Motion Planning for Dynamic Environments

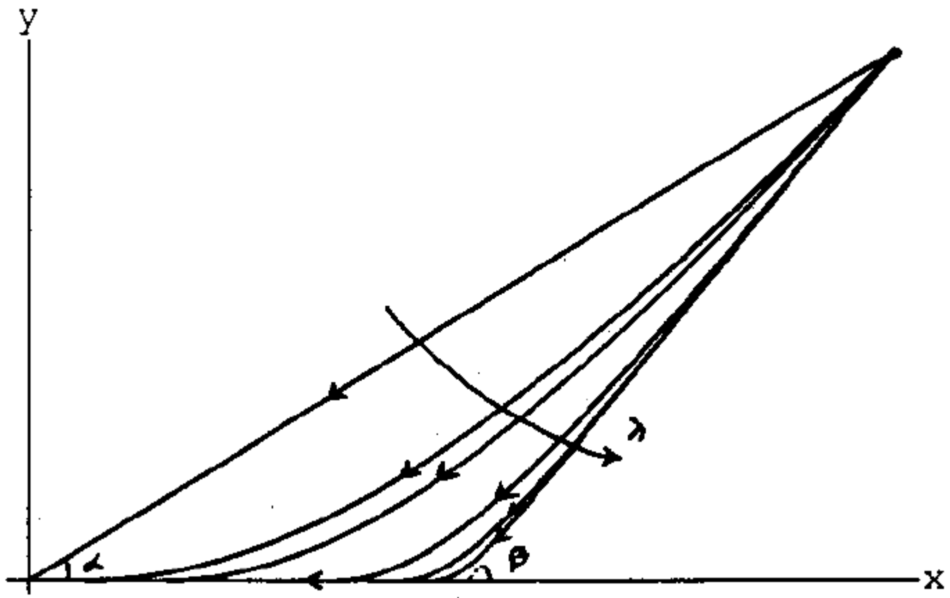
Part IV - Dynamic Environments: Methods

Steven M. LaValle
University of Illinois

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Solution to Homework 3

The chicken follows an interesting curve, depending on λ.



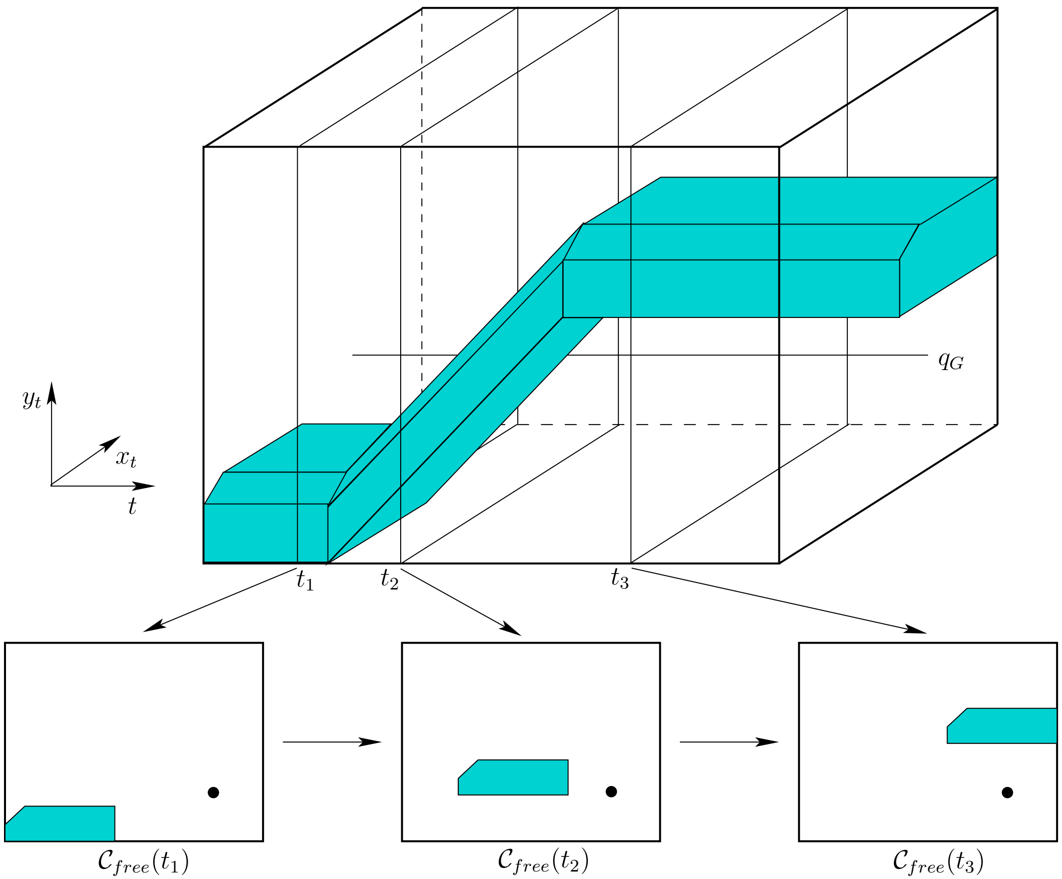
Rajeev Sharma, IEEE TRA 1992

Families:

- Completely predictable environments
- Sensor feedback and collision avoidance
- Planning under bounded motion uncertainty
- Dynamic programming over cost maps
- Information spaces that tolerate uncertainty

Completely predictable

Recall Configuration-Time Space



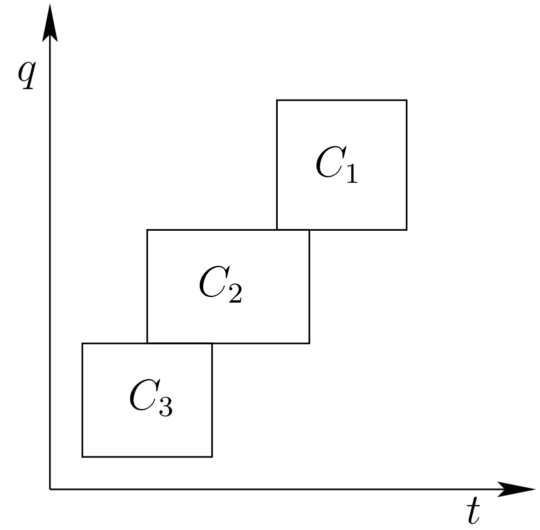
At each time slice $t \in T$, we must avoid

$$\mathcal{C}_{obs}(t) = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O}(t) \neq \emptyset\}$$

Solutions for Completely Predictable Environments

- Sampling-based methods
- Combinatorial methods
- Handling robot speed bounds
- Velocity tuning method

Transitivity issue:



In ordinary path planning, if C_1 and C_2 are adjacent and C_2 and C_3 adjacent, then a path exists from C_1 to C_3.

However, for dynamic environments it might require time travel.

Extending Sampling-Based Methods

Most approaches depend on a metric $\rho : \mathcal{C} \times \mathcal{C} \to [0, \infty)$.
Must extend into X to ensure that time only increases.

To extend across $Z = \mathcal{C} \times T$:

$$\rho_Z(x, x') = \begin{cases} 0 & \text{if } q = q' \\ \infty & \text{if } q \neq q' \text{ and } t' \leq t \\ \rho(q, q') & \text{otherwise.} \end{cases}$$

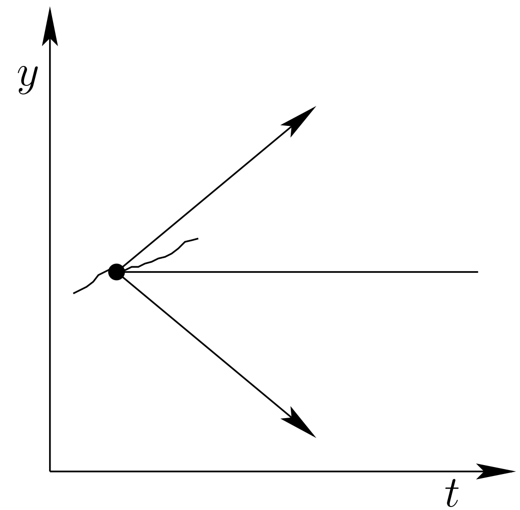
- Sampling-based RRTs extend across Z using ρ_Z.
  Bidirectional is a bit more complicated.
- Sampling-based roadmaps (including PRMs) extend to produce directed roadmaps.

Bounded Speed

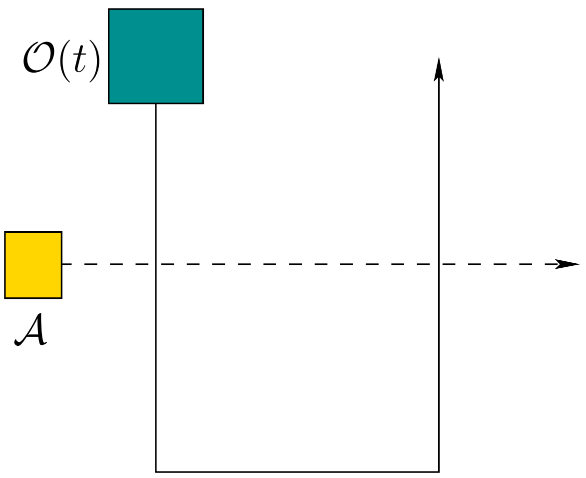
Robot velocity: $v = (\dot{x}, \dot{y})$
Speed bound: $|v| \leq b$ for some constant $b > 0$

The velocity v at every point in Z must point within a cone at all times:

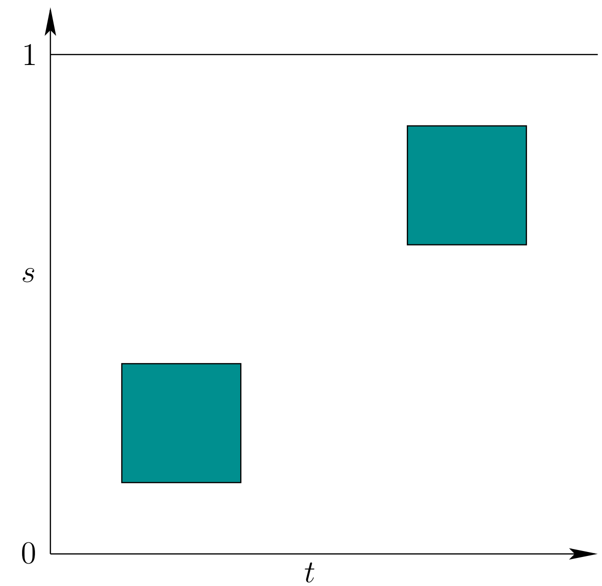
$$\big(x(t+\Delta t) - x(t)\big)^2 + \big(y(t+\Delta t) - y(t)\big)^2 \leq b^2(\Delta t)^2.$$



Warning: PSPACE-hard in general.



Workspace



State space

Compute a collision-free path: $\tau : [0,1] \rightarrow \mathcal{C}_{free}$.

Design a *timing function* (or *time scaling*): $\sigma : T \rightarrow [0,1]$.

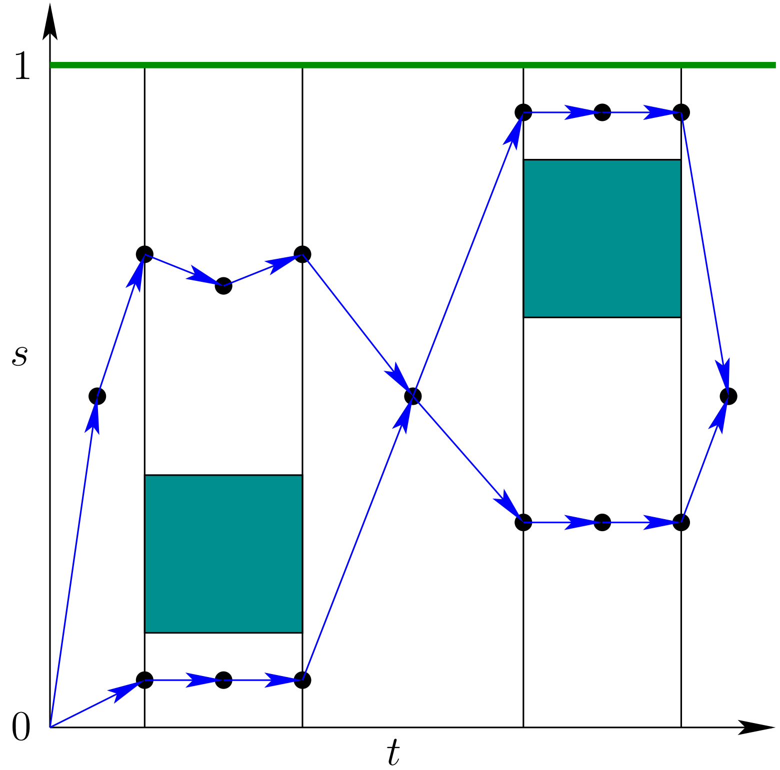
This produces a composition $\phi = \tau \circ \sigma$, which maps from T to $\mathcal{C}_{free}$ via $[0,1]$.

Velocity Tuning

Because it is a 2D problem, many methods can be used.

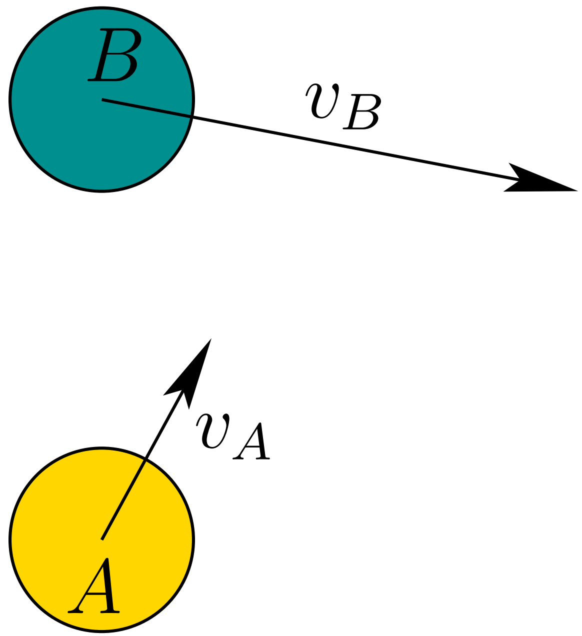
Simple grid search: BFS, DFS, Dijkstra, A^*, ...

It is more elegant and efficient to use combinatorial methods.

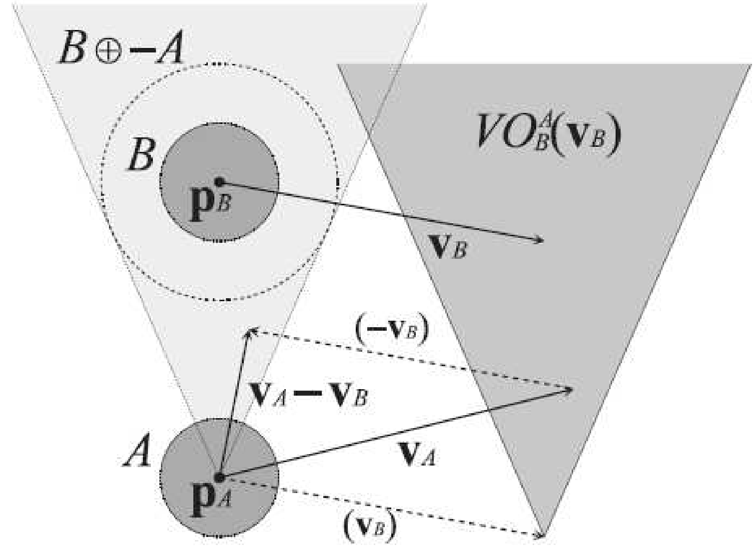


For example, trapezoidal decomposition.

Sensor Feedback



- Two rigid bodies A and B moving in $\mathbb{R}^2$.
- They have constant velocities v_A and v_B.
- If v_B is constant, what values of v_A cause collision?



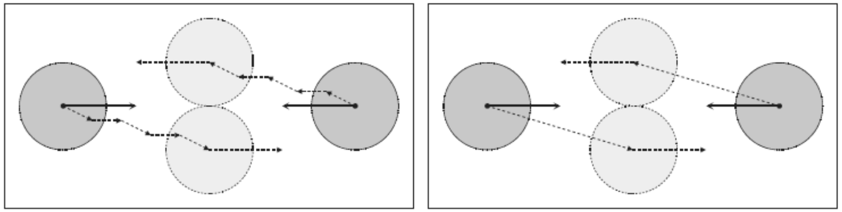
$$\lambda(p, v) = \{p + tv \mid t \geq 0\}$$

$$VO_B^A(v_B) = \{v_A \mid \lambda(p_A, v_A - v_B) \cap \mathcal{C}_{obs} \neq \emptyset\}$$

Here, $\mathcal{C}_{obs} = B \ominus A$ (Minkowski difference).

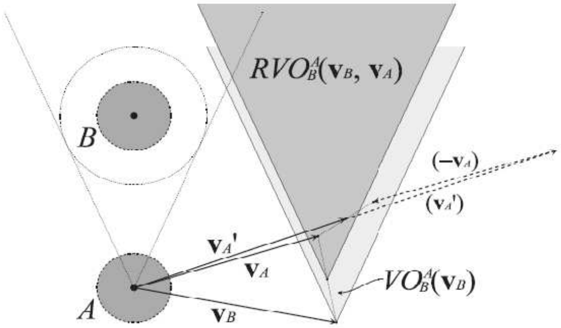
Fiorini, Shiller, 1998.

What is both bodies react? Oscillation possible.



Suppose that all bodies follow the same strategy.

This can be taken into account for a great advantage.



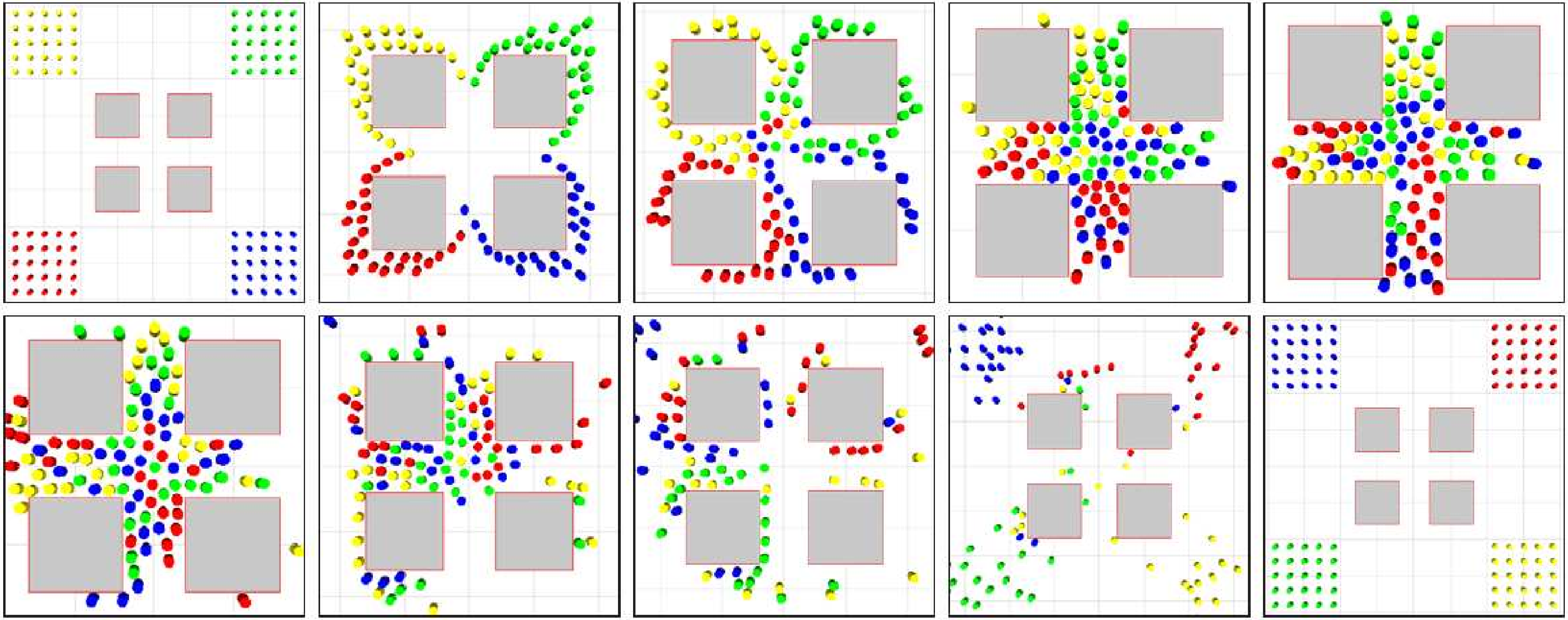
$$RVO_B^A(v_B) = \{v'_A \mid 2v'_A - v_A \in VO_b^A(v_B)\}$$

Choose v'_A as the average of its current velocity and a velocity that lies outside the velocity obstacle.

van den Berg, Lin, Manocha, 2008

Recriprocal Velocity Obstacles

A computed result:



Try it at the next ICRA coffee break...

Other Sensor Feedback Strategies

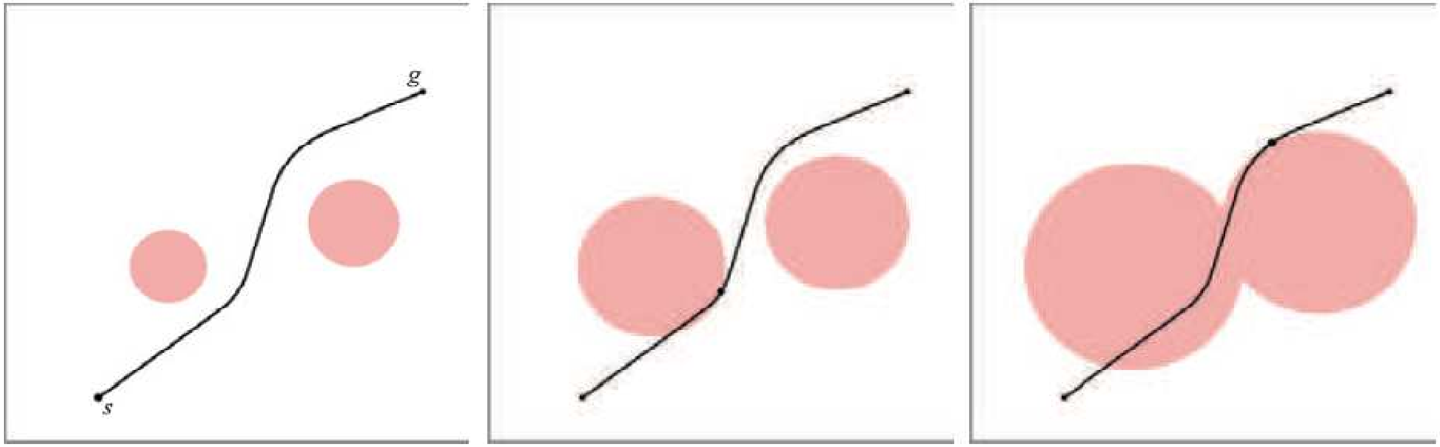
- Potential fields, Khatib, 1980
- Vector field histogram, Borenstein, Koren, 1991
- Dynamic window approach, Fox, Burgard, Thrun, 1997
- Nearness diagram, Minguez, Montano, 2004

Many more...

Bounded Uncertainty

Time-Minimal Trajectories

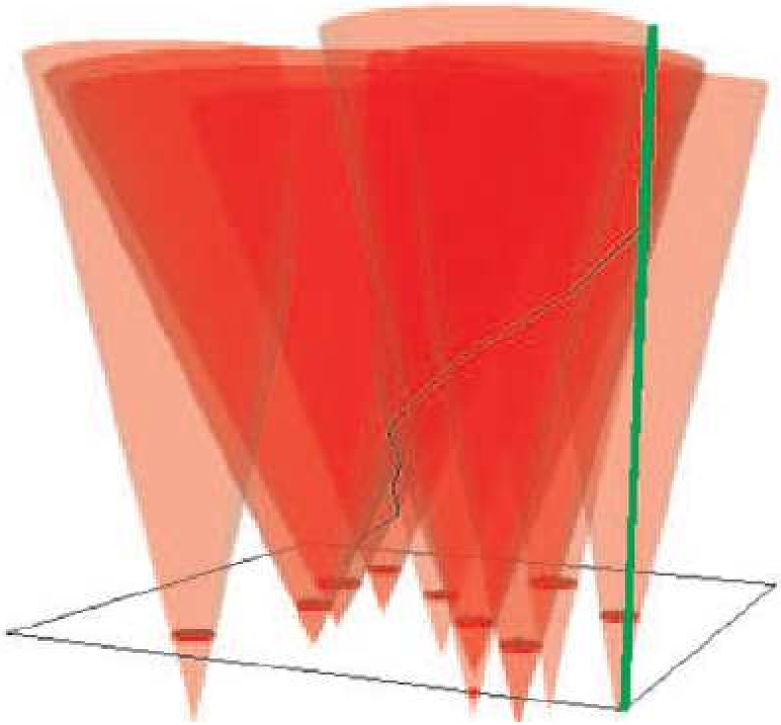
- Point (or disc) robot moves at constant speed.
- A finite set of point (or disc) obstacles.
- Obstacles have omnidirectional speed bound.
- Problem: Compute time-optimal collision-free trajectory.



van den Berg, Overmars, 2008

Time-Minimal Trajectories

A computed example, shown through configuration-time space:



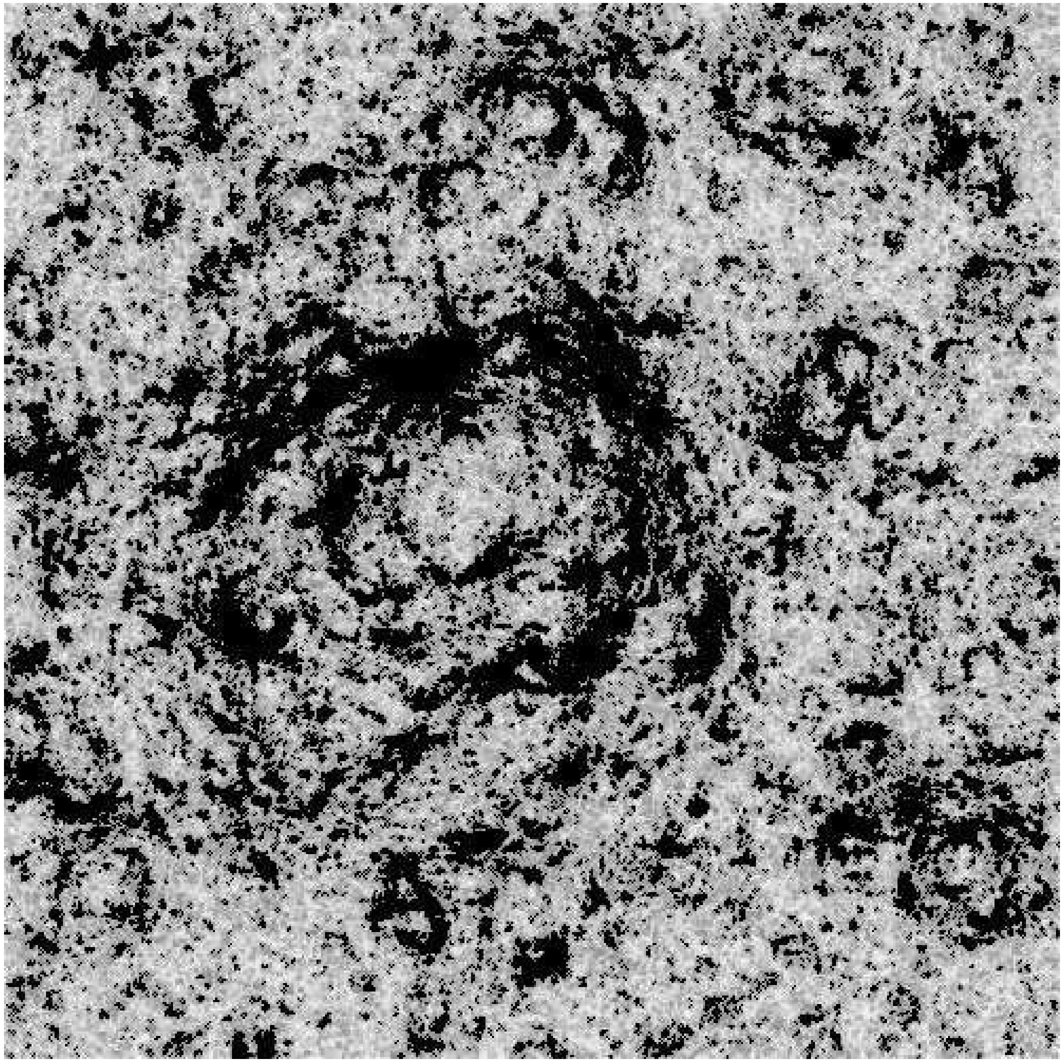
Can solve problems $O(n^3 \lg n)$ time.

It is related to shortest-path graphs in the plane (bitangents).

Recently improved to $O(n^2 \lg n)$ by Maheshwari et al.

Dynamic Programming

Cost Maps



Instead of a crisp $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$, a *cost* could be associated with each q (or each neighborhood).

Value Iteration Methods

Let X be any state space.

We can make a state-time space by $Z = X \times T$.

Let U be an action set.

There are $K+1$ stages $(1, 2, \ldots, K+1)$ along the time axis.

Let $x' = f(x, u)$ be a state transition equation.

Let L denote a stage-additive *cost functional*,

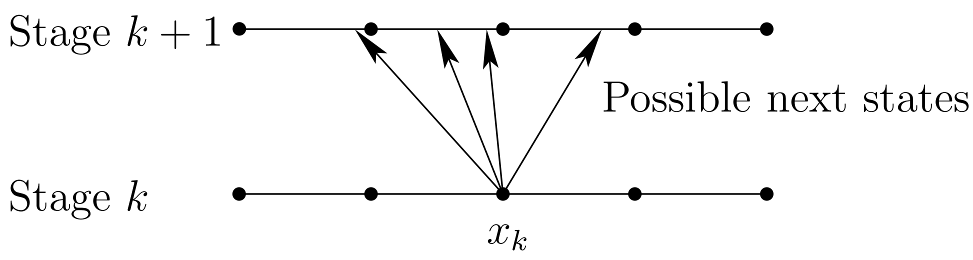
$$L = \sum_{k=1}^{K} l(x_k, u_k) + l_{K+1}(x_{K+1}).$$

The task or goal can be expressed in terms of L.

Value Iteration Methods

A feedback plan is represented as $\pi : X \to U$

Let $G^*_k(x_k)$ denote the *optimal cost to go* from x_k at stage k (optimized over all possible π).



Bellman's dynamic programming equation:

$$G^*_k(x_k) = \min_{u_k \in U(x_k)} \left\{ l(x_k, u_k) + G^*_{k+1}(x_{k+1}) \right\}$$

Bellman's dynamic programming equation:

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \Big\{ l(x_k, u_k) + G_{k+1}^*(x_{k+1}) \Big\}.$$

Algorithm:

- Initially, G_{K+1}^* is known (from $l_{K+1}(x_{K+1})$).
- Compute G_K^* from G_{K+1}^*.
- Compute G_{K-1}^* from G_K^*.
- $\vdots$

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Value Iteration Methods

But X and U are usually continuous spaces.

A finite subset of U can be sampled in Bellman's equation.

Interpolation (this is the 1D case) over X:

$$G^*_{k+1}(x) \approx \alpha G^*_{k+1}(s_i) + (1-\alpha) G^*_{k+1}(s_{i+1})$$

Value Iteration Methods

Stochastic version not difficult.

Let $p(x_{k+1}|x_k, u_k)$ be a probabilistic state transition equation.

Bellman's equation becomes:

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \left\{ l(x_k, u_k) + \sum_{x_{k+1}} G_{k+1}^*(x_{k+1}) p(x_{k+1}|x_k, u_k) \right\}.$$

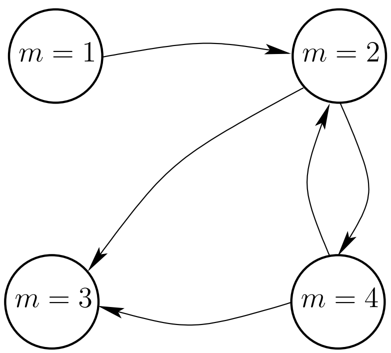
Optimizes the *expected* cost-to-go.

In the stationary case, there are Dijkstra-like versions.

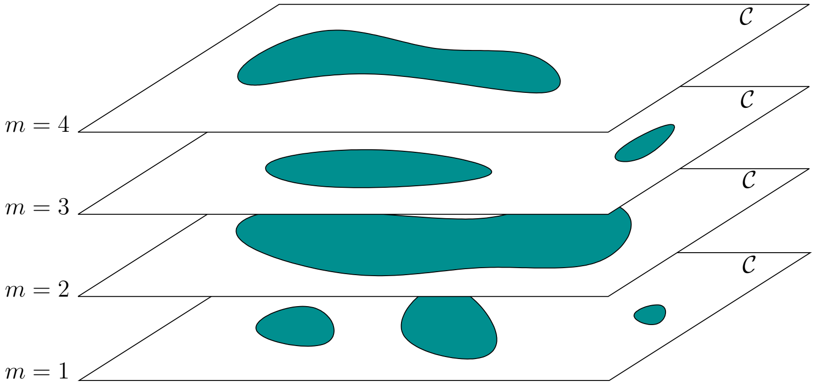
See *Planning Algorithms*: Sections 2.3.2, 8.5.5, 10.6

Applying to Dynamic Environments

Recall the hybrid system formulation.



Modes

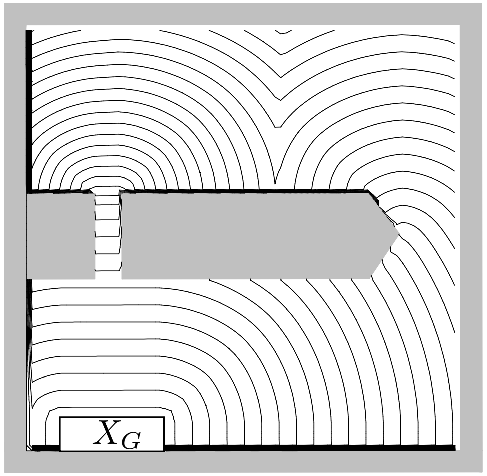


Layers

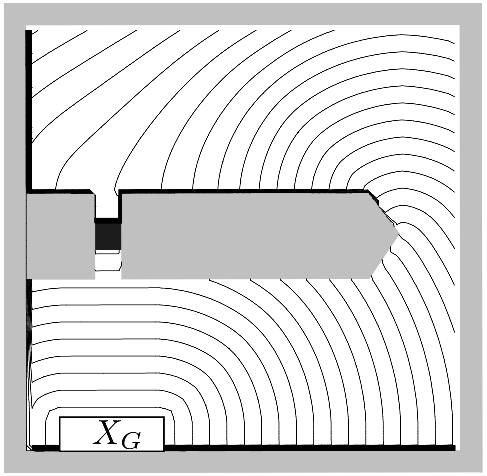
Doors may open or close according to a Markov chain.

Applying to Dynamic Environments

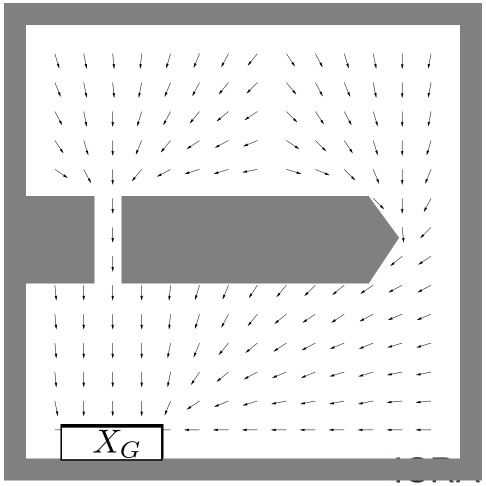
The optimal cost-to-go and feedback plan.

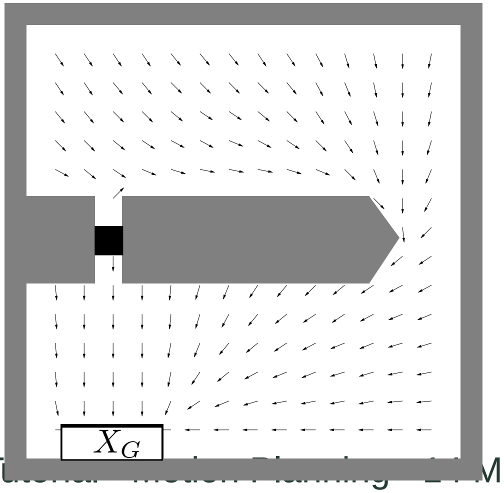


Cost-to-go, open mode

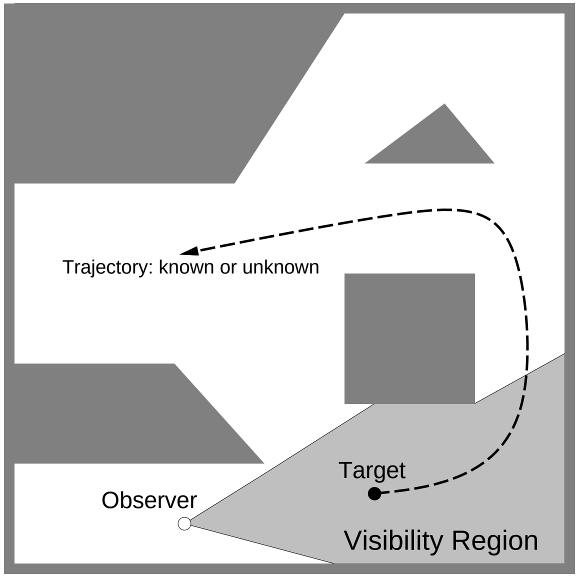


Cost-to-go, closed mode





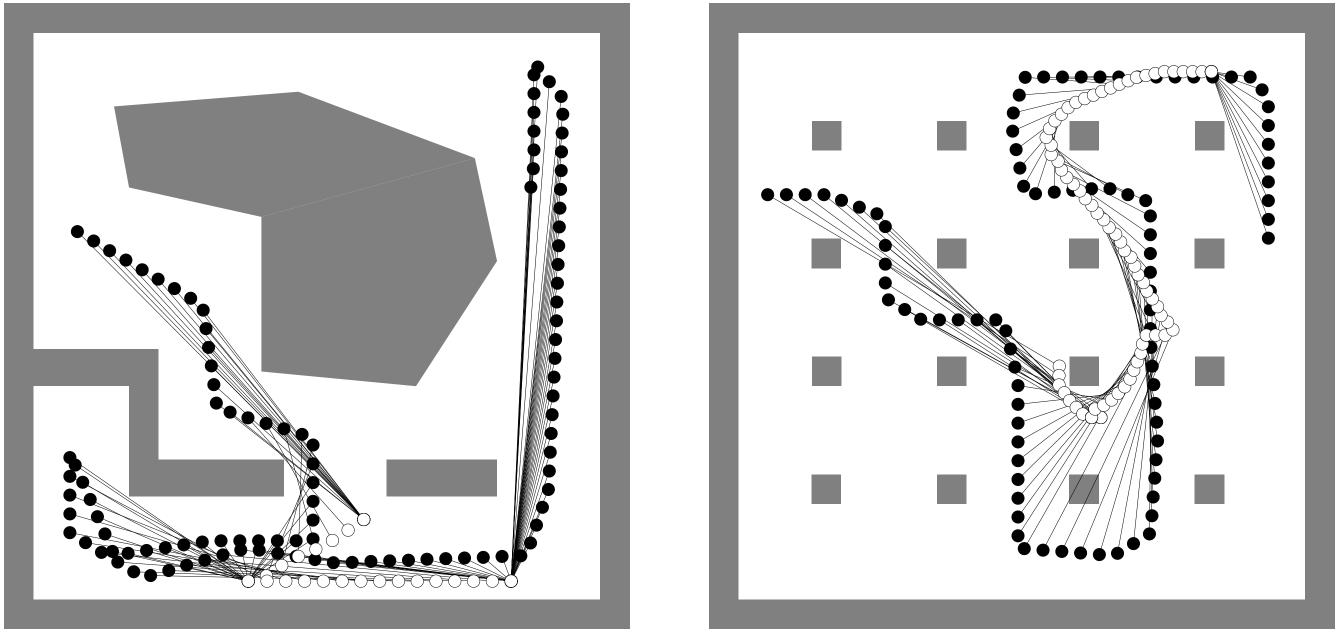
Maintaining Visibility



- A robot must follow a moving target with a camera.
- How to move the robot to maintain visibility as much as possible?
- Optimize the total robot motion.
- Predictable and partially predictable target cases

LaValle, Gonzalez-Banos, Becker, Latombe, 1997

Optimal robot trajectories computed using value iteration:



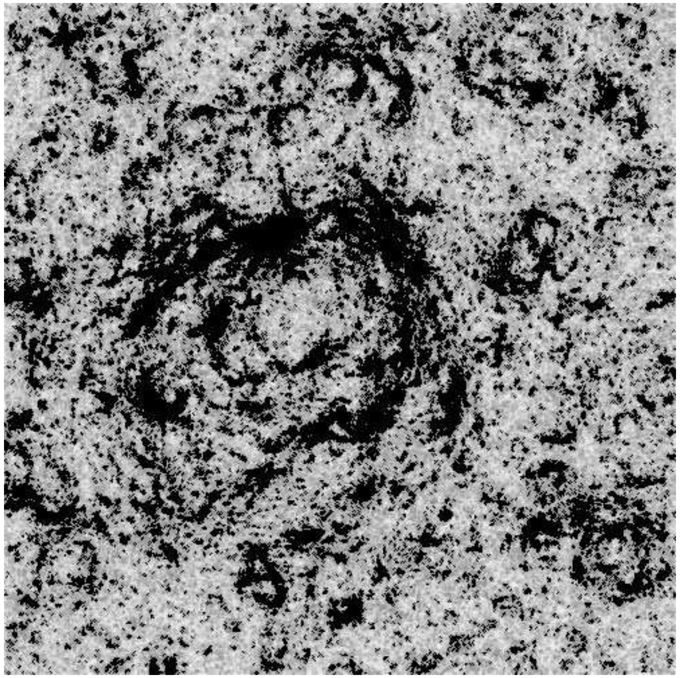
For unpredictable target, move robot to maximize the target's minimum time to escape.

D-Star

D-Star: Stentz, 1994

D-Star Lite: Koenig, Likhachev, 2002

Consider A^* search on a weighted grid graph.



Execution of the plan causes new information to be learned.

Enhance A^* to allow edge costs to increase or decrease.

Let

$$rhs(q) = \min_{q' \in Succ(q)} \{c(q, q') + g(q')\}$$

For the optimal cost-to-go function, Bellman's equation should be satisfied everywhere:

$$g(q) = rhs(q)$$

(Also, $g(q_G) = 0$.)

If it is not, then fix it!

D-Star

Let $h(q, q')$ be a *heuristic* underestimate of the optimal cost from q to q'.
Keep search queue sorted by key value:

$$\min(g(q), rhs(s)) + h(q_I, s)$$

If vertices have equal key value, then select one with smallest $\min(g(q), rhs(s))$.

When edges costs change, affected nodes are placed on the search queue.
Iterations continue until all affected nodes are fixed, and Bellman is happy again.

Dynamic Replanning

Sliding Window

Consider the following loop:

1. Plan a sequence of actions
2. Take the first action
3. Receive new information from sensors
4. Go to 1

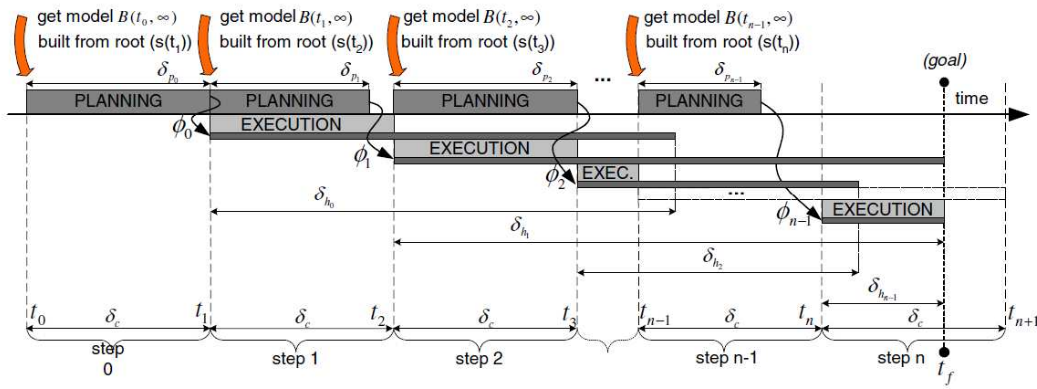
Sliding Window

That was the usual *sense-plan-act* loop.

Related ideas:

- Receding horizon control
- Model predictive control
- Dynamic replanning
- Partial motion planning
- Anytime planning

Partial Motion Planning

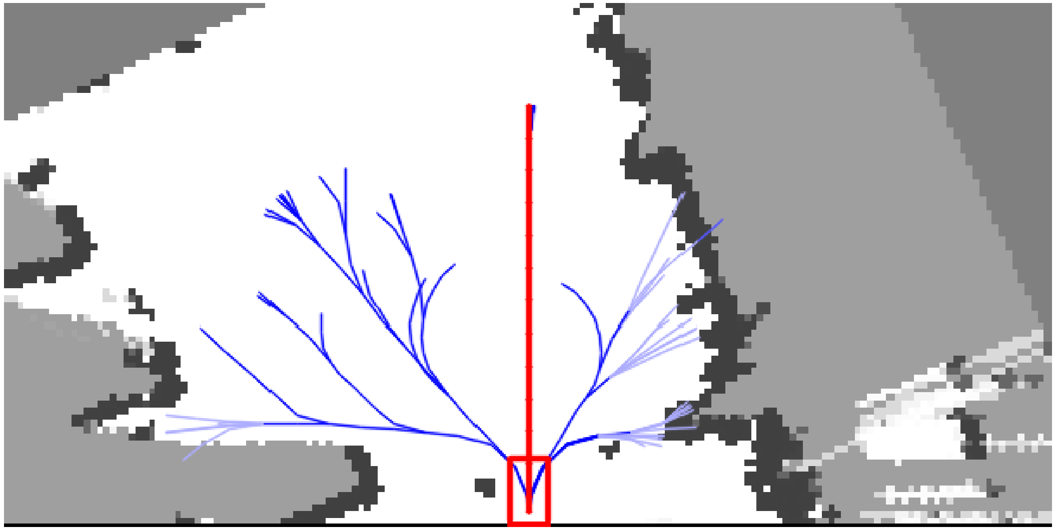


- Construct a partial plan toward the goal within allotted time.
- Compute X_{ric} (inevitable collision states).
- Ensure that paths are safe by avoiding X_{ric}.
- While executing, construct the next partial plan.

Fraichard, Asama, 2004; Petti, Fraichard, 2005

Probabilistic RRTs

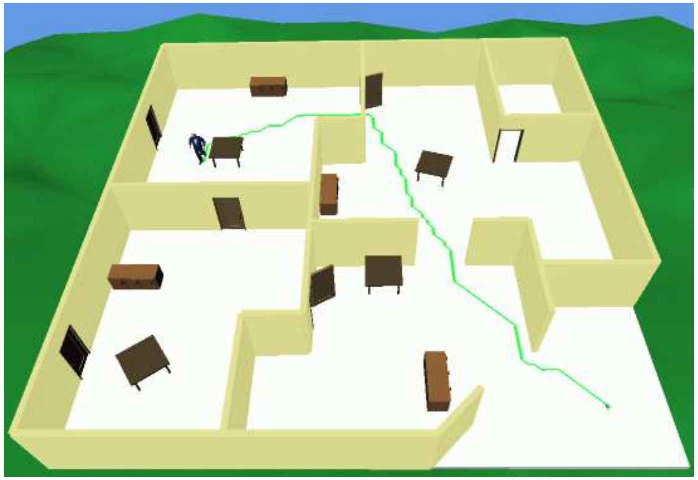
- Use partial planning paradigm.
- Build a probabilistic “cost map” that biases RRT growth into lower collision probabilities.
- Use HMM prediction models learned from other moving bodies.

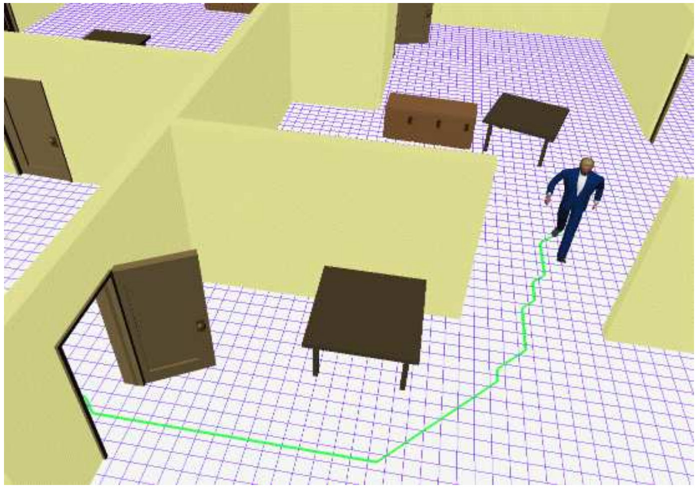


Fulgenzi, Spalanzani, and Laugier, 2009

Replanning From Scratch

Kuffner, 2004





Run A^* or Dijkstra but with reduced neighborhood structure.

Computation times around 10ms.

Replanning

A few other replanning works:

- Leven, Hutchinson, 2002
- Jaillet, Simeon, 2004
- Kallmann, Bargmann, Mataric 2004
- Vannoy, Xiao 2006
- Bekris, Kavraki, 2007
- Nabbe, Hebert, 2007
- Bekris, 2010

Anytime Algorithms

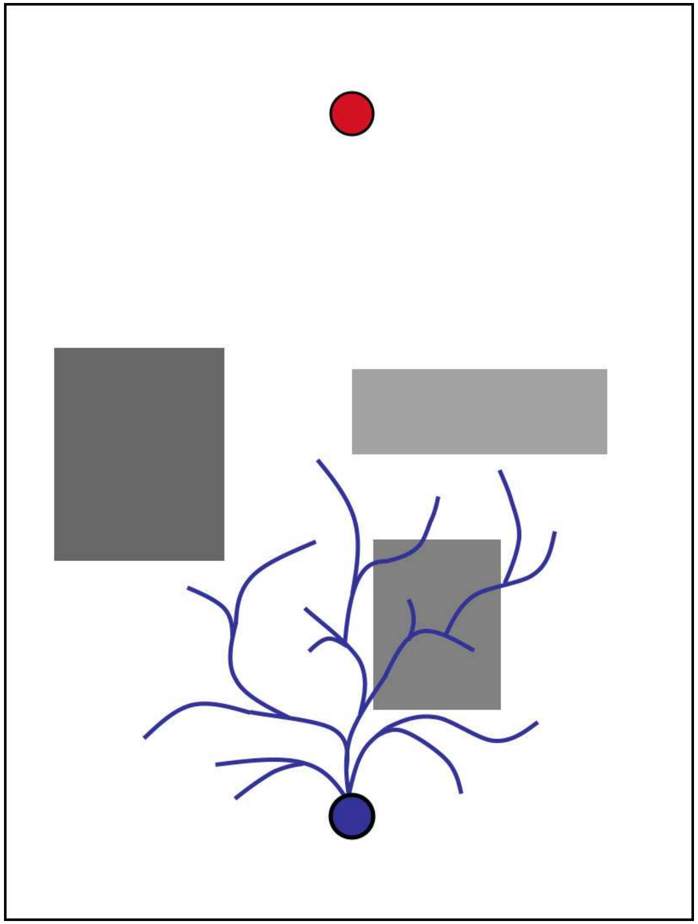
Appearing throughout compute science, an *any-time algorithm* has properties:

- May be terminated at any time
- The solution it produces gradually improves over time

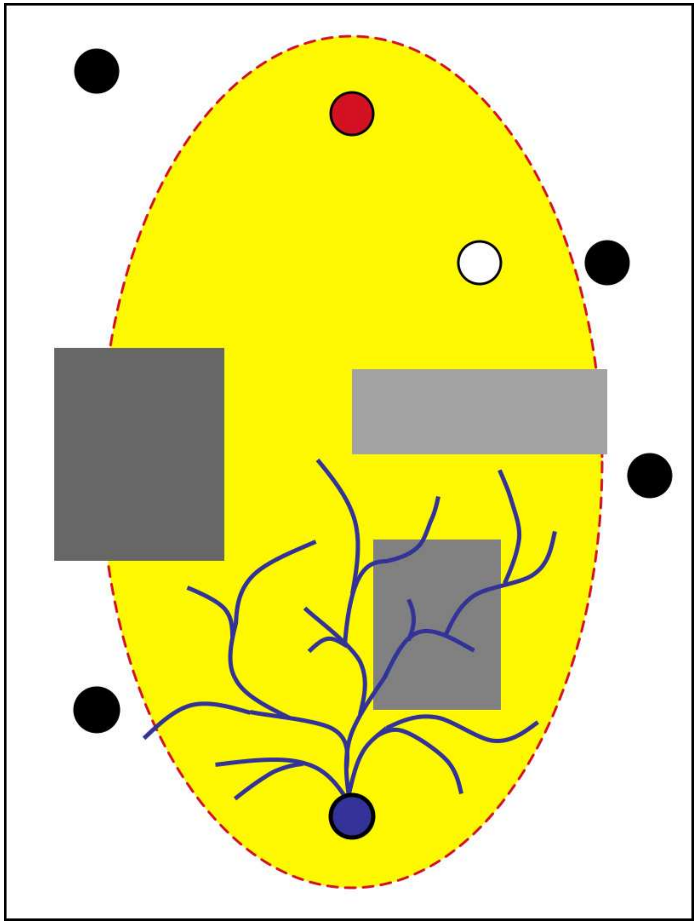
This seems ideally suited for on-line planning and execution.

Anytime RRTs

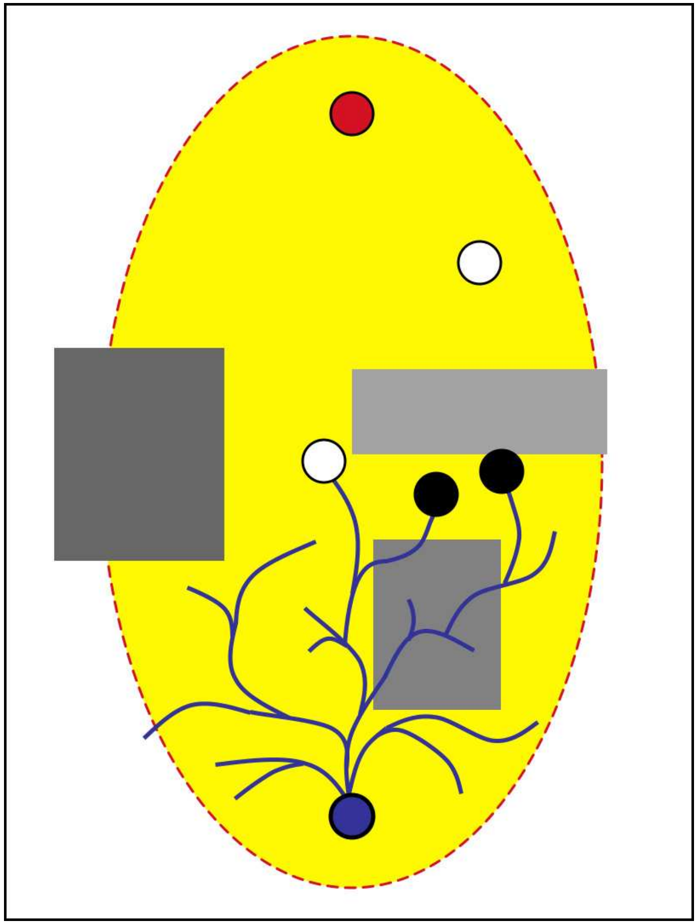
Completely predictable
Sensor Feedback
Bounded Uncertainty
Dynamic Programming
Dynamic Replanning
Information Spaces



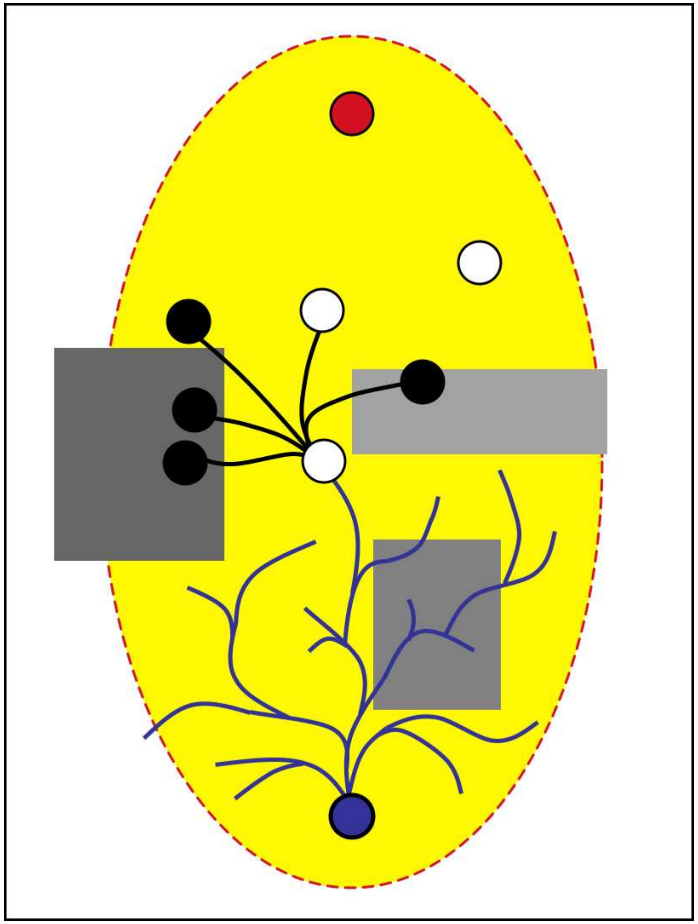
Ferguson, Stentz, 2006



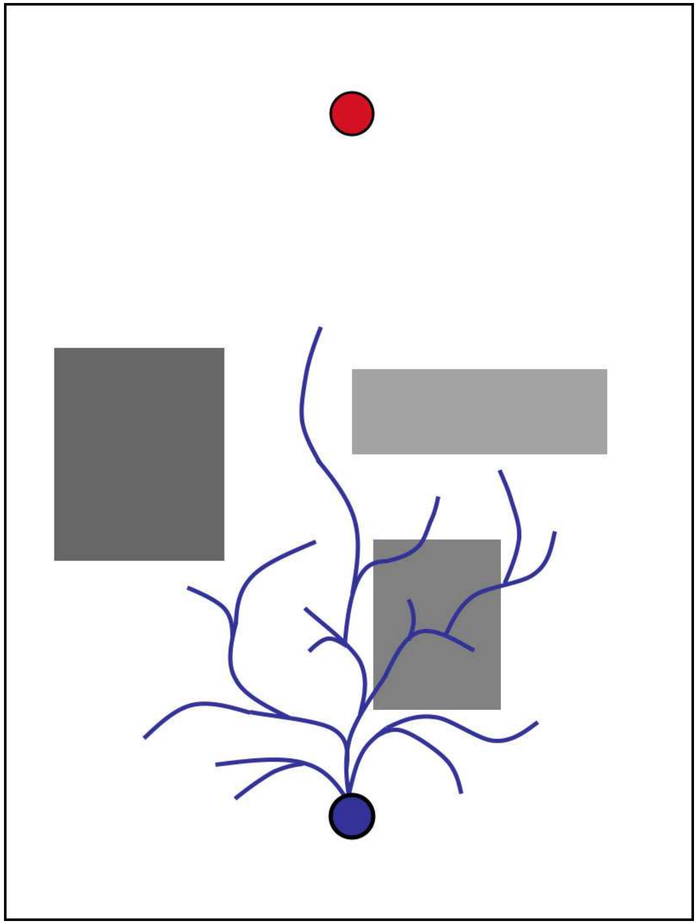
Anytime RRTs



Anytime RRTs



Anytime RRTs

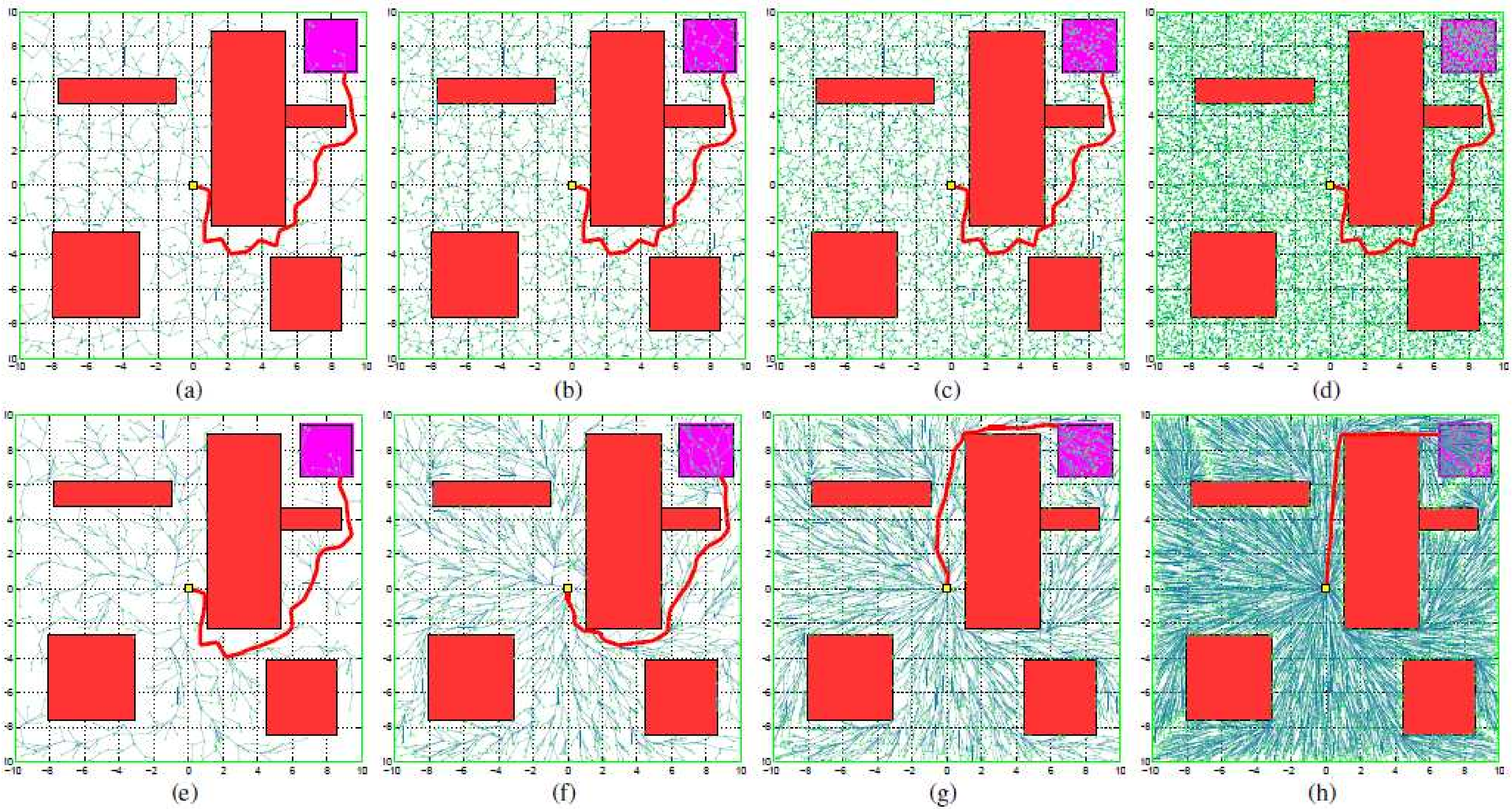


RRT*

- Grow RRT in the usual way
- When a new vertex x_{new} is added, try to connect to other RRT vertices within radius ρ.
- Among all paths to the root from x_{new}, add a new RRT edge only for the shortest one.
- If possible to reduce cost for other vertices within radius ρ by connecting to x_{new}, then disconnect them from their parents and connect them through x_{new}.
- The radius ρ is prescribed through careful percolation theory analysis (related to dispersion).
- RRT* yields asymptotically optimal paths through $\mathcal{C}_{free}$.

Karaman, Frazzoli, IJRR 2011

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN


(a) (b) (c) (d)
(e) (f) (g) (h)

Backwards A*:

- Sort queue by: $g(q) + h(q_I, q)$
- $g(q)$ is the optimal cost-to-come from q_G.
- $h(q_I, q)$ is the guaranteed underestimate of the optimal cost from q_I to q.

Anytime D*

Anytime A*:

- Sort queue by: $g(q) + \gamma h(q_I, q)$
- $\gamma \geq 1$ is an *inflation factor*
- It causes non-optimality, but no worse than a factor of γ.
- Approach: Generate a quick solution for large γ, and then gradually decrease it.

Anytime D*:

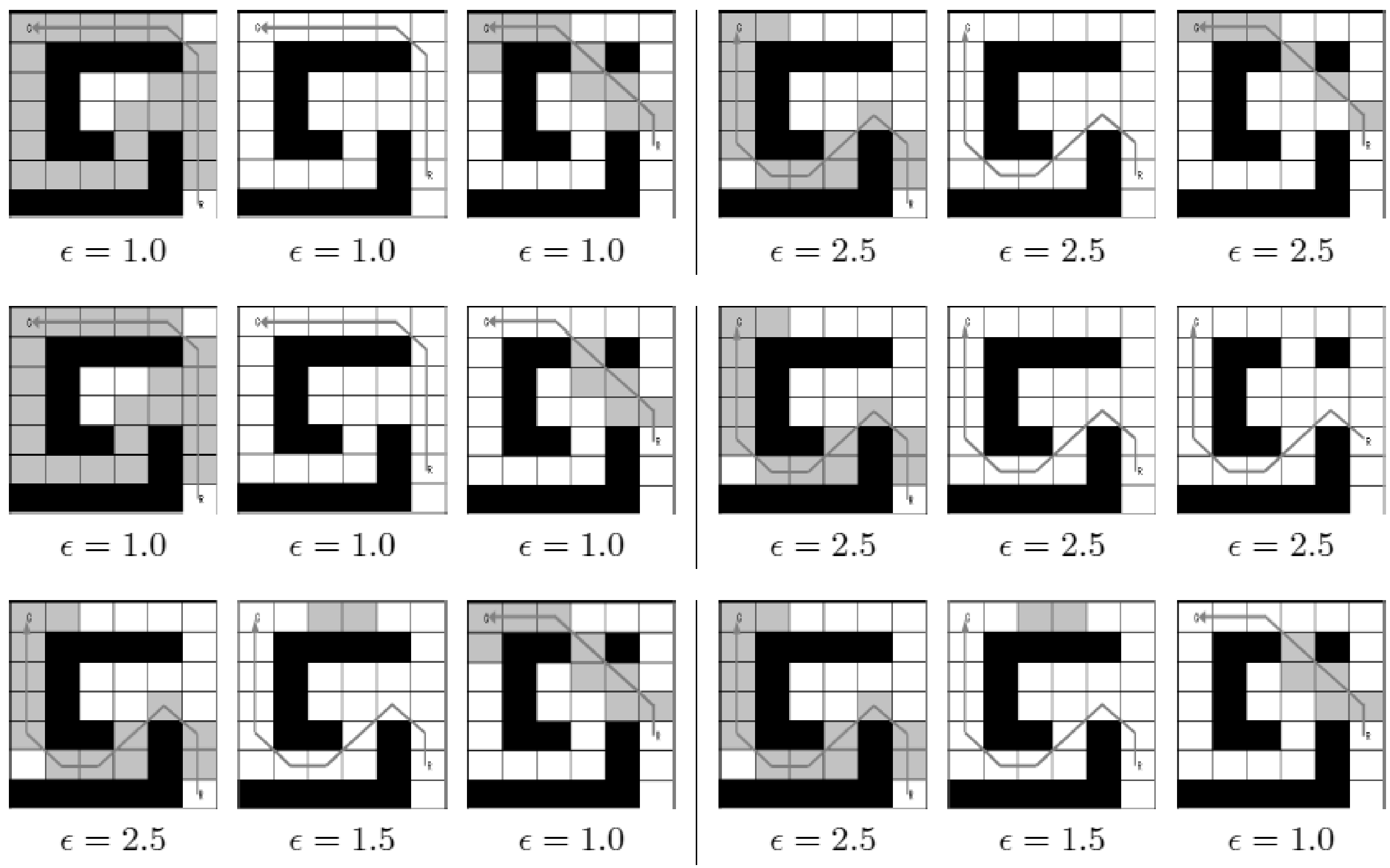
- Use $g(q) + \gamma h(q_I, q)$ in D* lite
- Optimality factor for computed paths remains γ.
- Likhachev, Ferguson, Gordon, Stentz, Thrun, 2005

Example:

left: **A***
right: **A*** with $\epsilon = 2.5$

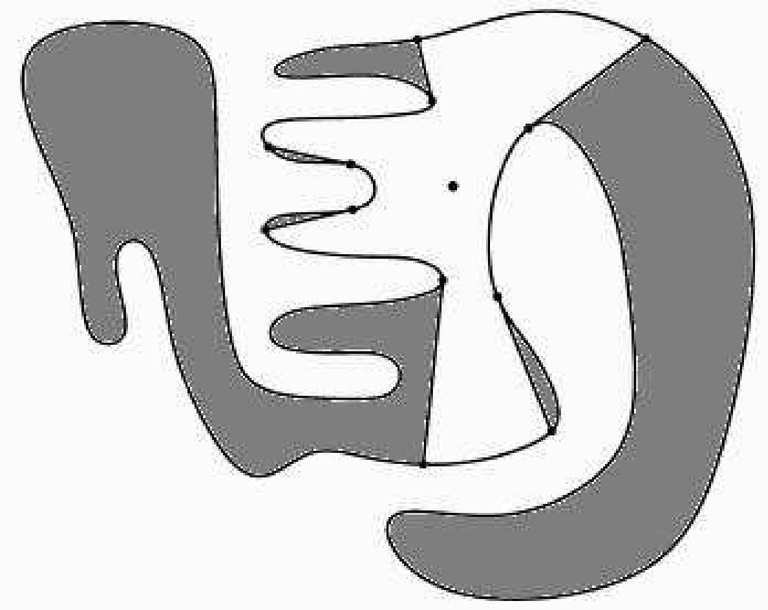
left: **D* Lite**
right: **D* Lite** with $\epsilon = 2.5$

left: **ARA***
right: **Anytime Dynamic A***



Information Spaces

Recall simple model: Evader moves on a continuous path.



An exact cell decomposition method can solve it.

Guibas et al. 1999

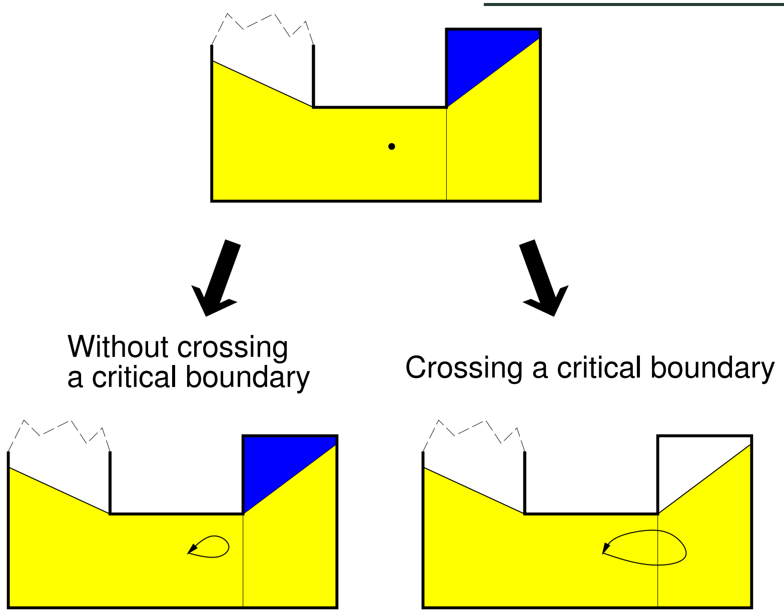
Identify all unique situations that can occur:

An information state is identified by (x, S) in which

x = *the position of the pursuer*

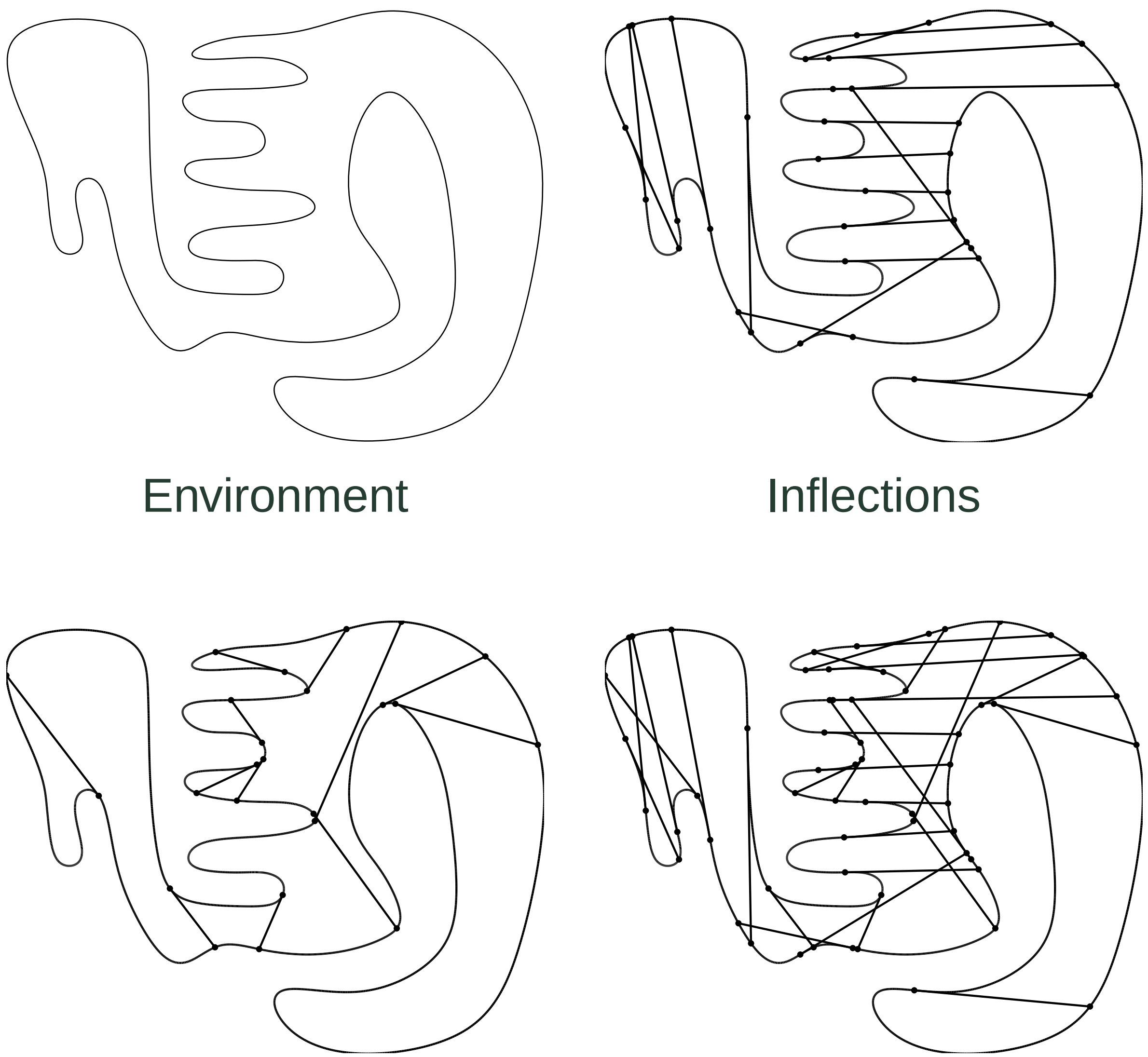
S = *set of possible evader positions*

The set of all information states forms an information space.



Many closed-path motions retain the same information state.

Visibility-Based Pursuit-Evasion



Environment Inflections

Bitangents Cell Decomposition

Imperfect State Information

Two types of imperfect state information:

1. Environment: Obstacles, cost map, moving body configurations
2. Robot: The localization problem

These generally force plan feedback to occur over an *information space*:

$$\pi : \mathcal{I} \to U$$

Imperfect State Information

What does $\iota \in \mathcal{I}$ look like? Possibilities:

- A partial map with robot localized
- A full map with a pdf over robot configurations
- A topological map with robot localized

In the most general setting, we may obtain either a set

$$F(\tilde{u}_{k-1}, \tilde{y}_k) \subseteq X$$

or a pdf

$$p(x_k \mid \tilde{u}_{k-1}, \tilde{y}_k)$$

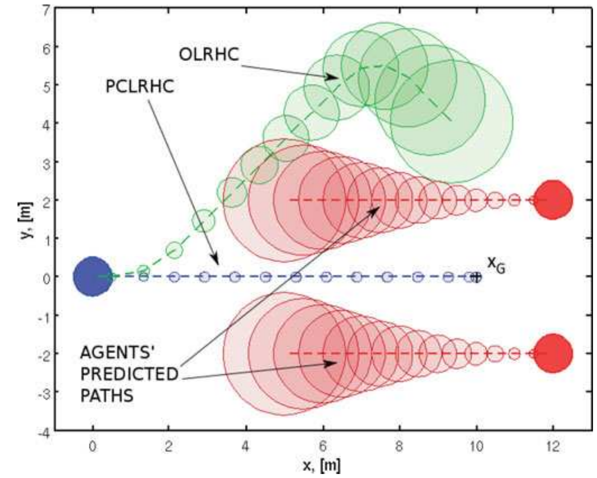
over whatever X state space is needed.

The state $x \in X$ may encode robot configuration, map, other bodies.

Planning in the Probabilistic Information Space

State: $x \in X$ encodes configuration and velocities of robot and bodies.
Stochastic transition law: $p(x_{k+1}|x_k, u_k)$
Disturbed sensor mapping: $p(y_k|x_k)$

- Receding horizon approach
- Partially closed loop: Estimate future sensor readings
- Compute information feedback strategies



DuToit, Burdick, 2012

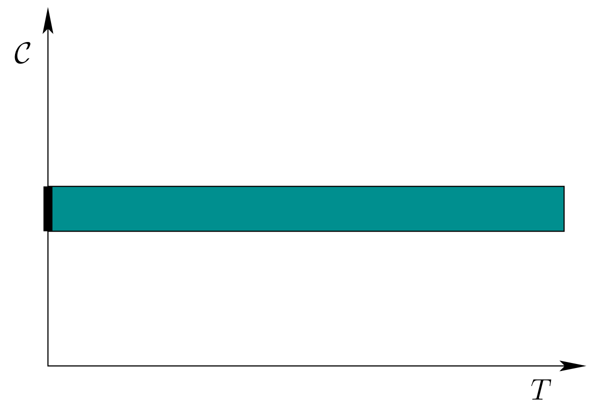
Planning in the Probabilistic Information Space

There are many other approaches to planning in belief space:

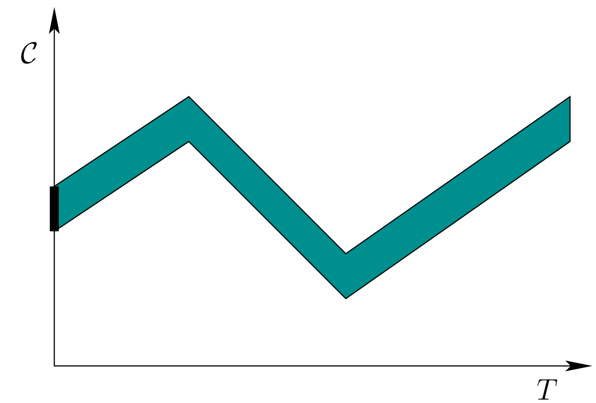
- Roy, Burgard, Fox, Thrun, 1998
- Pineau, Gordon, 2005
- Kurniawati, Hsu, Lee, 2008
- Prentice, Roy, 2009
- Hauser, 2010
- Platt, Kaelbling, Lozano-Perez, Tedrake, 2012

This list is very incomplete...

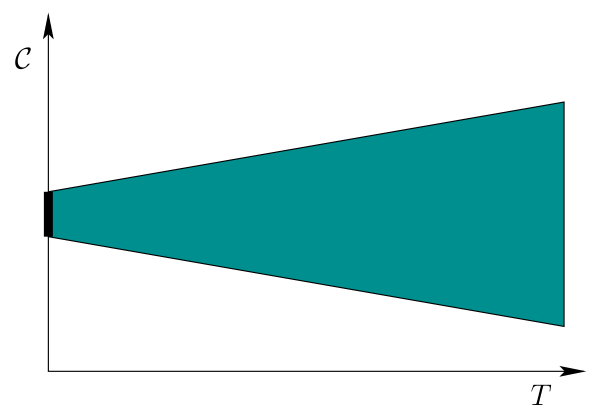
Forward Projections



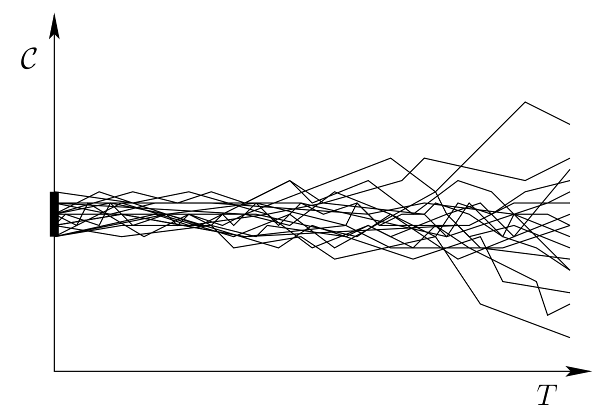
Static



Predictable



Bounded Uncertainty



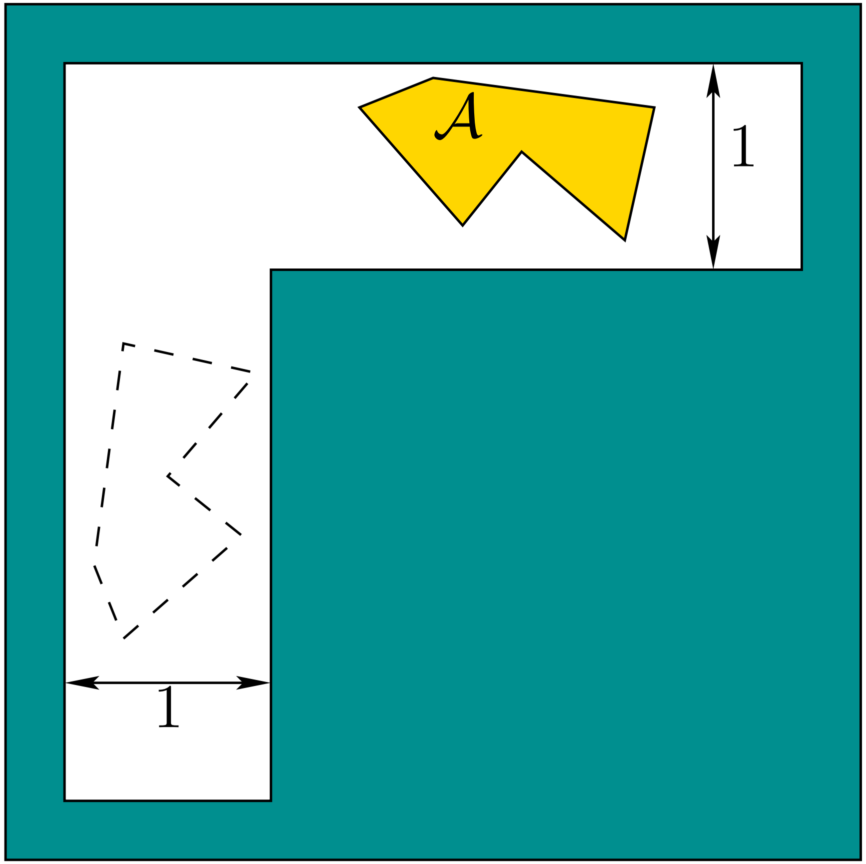
Probabilistic Uncertainty

Summary of Part IV

- Model: predictable, bounded uncertainty, probabilistic
- Sensor feedback vs. dynamic replanning vs. computing optimal strategy
- The power of dynamic programming
- In which information space should the robot live?
- There are NP-hard problems everywhere. We have yet to really understand what makes some problems simpler.
- Which method to use? Need demo, robust experimental system, theoretical guarantees?

Homework 4: Solve During This Century

Let $\mathcal{A}$ be a rigid, polygonal (or semi-algebraic) robot.
Let $\mu(\mathcal{A})$ denote the area of $\mathcal{A}$.



What is the largest robot, in terms of $\mu(\mathcal{A})$, that can fit through the corridor?