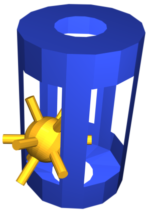
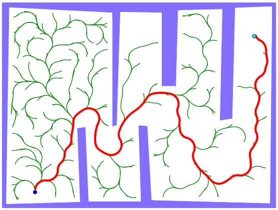
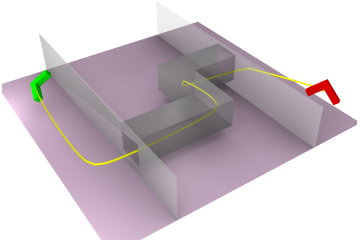
Motion planning I: basic concepts

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague



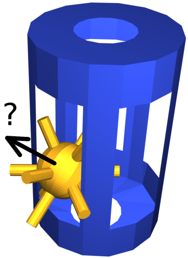


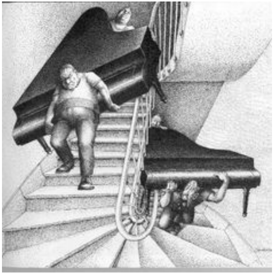


Motion planning: introduction

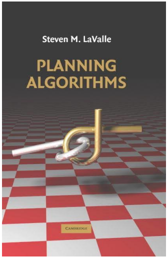
Informal definition: Motion planning is about automatic finding of ways how to move an object (robot) while avoiding obstacles (and considering other constraints).

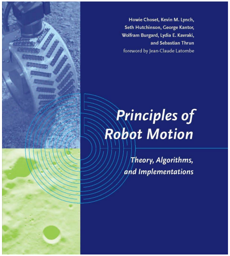
- Classical problem of robotics
- Also Piano mover's problem
- Relation to other fields
  - Mathematics: graph theory & topology
  - Computational geometry: collision detection
  - Computer graphics: visualizations
  - Control theory: feedback controllers required to navigate along paths
- Motion planning finds application in many practical tasks

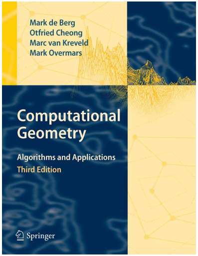




References







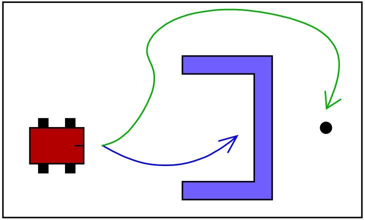
- S. M. LaValle, Planning algorithms, Cambridge, 2006, online: `planning.cs.uiuc.edu`
- H. Choset, K. M. Lynch et al., Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents series), Bradford Book, 2005
- M. de Berg, Computational Geometry: Algorithms and Applications, 1997
- C. Ericson. Real-time collision detection. CRC Press, 2004.

Path planning

- Requires models of robot and environment
- Can ensure finding global optimum
- Computationally intensive

Navigation/obstacle avoidance

- Fast, reactive way of reasoning
- Sensor-based navigation
- No (or limited) model of environment
- Cannot ensure reaching global goal
- Limited time horizon



navigation towards goal vs. planning towards goal

Planning is rather “global”; navigation is more “local”

Lectures overview

Introduction & motivation

↓

Formal definition, configuration space
Why we need discretization of configuration space

↓ ↓

Low-dimensional cases
Visibility graphs, Voronoi diagrams, . . .

General cases
Sampling-based planning
Planning under constraints

Technical details I
sampling, collision-detection, metrics, tips & tricks

Technical details II & Path following
physical simulations, basic path-following controllers

Motion planning: definitions

World $\mathcal{W}$

- is space where the robot operates
- $\mathcal{W}$ is usually $\mathcal{W} \subseteq \mathbf{R}^2$ or $\mathcal{W} \subseteq \mathbf{R}^3$
- $\mathcal{O} \subseteq \mathcal{W}$ are obstacles

Robot $\mathcal{A}$

- $\mathcal{A}$ is the geometry of the robot
- $\mathcal{A} \subseteq \mathbf{R}^2$ (or $\mathcal{A} \subseteq \mathbf{R}^3$)
- or set of links $\mathcal{A}_1, \ldots \mathcal{A}_n$ for $n-$body robot

Configuration q

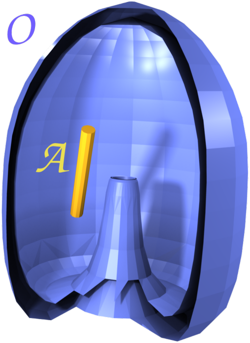
- Specifies position of **every** point of $\mathcal{A}$ in $\mathcal{W}$
- Usually a vector of **Degrees of freedom (DOF)**

$$q = (q_1, q_2, \ldots, q_n)$$

Configuration space $\mathcal{C}$ (aka C-Space or $\mathcal{C}$-space)

- $\mathcal{C}$ is a set of **all** possible configurations

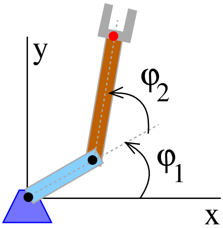
3D Bugtrap benchmark



$\mathcal{W} \subseteq \mathbf{R}^3$, $\mathcal{A} \subseteq \mathbf{R}^3$
$\mathcal{O} \subseteq \mathbf{R}^3$
(x, y, z) is 3D position
(r_x, r_y, r_z) is 3D rotation
$q = (x, y, z, r_x, r_y, r_z)$
$\mathcal{C}$-space is 6D

- A configuration is a **point** in $\mathcal{C}$
- $\mathcal{A}(q)$ is set of **all points** of the robot determined by configuration $q \in \mathcal{C}$
- Therefore, point $q \in \mathcal{C}$ **fully** describes how the robot looks in $\mathcal{W}$
- The number of dimensions of $\mathcal{C}$ equals to the number of DOFs of the robot.
- For robots with more than 4 DOFs, $\mathcal{C}$ is considered already as high-dimensional

Example: a robotic arm with two revolute joints; $q = (\varphi_1, \varphi_1) \rightarrow 2D \quad \mathcal{C}$-space
Robot geometry has two rigid shapes: $\mathcal{A}_1$ and $\mathcal{A}_2$



Obstacles in the configuration space: $\mathcal{C}_{\text{obs}}$

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \quad \mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$$

- $\mathcal{C}_{\text{obs}}$ contains robot-obstacle collisions and self-collisions
- Self-collisions: e.g. in the case of robotic arms
- q is feasible, if it is collision free $\rightarrow q \in \mathcal{C}_{\text{free}}$

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$$

Implicit definition of $\mathcal{C}_{\text{obs}}$

- We cannot (generally) enumerate points in $\mathcal{C}_{\text{obs}}$
- Difficult to determine the nearest colliding configuration
- The main reason, why high-dimensional $\mathcal{C}$ is difficult to search!

How to determine if q is collision-free or not?

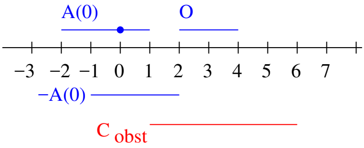
- Generally: compute $\mathcal{A}(q)$ and detect collisions with $\mathcal{O} \rightarrow$ time consuming
- Special cases: direct representation of $\mathcal{C}$, then point-location query

- $\mathcal{C}$-space can be explicitly constructed using Minkowski sum of $\mathcal{A}$ and $\mathcal{O}$
- Minkowski sum $\oplus$ of two sets X and Y is

$$X \oplus Y = \{x + y \in \mathbf{R}^n | x \in X \text{ and } y \in Y\}$$

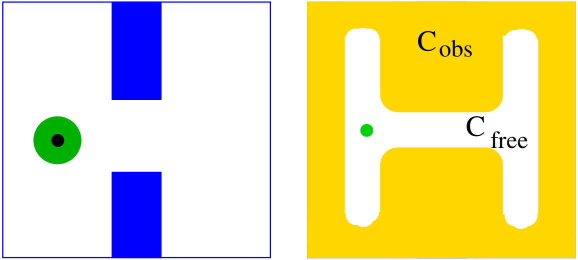
  where n is the dimension
- $\mathcal{C}_{\text{obs}}$ can be computed as $\mathcal{O} \oplus -\mathcal{A}(0)$
- $\mathcal{A}(0)$ is the robot at origin
- $-\mathcal{A}(0)$ is achieved by replacing all $x \in \mathcal{A}(0)$ by $-x$

Example: 1D robot $\mathcal{A} = [-2, 1]$ and obstacle $\mathcal{O} = [2, 4]$:



$\mathcal{C}_{\text{obs}} = [1, 6]$

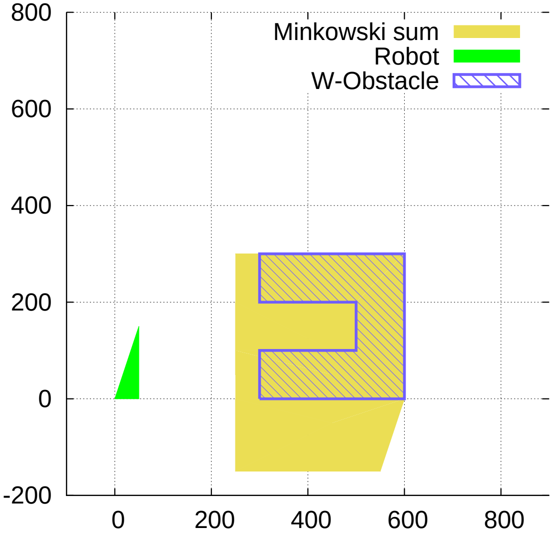
- 2D workspace $\mathcal{W} \subseteq \mathbf{R}^2$
- 2D disc robot $\mathcal{A} \subseteq \mathbf{R}^2$, reference point in the disc's center
- We assume **only translation**
- Therefore, configuration $q = (x, y)$ and $\mathcal{C}$ is 2D



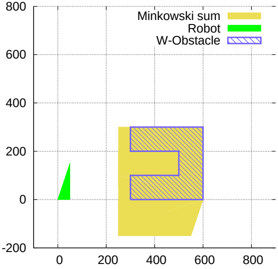
Workspace   Configuration space

- All $q \in \mathcal{C}_{\text{free}}$ are collision-free $\rightarrow \mathcal{A}(q) \cap \mathcal{O} = \emptyset$
- Volume of $\mathcal{C}_{\text{free}}$ depends both on the robot and obstacles
- What happens if the robot is a point?

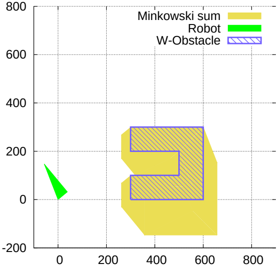
- 2D robot, only translation, $q = (x, y) \rightarrow$ 2D $\mathcal{C}$



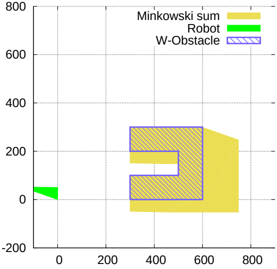
- 2D robot, translation + rotation, $q = (x, y, \varphi) \rightarrow$ 3D $\mathcal{C}$
- Requires to compute Minkowski sum for each rotation



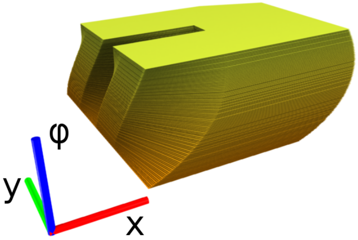
$\varphi = 0$

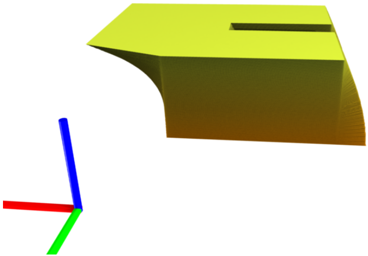


$\varphi = 45$

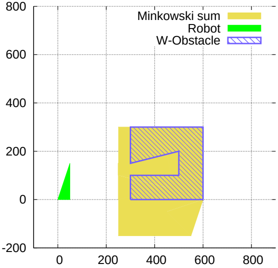


$\varphi = 90$

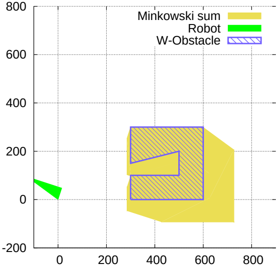




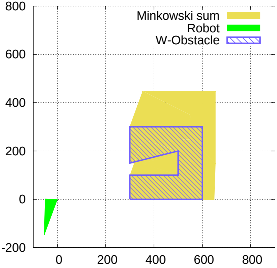
- 2D robot, translation + rotation, $q = (x, y, \varphi) \rightarrow$ 3D $\mathcal{C}$
- Requires to compute Minkowski sum for each rotation



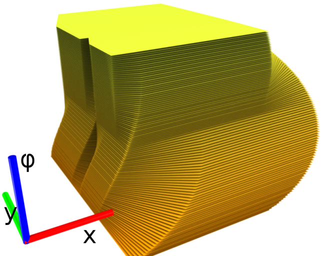
$\varphi = 0$

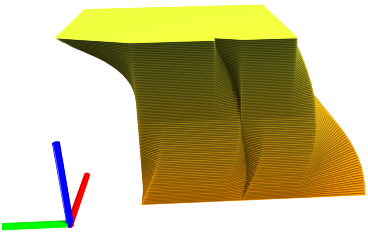


$\varphi = 50$



$\varphi = 90$





Explicit construction of $\mathcal{C}$

- Construction of $\mathcal{C}$ Minkowski sums is straightforward, but . . .
- We have only 2D/3D models of robots and obstacles

$\rightarrow$ directly we can construct $\mathcal{C}$ only for “translation only” systems

- Other DOFS need to be discretized and Minkowski sum computed for each combination

Minkowski sum of two objects of n and m complexity

2D polygons

- convex $\oplus$ convex, $O(m+n)$
- convex $\oplus$ arbitrary, (mn)
- arbitrary $\oplus$ arbitrary, $(m^2 n^2)$

3D polyhedrons

- convex $\oplus$ convex, $O(mn)$
- arbitrary $\oplus$ arbitrary, $(m^3 n^3)$

- Explicit construction of $\mathcal{C}$ is computationally demanding!
- Not practical for high-dimensional systems

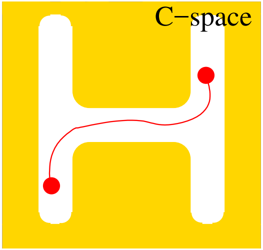
- A **path** in $\mathcal{C}$ is a continuous curve connecting two configurations q_{init} and q_{goal}:

$$\tau : s \in [0, 1] \to \tau(s) \in \mathcal{C}; \quad \tau(0) = q_{\text{init}} \text{ and } \tau(1) = q_{\text{goal}}$$

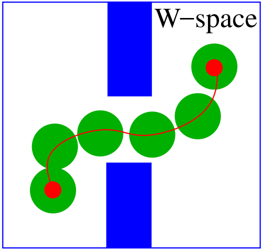
- A **trajectory** is a path parametrized by time

$$\tau : t \in [0, T] \to \tau(t) \in \mathcal{C}$$

- Trajectory/path defines motion is workspace



Path in $\mathcal{C}$



Workspace motion

Path/motion planning problem

Let's assume we have

- model of the world $\mathcal{W}$ and robot $\mathcal{A}$
- and configurations $q_{\text{init}}, q_{\text{goal}} \in \mathcal{C}_{\text{free}}$

Path planning

- To find a collision-free path $\tau(s)$ from q_{init} to q_{goal}
- i.e., $q(s) \in \mathcal{C}_{\text{free}}$ for all $s \in [0, 1]$, $s(0) = q_{\text{init}}$, $s(1) = q_{\text{goal}}$

Motion planning

- To find a collision-free trajectory $\tau(t)$ from q_{init} to q_{goal}
- i.e., $q(t) \in \mathcal{C}_{\text{free}}$ for all $t \in [0, T]$, $s(0) = q_{\text{init}}$, $s(T) = q_{\text{goal}}$

Other specifications

- Kinematic constraints (e.g. 'car-like' vehicle)
- Dynamic constraints (e.g. maximal acceleration)
- Task constraints (e.g 'do not spill the beer')

Confusion in terminology

- Path/motion planning are studied in several disciplines
  - Robotics, computation geometry, mathematics, biology
  - . . . since 1950's !
- Each field uses different meaning for "path" and "trajectory"

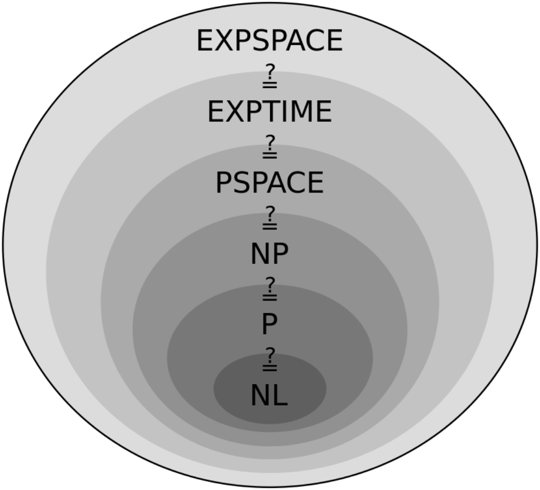
  . . . and different meaning for path/motion planning
- this continues up to now

What is then the "trajectory"?

- Robotics (including this lecture): path + time
- Control-oriented part of robotics: path + time + control inputs
- Computational biology: 3D path of atom(s) (with or without time)

Before you start to solve a planning problem, define (or agree on) the basic terms first!

Complexity of motion planning



General motion planning problem is PSPACE-complete.

☛ J. Canny. The complexity of robot motion planning. MIT press, 1988.

Robotic task

Specification

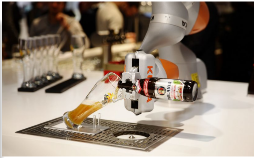
Description of inputs, obstacles, robot, constraints, kinematics,

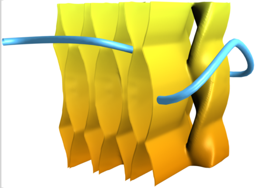
Motion planning

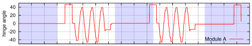
computing path/trajectory

Execution

Path following, control of motors







Motion planning

Continuous problem

$\mathcal{C}$-space specification, $q_{\mathrm{init}}, q_{\mathrm{goal}}$

Discretization of $\mathcal{C}$

Explicit construction random/deterministic sampling of $\mathcal{C}$

Graph search

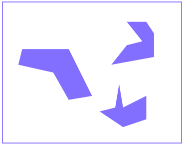
Dijkstra, A*, D*, . . .

The art-of-motion-planning

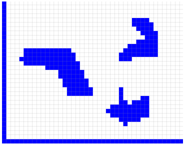
- Understand and formulate the problem, define $\mathcal{C}$
- Apply suitable method to represent $\mathcal{C}$ by a graph
- Search the graph

World representations

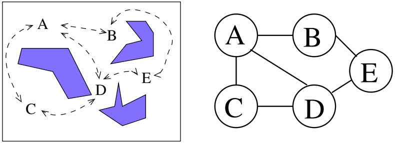
- Map: the representation of the world
  - grid-maps: 2D/3D/nD arrays/grids — represent both $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$
  - geometric maps: polygons, polyhedrons (usually for $\mathcal{C}_{\text{obs}}$)
  - topological maps: relations between regions of $\mathcal{C}_{\text{free}}$
- Properties
  - Memory requirements
  - Supported operations (e.g. merging maps, adding new information, deleting obstacles, ...)
  - Computational complexity of these procedures
  - Precision
  - Robustness (with respect to numerical errors)
- One should always choose a map suitable for the given application



polygons



grid

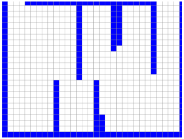


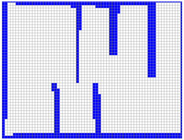
topology

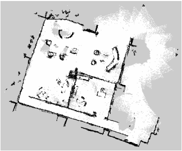
Grid maps

- 2D or 3D array (grid) of cells
- Binary maps: 0/1 (obstacle, free spaces)
- Probability: 0–1 (0=free space, 1=obstacle)
  - occupancy grid
  - often used for integration of sensor data

✓ Metric information (distance/angle/area . . .)
✓ Easy implementation
✓ Efficient search for obstacle cells, nearest obstacle cell, . . .
✓ Straightforward update of cells & map merging
✓ Integration of data from different sensors
✗ High memory requirements
  - depends on environment size & map resolution
  - practical limit to 2D and 3D environments







Polygonal maps

- 2D worlds
- Obstacle is represented by polygon

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

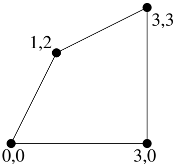
- (x_i, y_i) are vertices
- The map is the collection of obstacles
- Simple polygon: does not intersect itself, no holes
- Polygons with holes: contour + one or more holes

✓ Memory efficient, easy to process, metric information

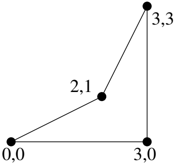
✓ Fast tests for collisions, point location

✗ Numerical stability of (some) algorithms

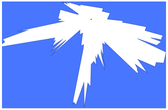
✗ Number of vertices can dramatically grow if map is built from (unfiltered) sensor data



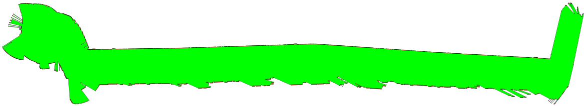
Convex



Non-convex



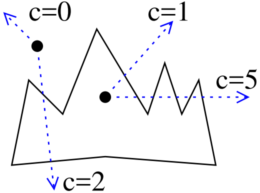
Polygon from Lidar

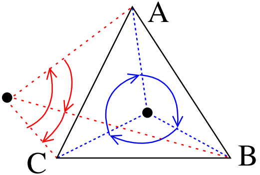


Map $\sim$ 100 $\times$ 5 m, $\sim$1k vertices

Point-in-polygon

- Is a point inside/outside of a polygon?
- **Crossing test:**
  - shot a ray from the query point and compute crossings
  - the point is inside if the number of crossings is odd
- **Winding number:**
  - sum up (signed) angles from query point to all vertices
  - point is outside, if the sum is near-zero
  - slow (practically): required trigonometric functions
- Crossing test & Winding number: for convex/non-convex, $O(n)$
- Faster algorithm for convex polygons: $O(\log n)$





Collision-detection

- Used to determine if $q \in \mathcal{C}_{\text{free}}$ or $q \in \mathcal{C}_{\text{obs}}$
- Leads to computations of intersections between polygons $\mathcal{A}(q)$ and $\mathcal{O}$
- Collision determination: compute the result of the collision
- Collision detection: only report if there is collision or not (True/False)

Intersection of two polygons *P* and *Q*

- The result is the polygon of intersection $\rightarrow$ collision determination
- Time complexity $O(|P| + |Q|)$

Collision detection

- Naïve: check all segments of $\mathcal{A}(q)$ vs. all segments of $O \rightarrow O(|\mathcal{A}||\mathcal{O}|)$
- Disadvantage: also “distant” segment are tested (slow)
- Better solution: sweepline method, e.g. Bentley-Ottman algorithm

☛ Bentley, J. L.; Ottmann, T. A. (1979), "Algorithms for reporting and counting geometric intersections", IEEE Transactions on Computers, C-28 (9)

Special cases with an explicit representation of $\mathcal{C}$

Point robot in 2D or 3D $\mathcal{W}$

- The map of $\mathcal{W}$ is also representation of $\mathcal{C}$
- Polygons/polyhedrons are suitable

Disc/sphere robot in 2D or 3D $\mathcal{W}$

- The obstacles are “enlarged” by radius of the robot (Minkowski sum)
- Then, representation of $\mathcal{W}$ is also representation of $\mathcal{C}$

Geometric planning methods

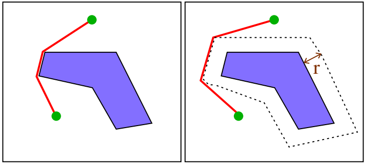
- Assume point/disc robots
- Use geometric (usually polygonal) representation of W (=$\mathcal{C}$)
- Voronoi diagram, Visibility map, Decomposition-based methods

Motion planning

Continuous problem
$\mathcal{C}$-space specification, q_{init}, q_{goal}

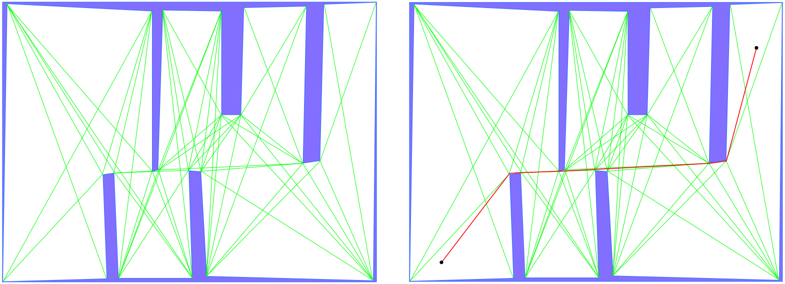
Discretization of $\mathcal{C}$
Explicit construction random/deterministic sampling of $\mathcal{C}$

Graph search
Dijkstra, A*, D*, . . .



Visibility graph

- Two points v_i, v_j are visible $\iff$ $(sv_i + (1-s)v_j) \in \mathcal{C}_{\text{free}}, \quad s \in (0,1)$
- Visibility graph (V, E), V are vertices of polygons, E are edges between visible points
- Start/goal are connected in same manner to visible vertices



Visibility graph — After connecting start/goal + path

- No clearance
- Suitable only for 2D

Visibility graph (VG)

- Straightforward, näive, implementation $O(n^3)$

Input: polygonal obstacle
Output: visibility graph $G = (V, E)$
V = all vertices of polygonal obstacles
foreach $u, v \in V$ **do**
  foreach *obstacle edge* e **do**
    if *segment* u, v *intersects* e **then**
      continue;
    add edge u, v to E

- n^2 pairs of vertices
- Complexity of checking one intersection is $O(n)$

$\rightarrow$ Total complexity $O(n^3)$

Fast methods

- Lee's algorithm $O(n^2 \log n)$
- Overmars/Welz method $O(n^2)$
- Ghosh/Mount method $O(|E| n \log n)$

☛ Lee, Der-Tsai, Proximity and reachability in the plane, 1978

☛ D. Coleman, Lee's O(n2 log n) Visibility Graph Algorithm Implementation and Analysis, 2012.

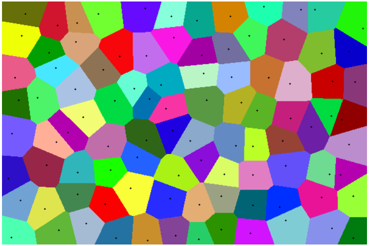
☛ M. H. Overmars, E. Welzl, New methods for Computing Visibility Graphs, Proc. of 4th Annual Symposium on Comp. Geometry, 1998

☛ S. Ghosh and D. M. Mount, An output-sensitive algorithm for computing visibility graphs, SIAM Journal on Computing, 1991

Voronoi diagram

- Let $P = v_1, \ldots, v_n$ are n distinct points ("input sites") in a $d-$dimensional space
- Voronoi Diagram (VD) divides P into n cells $V(p_i)$

$$V(p_i) = \{x \in \mathbf{R}^d : ||x - p_i|| \leq ||x - p_j|| \;\; \forall j \leq n\}$$



- Cells are convex
- Used in point location (1-nn search), closes-pair search, spatial analysis
- Construction using Fortune's method in $O(n \log n)$

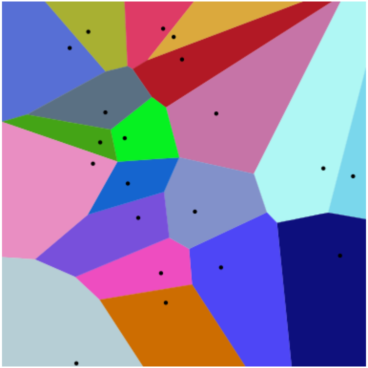
☛ S. Fortune. A sweepline algorithm for Voronoi diagrams. Proc. of the 2nd annual composium on Computational geometry. pages 313-322. 1986.

Voronoi diagram

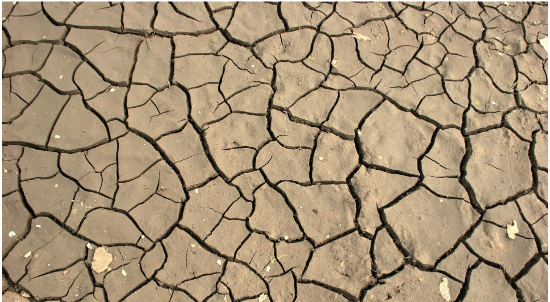
- Let $P = v_1, \ldots, v_n$ are n distinct points ("input sites") in a $d-$dimensional space
- Voronoi Diagram (VD) divides P into n cells $V(p_i)$

$$V(p_i) = \{x \in \mathbf{R}^d : ||x - p_i|| \leq ||x - p_j|| \;\; \forall j \leq n\}$$

- Note, that other metrics can be considered!

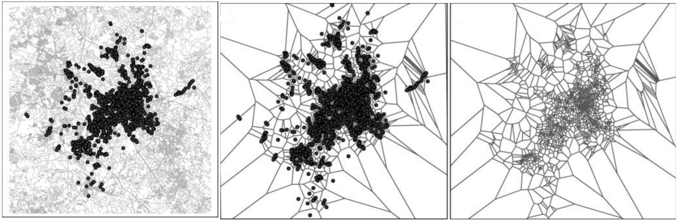


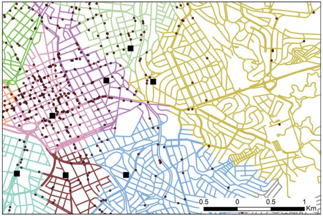
- VD can be found also in nature



Voronoi diagram: spatial analysis

- One of first analysis was Cholera epidemic in London
- Often used in criminology

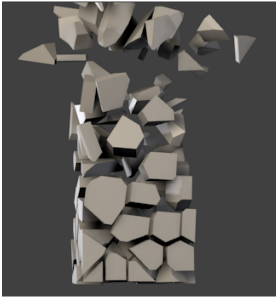




☛ Melo, S. N. D., Frank, R., Brantingham, P. (2017). Voronoi diagrams and spatial analysis of crime. The Professional Geographer, 69(4), 579-590.

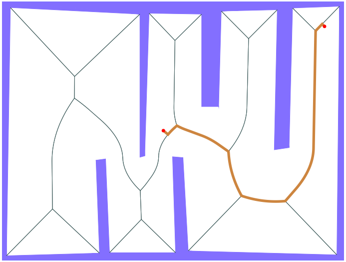
Voronoi diagram in computer graphics

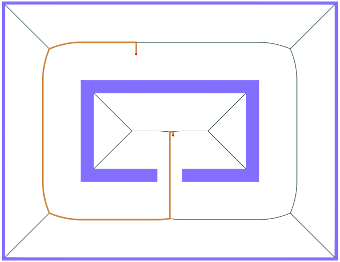
- Used in many low-level routines (e.g., point location)
- Modeling fractures
  - Object is filled with some random points
  - VD is computed to provide set of convex cells
  - Interaction between cells can be modeled e.g. using rigid body dynamics

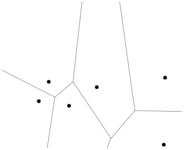



Generalized Voronoi diagram

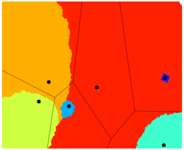
- Many types of Voronoi Diagrams exist
  - e.g. points + weights, segments, spheres, ...
- Segment Voronoi Diagram (SVD) is computed on line-segments describing obstacles
- ✓ Maximize the path clearance
  - biggest possible distance between path and the nearest obstacle



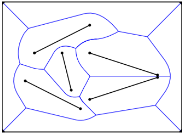




Classic VD



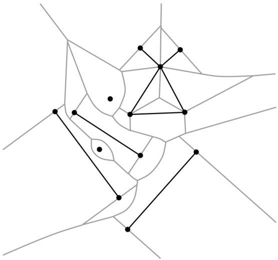
Weighted VD



Segment VD

Algorithms for computing Segment Voronoi diagram of n segments

- Lee & Drysdale: $O(n \log^2 n)$, no intersections
- Karavelas: $O((n+m)\log^2 n)$, m intersections between segments

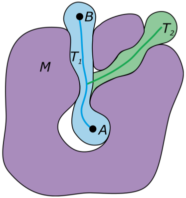


Karavelas 2004

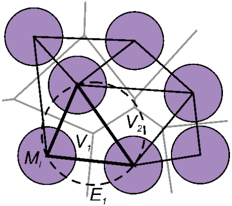
☛ Karavelas, M. I. "A robust and efficient implementation for the segment Voronoi diagram." International symposium on Voronoi diagrams in science and engineering. 2004

☛ Lee, D. T, R. L. Drysdale, III. "Generalization of Voronoi diagrams in the plane." SIAM Journal on Computing 10.1 (1981): 73-87.

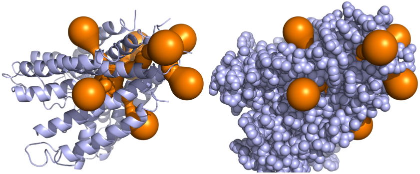
- Proteins are modeled using hard-sphere model
- Weighted Voronoi diagram of the spheres (weight is the atom radii — Van der Waals radii)
- Path in the Voronoi diagram reveals “void space” and “tunnels”
- Tunnel properties (e.g. bottleneck) estimate possibility of interaction between protein and a ligand



Tunnels*



Voronoi diagram *

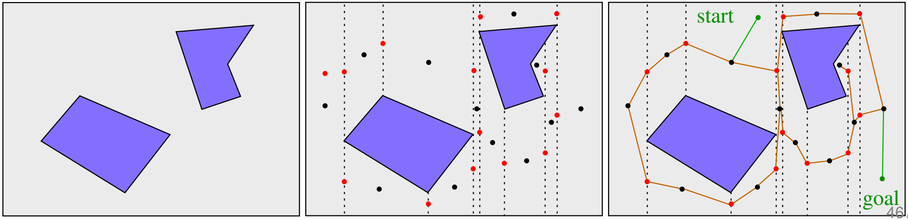

Tunnels on 1BL8

* ☛ A. Pavelka, E. Sebestova, B. Kozlikova, J. Brezovsky, J. Sochor, J. Damborsky, CAVER: Algorithms for Analyzing Dynamics of Tunnels in Macromolecules, IEEE/ACM Trans. on compt. biology and bioinformatics, 13(3), 2016.

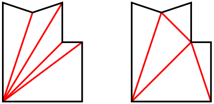
- The free space is partitioned into a finite set of cell
  - Determination of cell containing a point should be trivial
  - Computing paths inside the cells should be trivial
- The relations between the cells is described by a graph
- Path from start to goal is solved on the graph

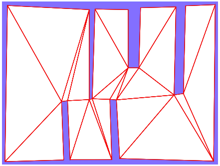
Vertical cell decomposition

- Make vertical line from each vertex, stop at obstacles
- Determine centroids of the cells, centers of each segments
- Graph connects the neighbor centroids through the centers
- Connect start/goal to centroid of their cells
- Can be built in $O(n \log n)$ time



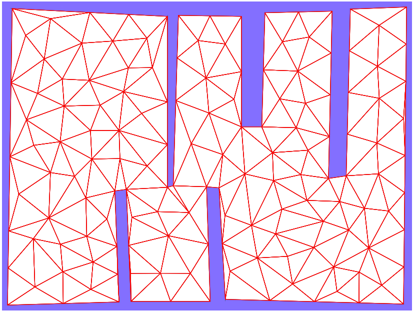
- Variant of decomposition-based methods
- $\mathcal{C}_{\text{free}}$ is triangulated
- Can be computed in $O(n \log \log n)$ time
- Polygons can be triangulated in many ways
- $\mathcal{C}_{\text{free}}$ is represented by graph $G = (V, E)$
    - V are centroids of the triangles
    - $E = (e_{i,j})$ if Δ_i is neighbor of Δ_j
- Or
    - V are vertices of the triangulation
    - E are edges of the triangulation
- Planning: start/goal are connected to graph, then graph search
- How to triangulate polygonal map composed of n disconnected polygons?



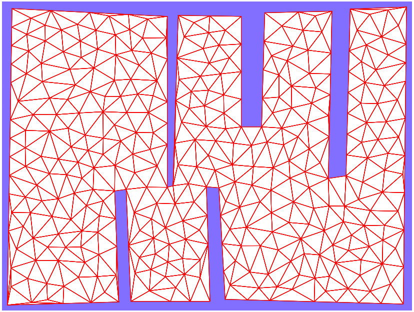


Decomposition via triangulation II

- Finer triangulation via Constrained Delaunay Triangulation (CDT)
  - if a triangle does not meet a criteria, it is further triangulated
  - criteria: triangle area or the largest angle



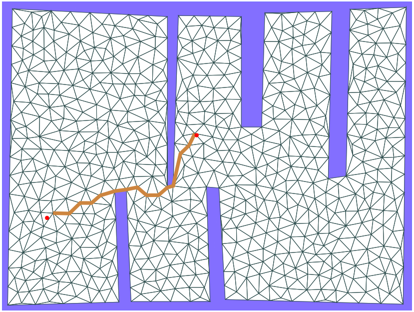
CDT



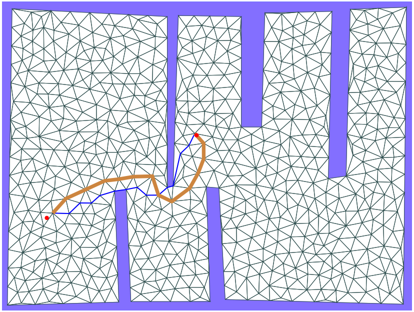
Finer CDT (area of Δ)

Decomposition via triangulation II

- Finer triangulation via Constrained Delaunay Triangulation (CDT)
  - if a triangle does not meet a criteria, it is further triangulated
  - criteria: triangle area or the largest angle



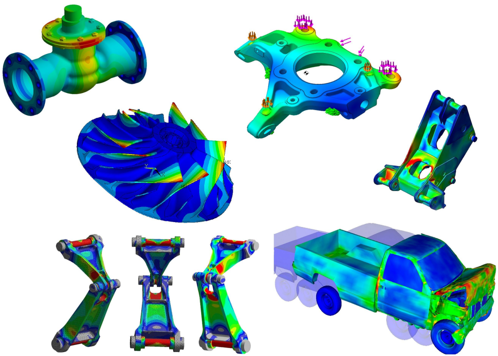
Path on edges

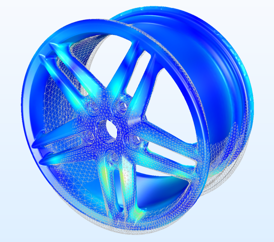


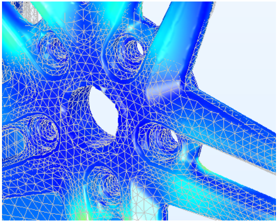
Modification: ignore segments connecting obstacles

CDT in civil engineering

- Structural analysis: modeling behavior of a structure under load, wind, pressure, . . .
- Finite element method







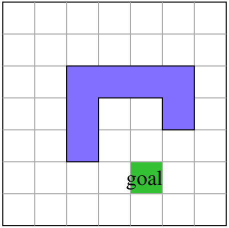
Navigation functions

- Let's assume a forward motion model

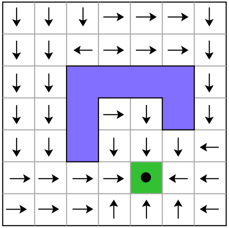
$$\dot{q} = f(q, u)$$

  where $q \in \mathcal{C}$ and $u \in \mathcal{U}$; $\mathcal{U}$ is the action space
- The navigation function $F(q)$ tells which action to take at q to reach the goal

Example: robot moving on grid, actions $\mathcal{U} = \{\rightarrow, \leftarrow, \uparrow, \downarrow, \bullet\}$



Discrete planning problem



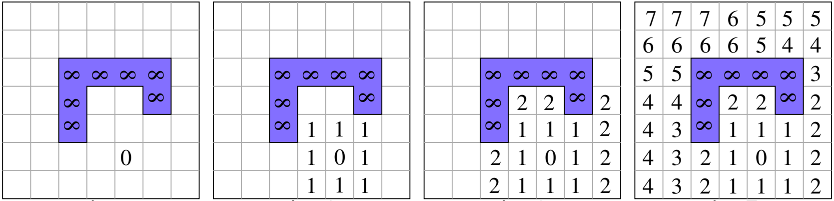
Navigation function

- In discrete space, navigation f. is by-product of graph-search methods

- Simple way to compute navigation function on discrete space X
- Explores X in “waves” starting from goal until all states are explored

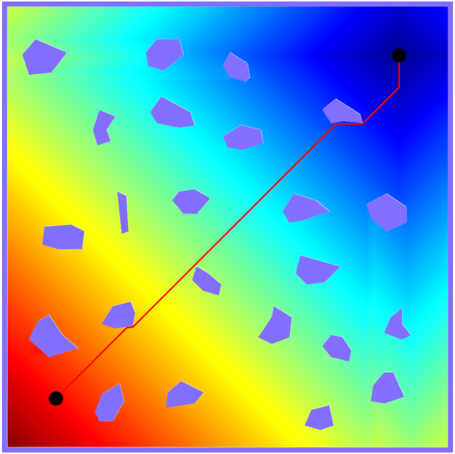
```
open = {goal}
i = 0
while open ≠ ∅ do
    wave = ∅                      // new wave
    foreach x ∈ open do
        value(x) = i
        foreach y ∈ N(x) do
            if y is not explored then
                add y to wave
    i = i + 1
    open = wave
```

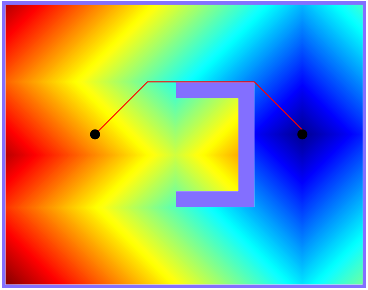
- $N(x)$ are neighbors of x
- 4-/8-point connectivity
- The increase of the wave value i should reflect the distance between x and its neighbors
- Path is retrieved by gradient descend from start
- $O(n)$ time for n reachable states

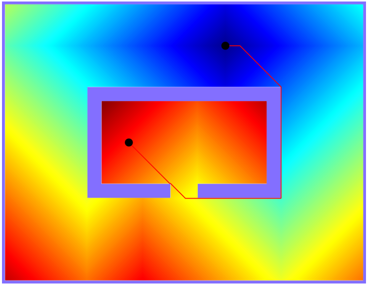


goal state | $i = 1$ | $i = 2$ | $i = 7$

Wavefront planner

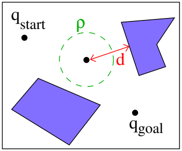






Potential field: principle

- Potential field U: the robot is repelled by obstacles and attracted by q_{goal}
- Attractive potential U_{att}, repulsive potential U_{rep}
- Weights K_{att} and K_{rep}, d is the distance to the nearest obstacle, ϱ is radius of influence



$$U_{att}(q) = \frac{1}{2} K_{att} dist(q, q_{\mathrm{goal}})^2 \qquad U_{rep}(q) = \begin{cases} \frac{1}{2} K_{rep}(1/d - 1/\varrho)^2 & \text{if } d \leq \varrho \\ 0 & \text{otherwise} \end{cases}$$

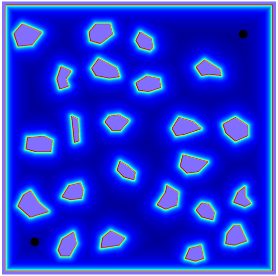
- Combined attractive/repulsive potential

$$U(q) = U_{att}(q) + U_{rep}(q)$$

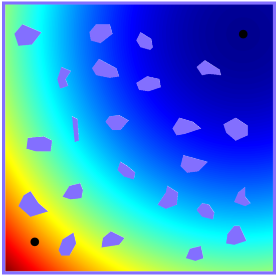
- Goal is reached by following negative gradient $-\nabla U(q)$
- Gradient-descend method

☛ Y. K. Hwang and N. Ahuja, A potential field approach to path planning, IEEE Transaction on Robotics and Automation, 8(1), 1992.

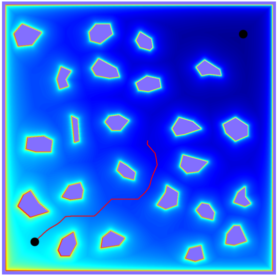
Potential field: parameters



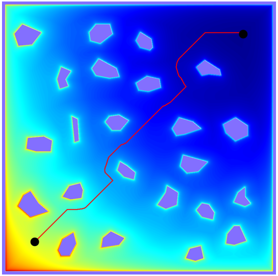
$K_{att} = 0$, no attraction



$K_{att} \gg K_{rep}$, no repulsion

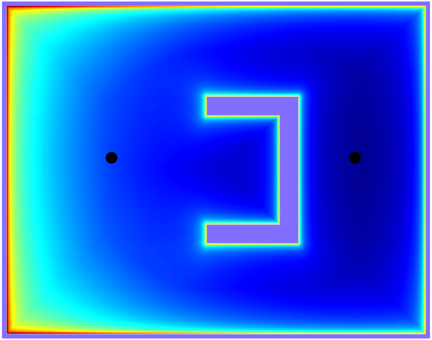


$K_{att} \sim K_{rep}$

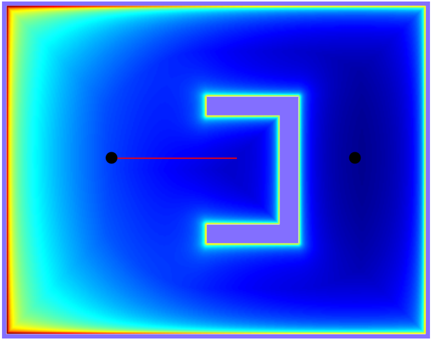


optimal settings

- Potential field may have more local minima/maxima
- Gradient-descent stucks there



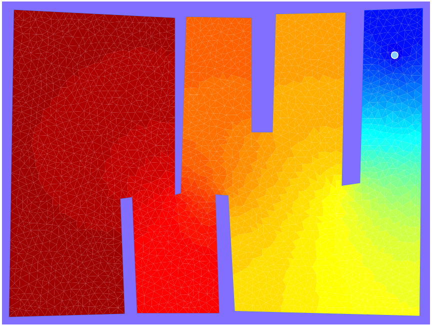
potential field



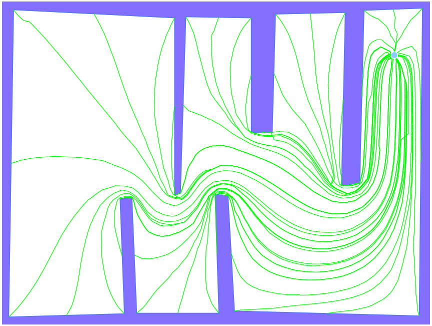
gradient-descent to minimum

- Escape using random walks
- Use a better potential function without multiple local minima — harmonic field

- Harmonic field is an ideal potential function: only one extrem



Harmonic field



Paths from various q_{init}

Images by J. Mačák, Multi-robotic cooperative inspection, Master thesis, 2009

Potential field: summary

- Usually computed using grid or a triangulation of the $\mathcal{W}$
- Suitable for 2D/3D $\mathcal{C}$-space
  - memory requirements (in case of grid-based computation)
  - requires to compute distance *d* to the nearest obstacle in $\mathcal{C}$!
- Parameters K_{att}, K_{rep} and ϱ need to be tuned
- Problem with local minima $\rightarrow$ hamornic fields

But how to really find the path?

So far we know ...

- Visibility graphs, Voronoi diagrams, Decomposition-based planners
- Navigation functions & Potential fields

What they do?

- Discretize workspace/$\mathcal{C}$-space by "converting" it to a graph structure
- The graph is also called **roadmap**
- The roadmap is a "discrete image" of the continuous $\mathcal{C}$-space
- The path is then found as path in the graph

Graph-search

- Breath-first search
- Dijkstra
- A*, D* (and their variants)

Motion planning

Continuous problem

$\mathcal{C}$-space specification, q_{init}, q_{goal}

Discretization of $\mathcal{C}$

Explicit construction random/deterministic sampling of $\mathcal{C}$

Graph search

Dijkstra, A*, D*, ...

Graph search: Dijkstra's algorithm

- Finds shortest path from $s \in V$ (source) to all nodes
- dist(v) is the distance traveled from the source to the node s; prev(v) denotes the predecessor of node v

```
Q = ∅
for v ∈ V do
    prev[v] = -1            // predecessor of v
    dist[v] = ∞                // distance to v
dist[s] = 0
add all v ∈ V to Q
while Q is not empty do
    u = vertex from Q with min dist[u]
    remove u from Q
    foreach neighbor v of u do
        dv = dist[u] + d_{u,v}
        if dv < dist[v] then
            dist[v] = dv
            prev[v] = u
```

- Path from $v \to s$: $\quad v, pred[v], pred[pred[v]], \ldots s$

☛ Dijkstra, E. W. "A note on two problems in connection with graphs." Numerische mathematik 1.1 (1959): 269-271.

Completeness and optimality

Completeness

- Algorithm is complete, if for any input it correctly reports in finite time if there is a solution or no.
- If a solution exists, it **must** return one in a finite time
- Computationally very hard
- Complete methods exist only for low-dimensional problems

Probabilistic completeness

- Algorithm is prob. complete if for scenarios with existing solution the probability of finding that solution converges to one.
- If solution does not exists, the method can run forever

Optimal vs. non-optimal

- Optimal planning: algorithm ensures finding of the optimal solution (according to a criterion)
- Non-optimal: any solution is returned

Completeness and optimality

Visibility graph

- Complete and optimal

Voronoi diagram, decomposition-based method

- Complete, non-optimal

Navigation function

- Complete
- Optimal for Wavefront/Dijkstra/-based navigation functions

Potential field

- Complete only if harmonic field is used (one local minima!)

Consider the limits of these methods!

- Point/Disc robots, low-dimensional $\mathcal{C}$-space

☛ E. Rimon and D. Koditschek. "Exact robot navigation using articial potential functions." IEEE Transactions on Robotics and Automation, 1992.

Do we always need optimal solution?

- No! in many cases, non-optimal solution is fine
  - e.g. for assembly/disassembly studies, computational biology
  - generally: if the **existence of a solution** is enough for subsequent decisions
- in industry:
  - scenarios, where robot waits due to mandatory technological breaks
  - e.g., in robotic welding and painting

When to prefer optimal one?

- Repetitive executing of the same plan
- Benchmarking of algorithms

It is necessary to carefully design the criteria!

Shortest path vs. fastest path vs. path for good spraying

- Motion planning: how to move objects and avoid obstacles
- Configuration space $\mathcal{C}$
- Generally, planning leads to search in continuous $\mathcal{C}$
- But we (generally) don't have explicit representation of $\mathcal{C}$
- We have to first create a discrete representation of $\mathcal{C}$
- and search it by graph-search methods
- Special cases: point robot and 2D/3D worlds
  - Explicit representation of $\mathcal{W}$ is also rep. of $\mathcal{C}$
  - Geometric planning methods: Visibility graph, Voronoi diagram, decomposition-based
  - Also navigation functions + potential field

Motion planning

Continuous problem

$\mathcal{C}$-space specification,
q_{init}, q_{goal}

Discretization of $\mathcal{C}$

Explicit construction
random/deterministic
sampling of $\mathcal{C}$

Graph search

Dijkstra, A*, D*, . . .