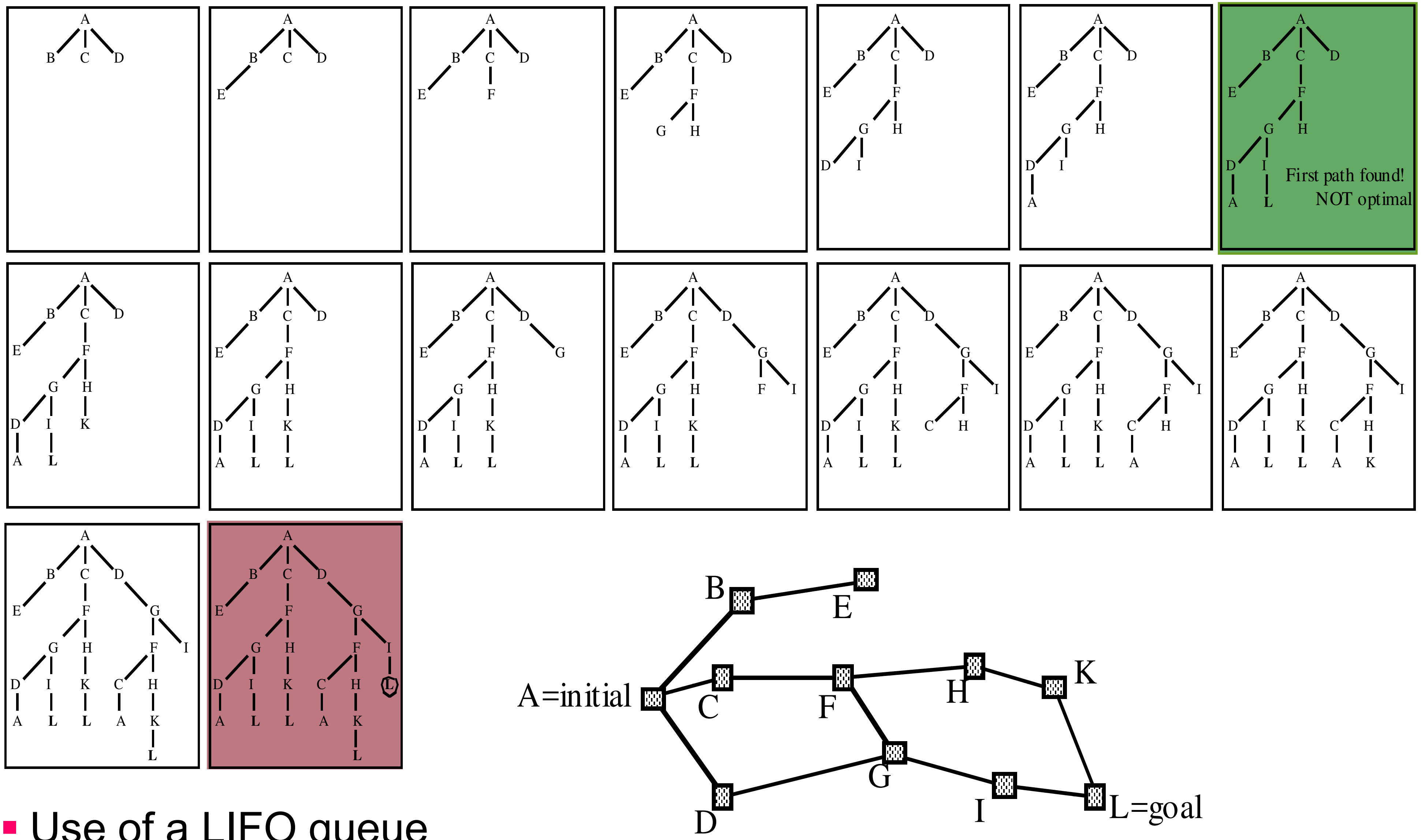


Fig. 1: NF1: put in each cell its L^1-distance from the goal position (used also in local path planning)

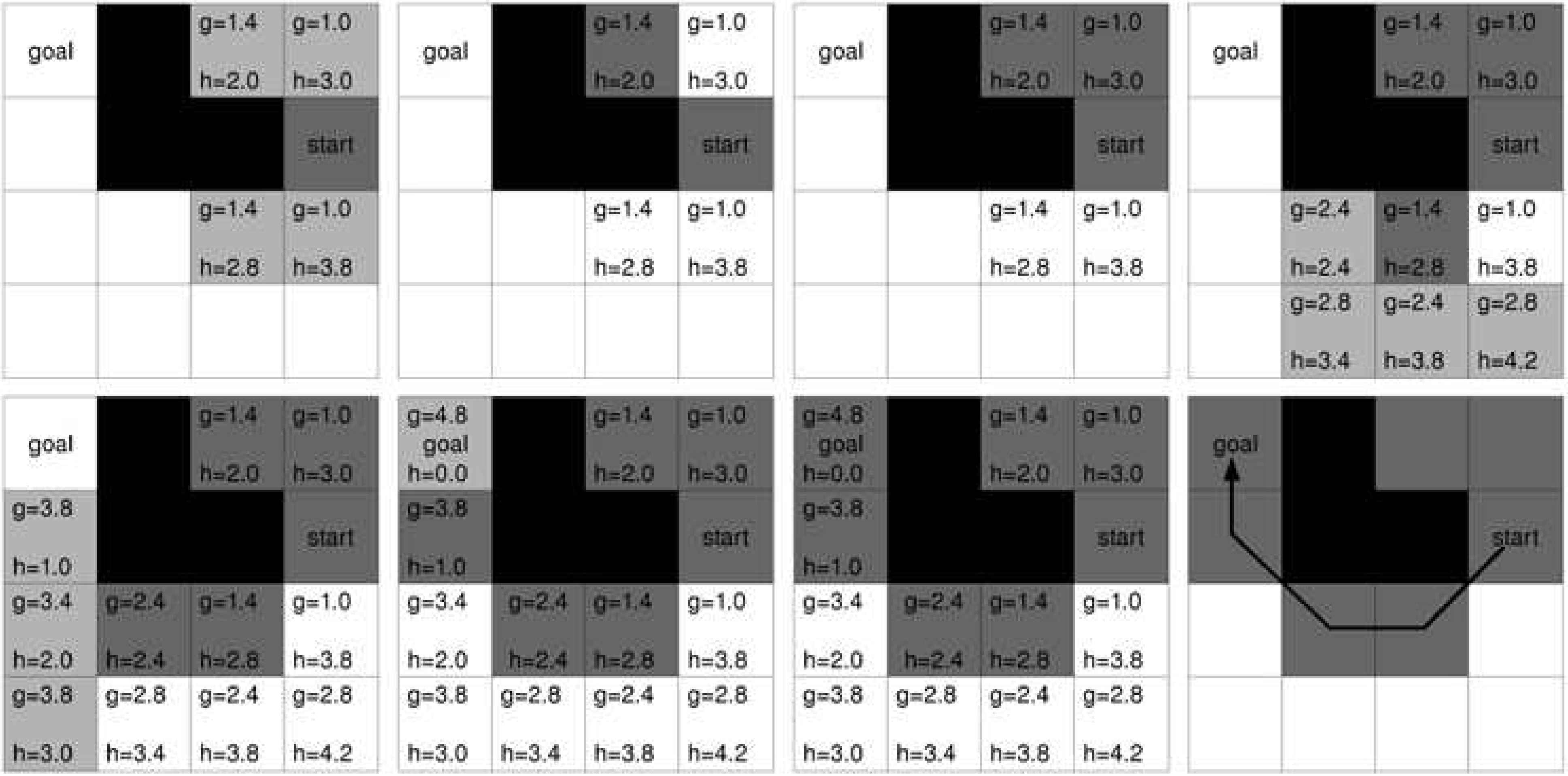
Graph Search Strategies: Depth-First Search



- Use of a LIFO queue
- Memory efficient (fully explored subtrees can be deleted)

Graph Search Strategies: A* Search

- Similar to Dijkstra's algorithm, A* also uses a HEAP (but „f(n)-sorted")
- A* uses a heuristic function h(n) (often Euclidean distance)
- $f(n) = g(n) + \varepsilon\, h(n)$



A*: Choice of heuristic

Requirements

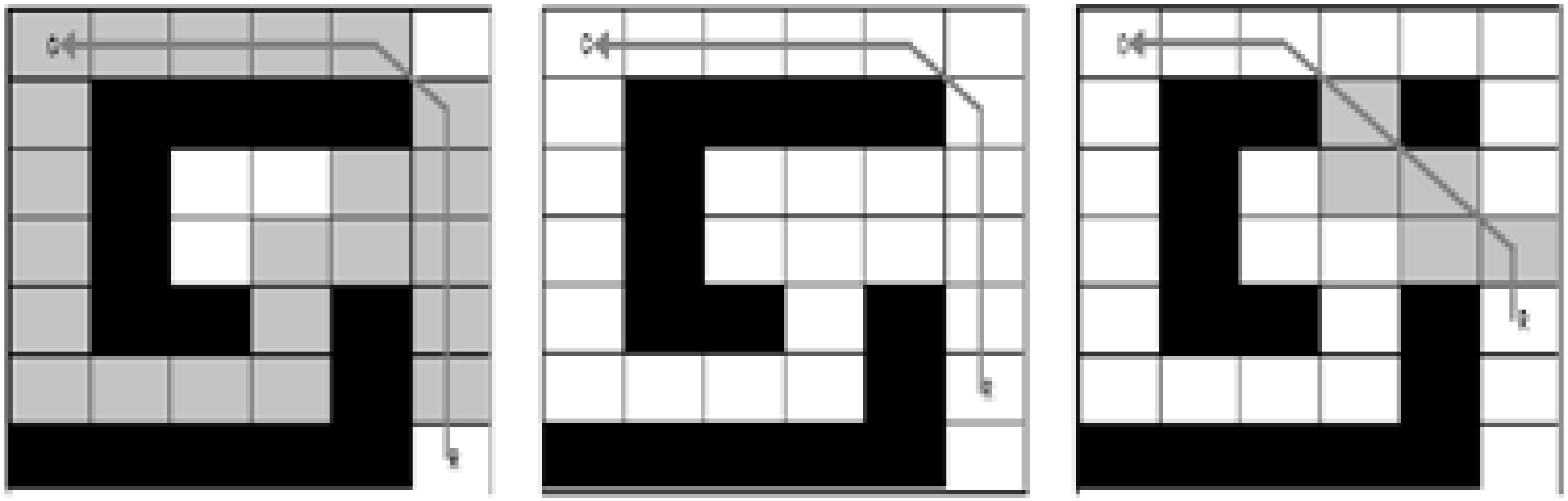
- Must not be over-estimating the real distance

Choices

- Euclidian distance
- Euclidian distance in 2D/3D for work-space in SE(2), SE(3).
- Distance L_∞, L_1, ...
- Result for another planning in a lower-dimension space.
- ...

Graph Search Strategies: D* Search

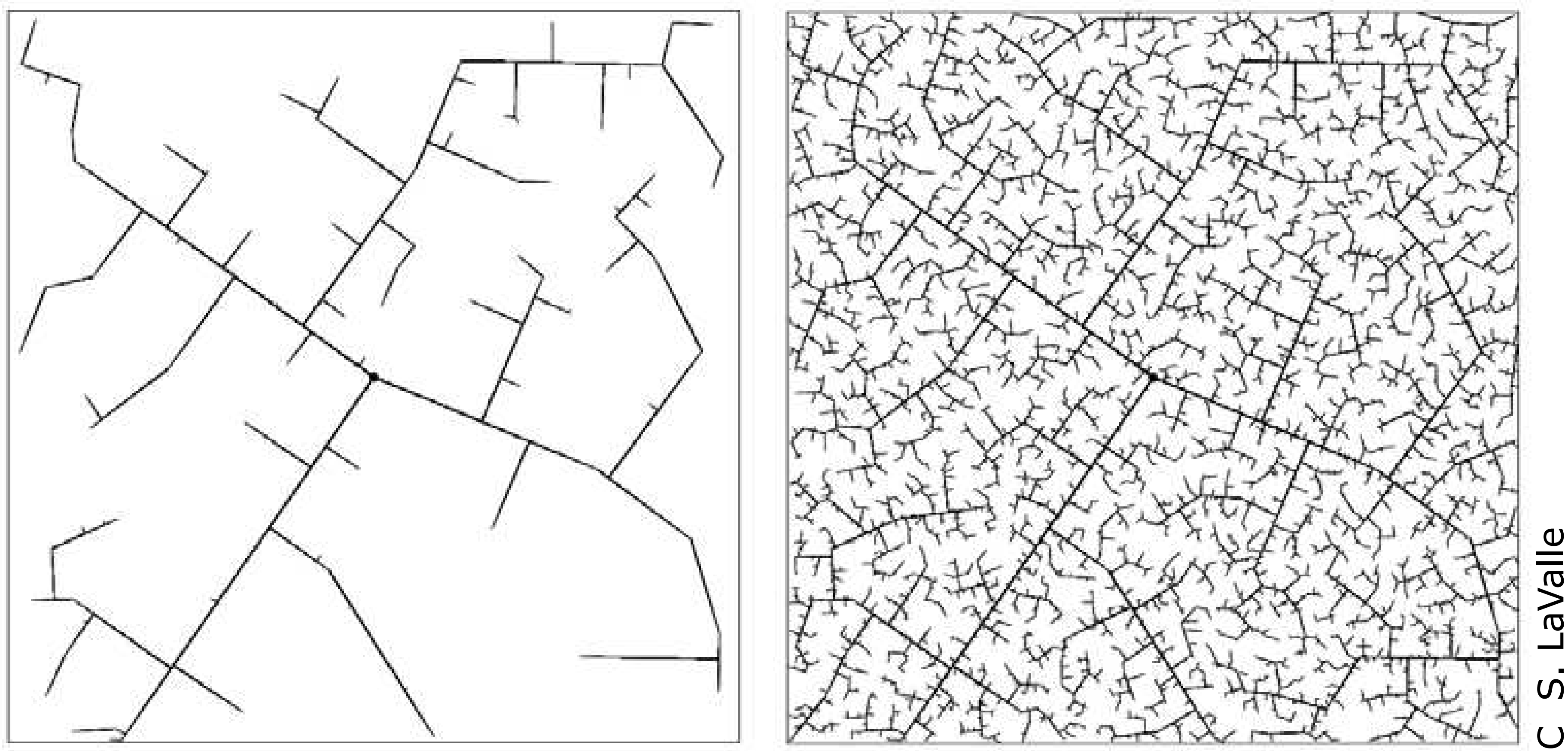
- Similar to A* search, except that the search begins from the goal outward
- f(n) = g(n) + ε h(n)
- First pass is identical to A*
- Subsequent passes reuse information from previous searches



C M. Likhachev

Graph Search Strategies: Randomized Search

- Most popular version is the rapidly exploring random tree (RRT)
  - Well suited for high-dimensional search spaces
  - Often produces highly suboptimal solutions



45 iterations   2345 iterations

Rapidly Exploring Trees

Reference

- Lavalle: Chapter 5, Section 5.5:
  `http://planning.cs.uiuc.edu/ch5.pdf`

Questions

- How would you account for obstacles?
- Where would you introduce motion constraints?

Outline

Conclusions

How to choose your motion planner?

- Low-dimension, completion guarantees: discrete grid, graph search
- Motion constraints: lattice plus discrete grid (eventually RET)
- High-dimension: randomized search
- Tight corridors: hybrid methods

Note: these are generic rules of thumb, not necessary definitive for a given problem...