EGN 4060c: Introduction to Robotics

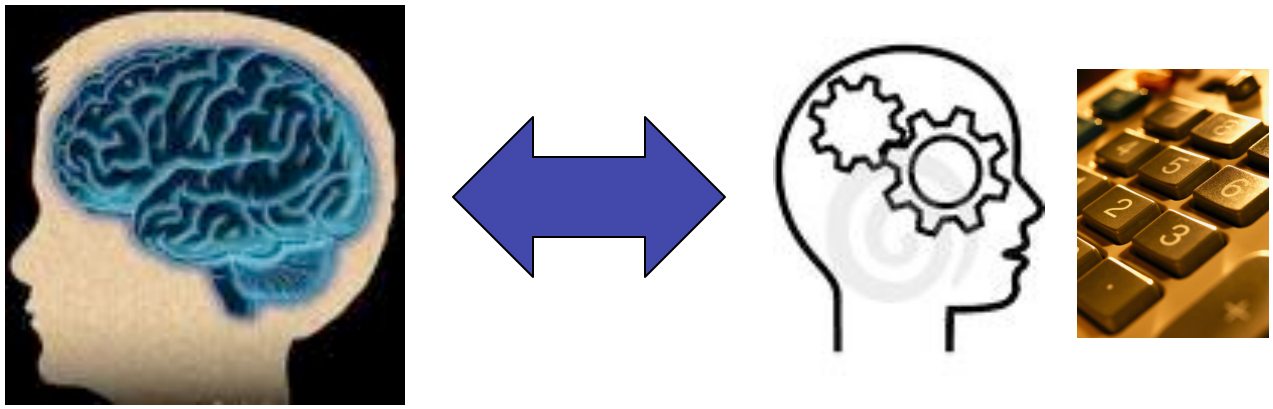
Lecture 8: Planning (Part 2)

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Announcements

- Lab report due Wed
- Sign up for extra lab times on Wed if required
- Next homework on architectures due Sept 24th
- Will put an additional background planning reading on webcourses

Planning is the Basis for Intelligence



•Planning allows the robot to deal with new situations that the programmer might not have anticipated.
•Without being able to search for better combinations of actions, the robot is stuck either following preprogrammed scripts or reacting to the immediate sensory input.

Planning Representation

Different planners use different representations

- Planning over a 2D occupancy-grid using tree-based search
- Planning over sets of boolean propositions (STRIPS)
- Planning for point robots in configuration space
  - Graphs
  - Cellular decompositions

Mission Planning

- Need to come up with high-level goals for a robot in a non-geometric space
- Less emphasis on cost and more emphasis on goal achievement
- Problems:
  - Choosing operators and representation
  - Finding the right level of granularity
  - Interleaving planning and execution

Classical Planning vs. Path Planning

Classical

- Connectivity graph doesn't exist a priori; must be constructed during planning
- Non metric space; difficult to tell how close a node is to the goal state
- Determining completeness of search is harder

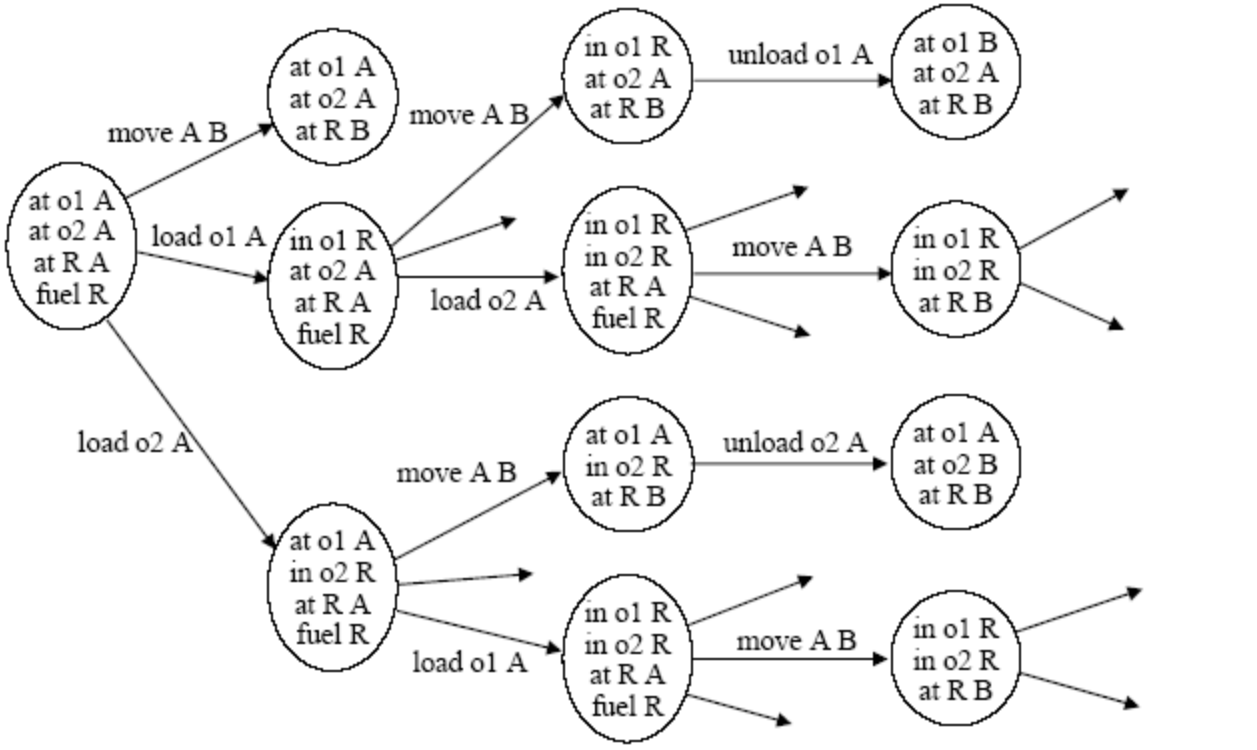
Path Planning

- Map and connectivity is usually known
- Distance metrics exist for approximating the distance between nodes
- Map is usually finite so easier to tell when search is complete

PDDL Specification

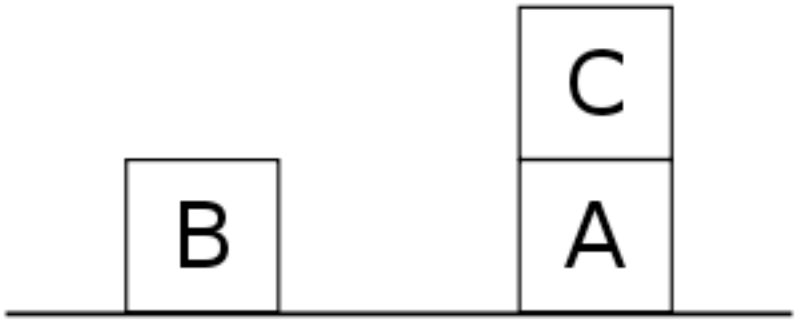
```
(define (domain blocksworld)
(:predicates (clear ?x)
(holding ?x)
(on ?x ?y))
(
    :action stack
    :parameters (?ob ?underob)
    :precondition (and (clear ?underob) (holding ?ob))
    :effect (and (holding nil) (on ?ob ?underob)
    (not (clear ?underob)) (not (holding ?ob)))
)
```

State-Space Planner



Number of states can be quite large for real-world planning problems…….

Sussman Anomaly



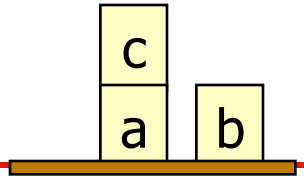
2 subgoals

- A on B
- B on C

Planner must interleave goal achievement.

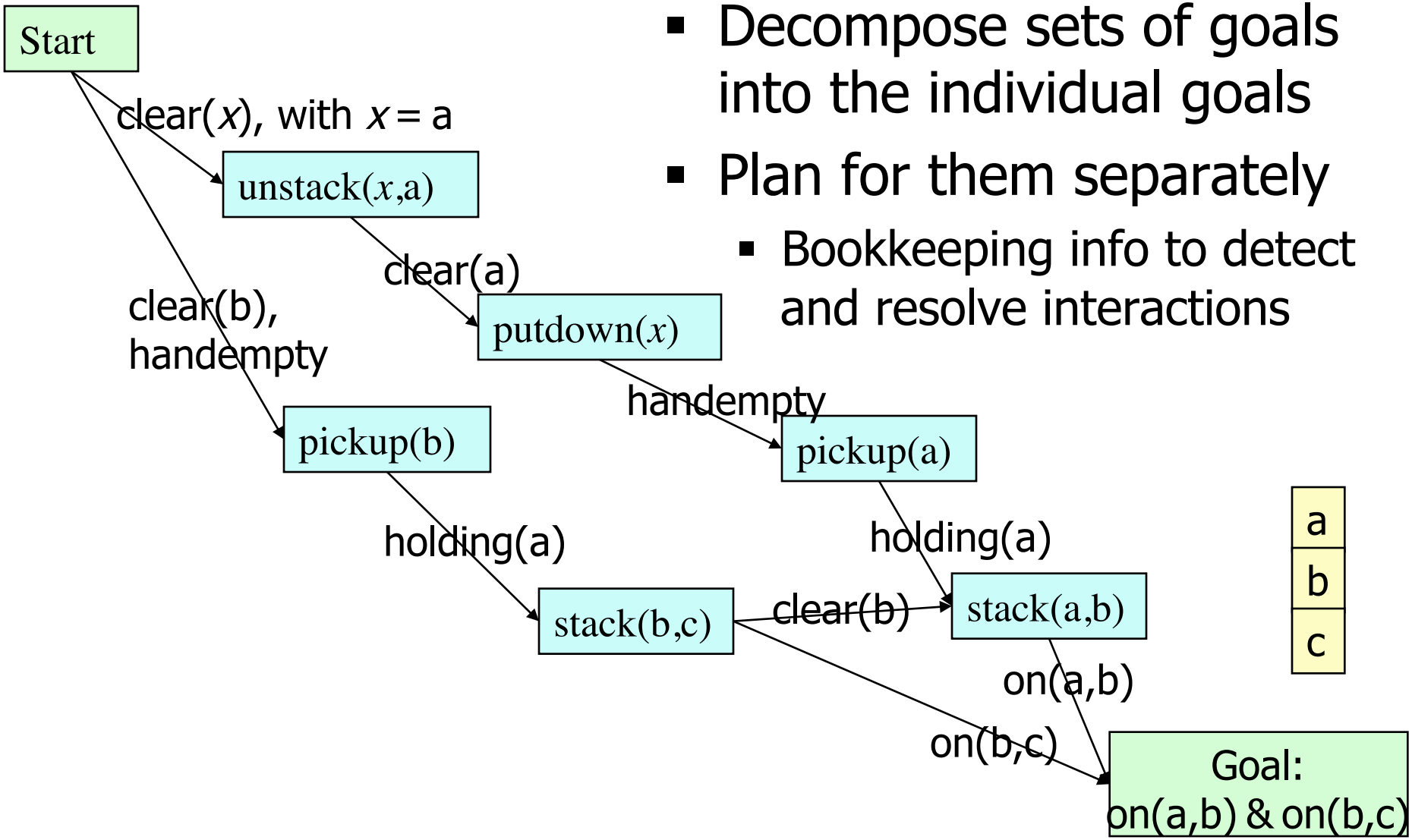
Completing either goal in entirety will render the other goal impossible

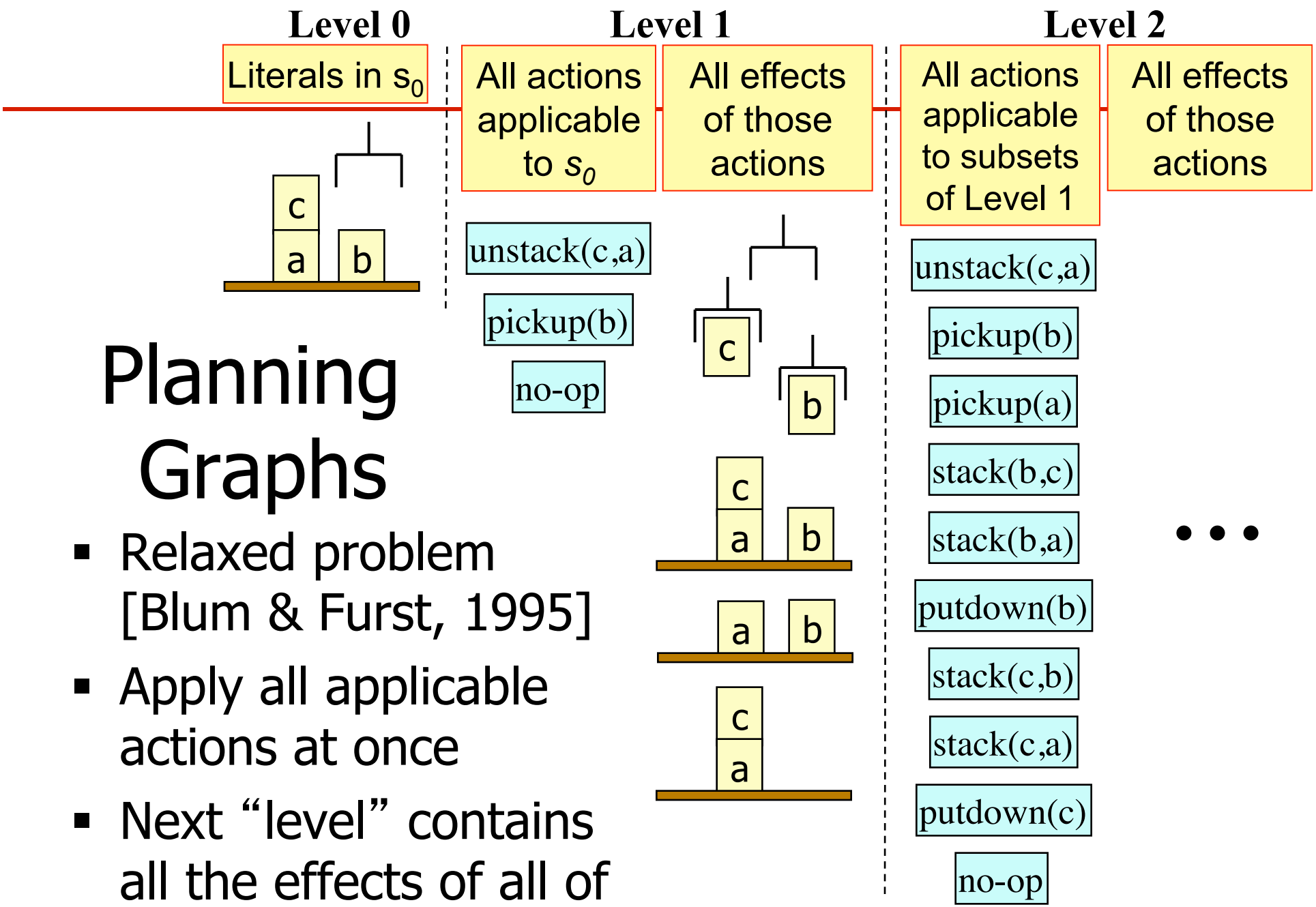
The Sussman anomaly shows that methods that treat subgoals independently are not guaranteed to be complete.



Plan-Space Planning (UCPOP)

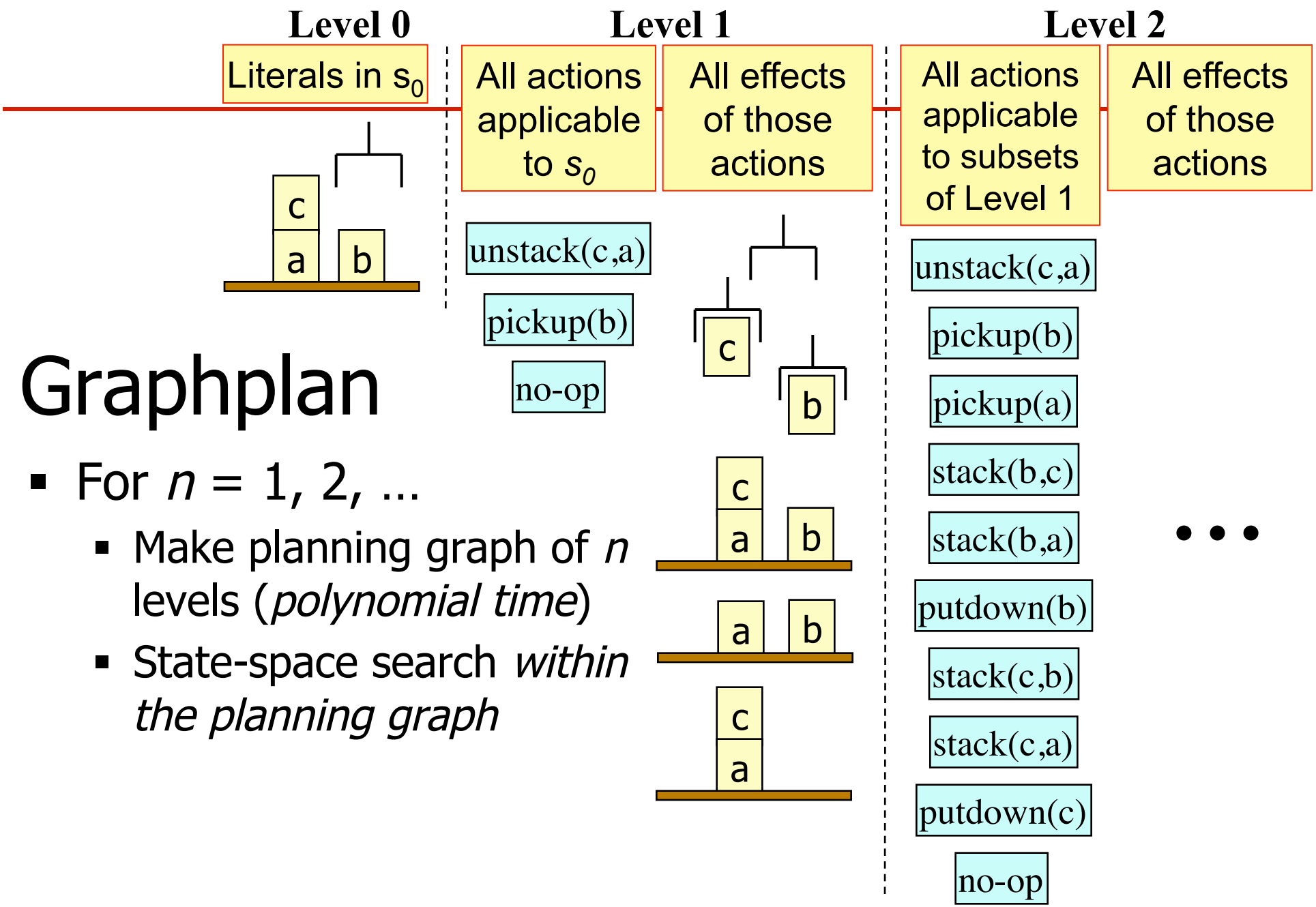
- Decompose sets of goals into the individual goals
- Plan for them separately
  - Bookkeeping info to detect and resolve interactions





Planning Graphs

- Relaxed problem [Blum & Furst, 1995]
- Apply all applicable actions at once
- Next "level" contains all the effects of all of those actions



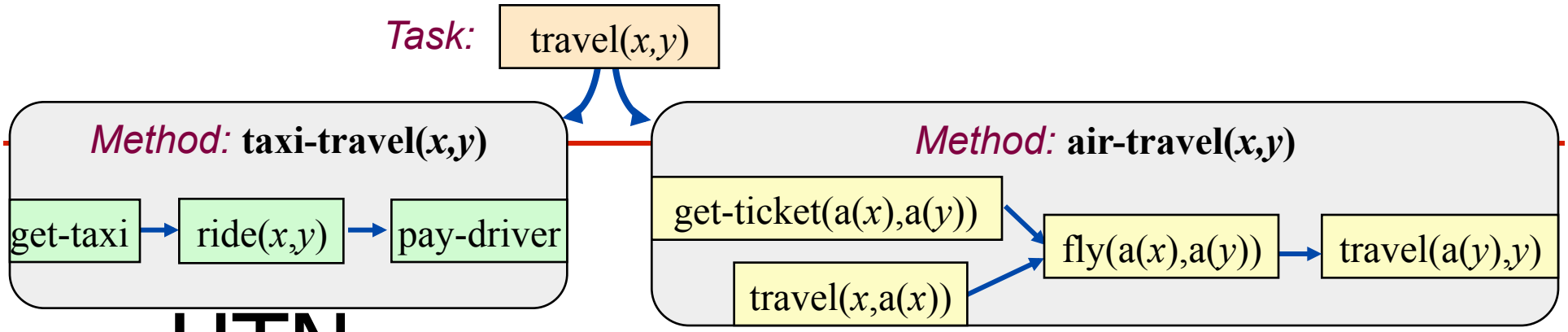
Graphplan

- For n = 1, 2, …
  - Make planning graph of n levels (*polynomial time*)
  - State-space search *within the planning graph*

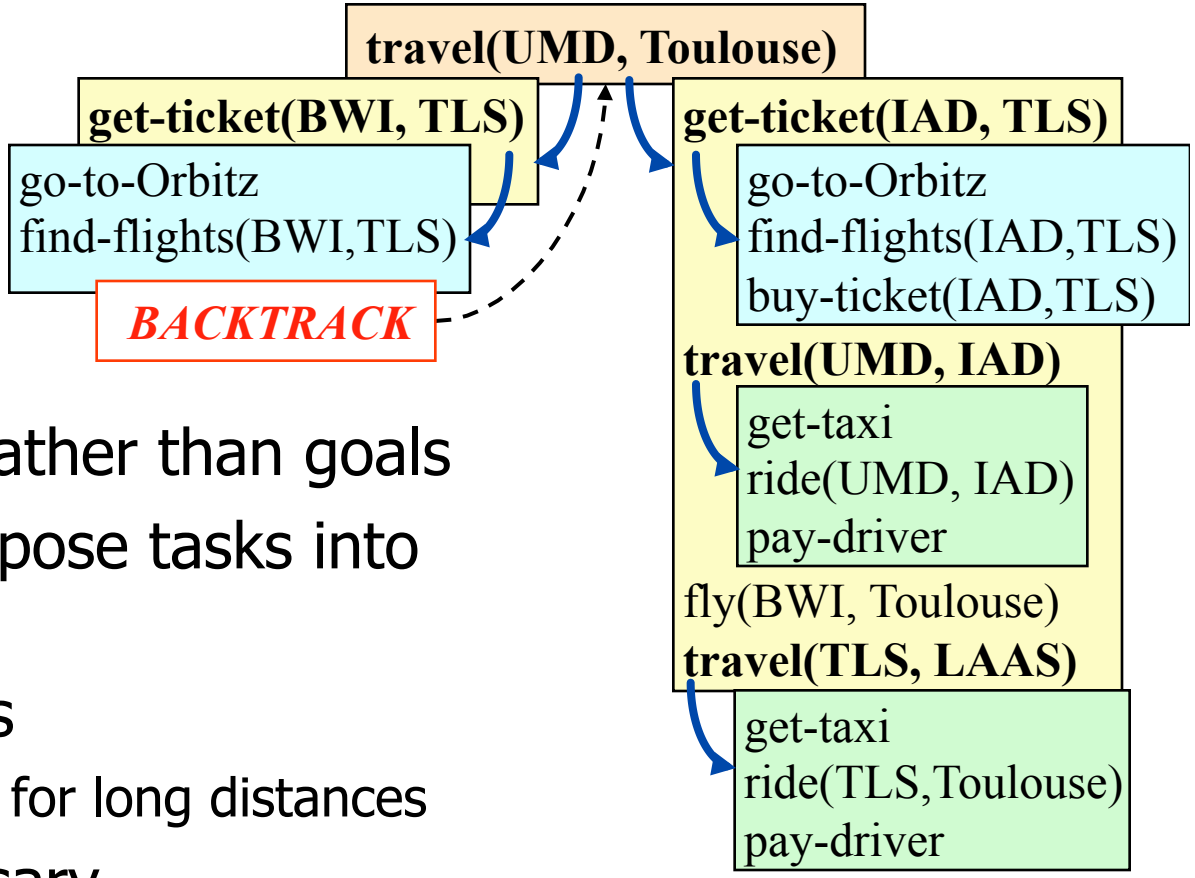
Planning as Humans Do It

- Make a plan to get you from your bed to UCF in the morning
  - Do you consider all possible options that you can take ? OR
  - Do you remember recipes and procedures that have worked successfully for you in the past?

- That's what HTN (hierarchical-task network) planning is about!

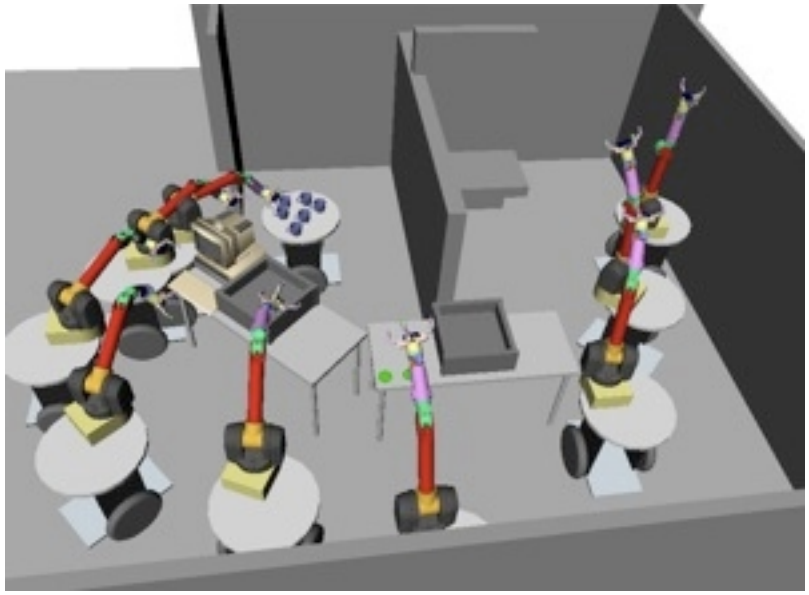


HTN Planning



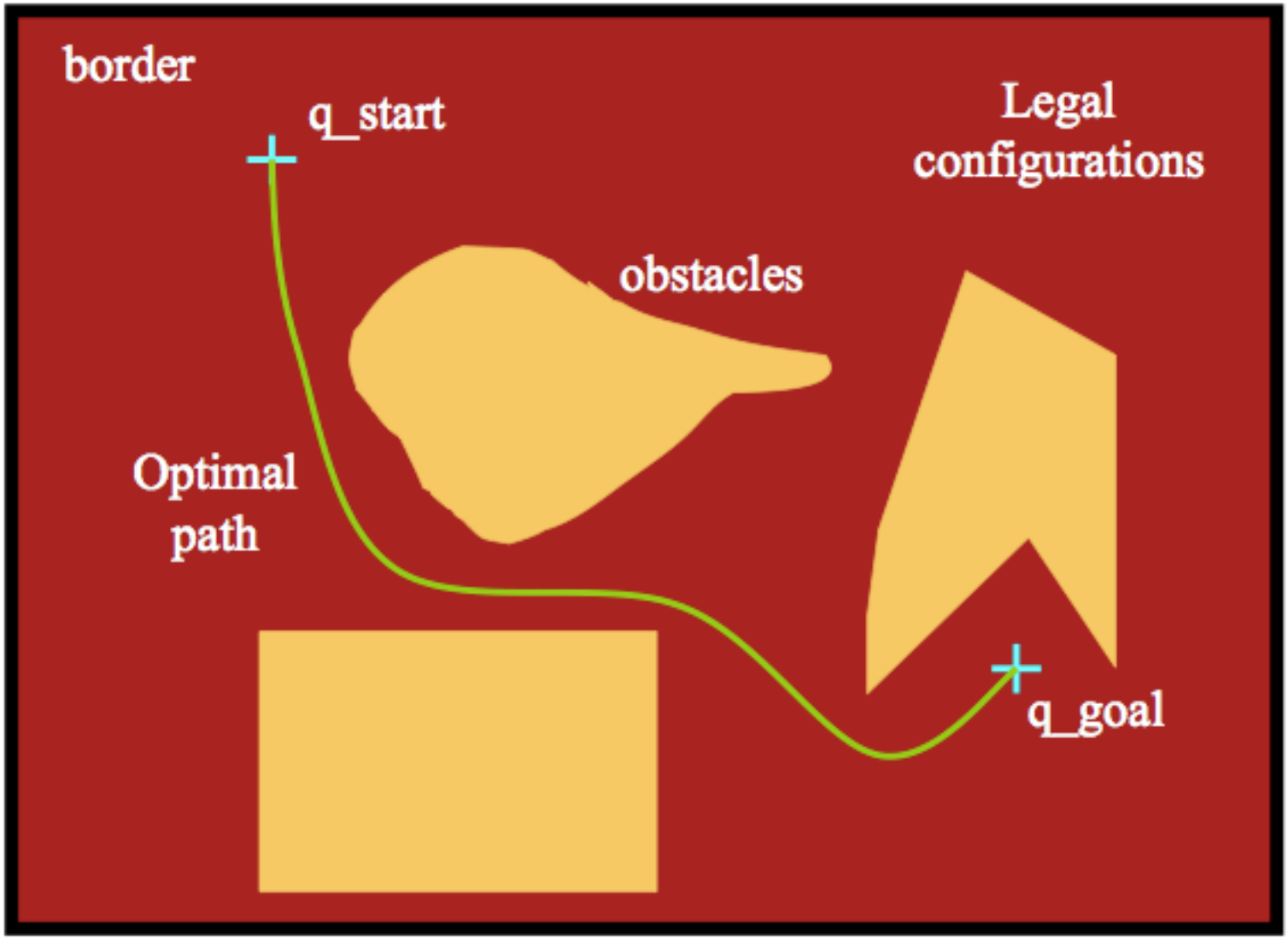
- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints
    - E.g., taxi not good for long distances
  - Backtrack if necessary

High Dimensional Planning



Robots with many joints need special planning techniques

Configuration Space



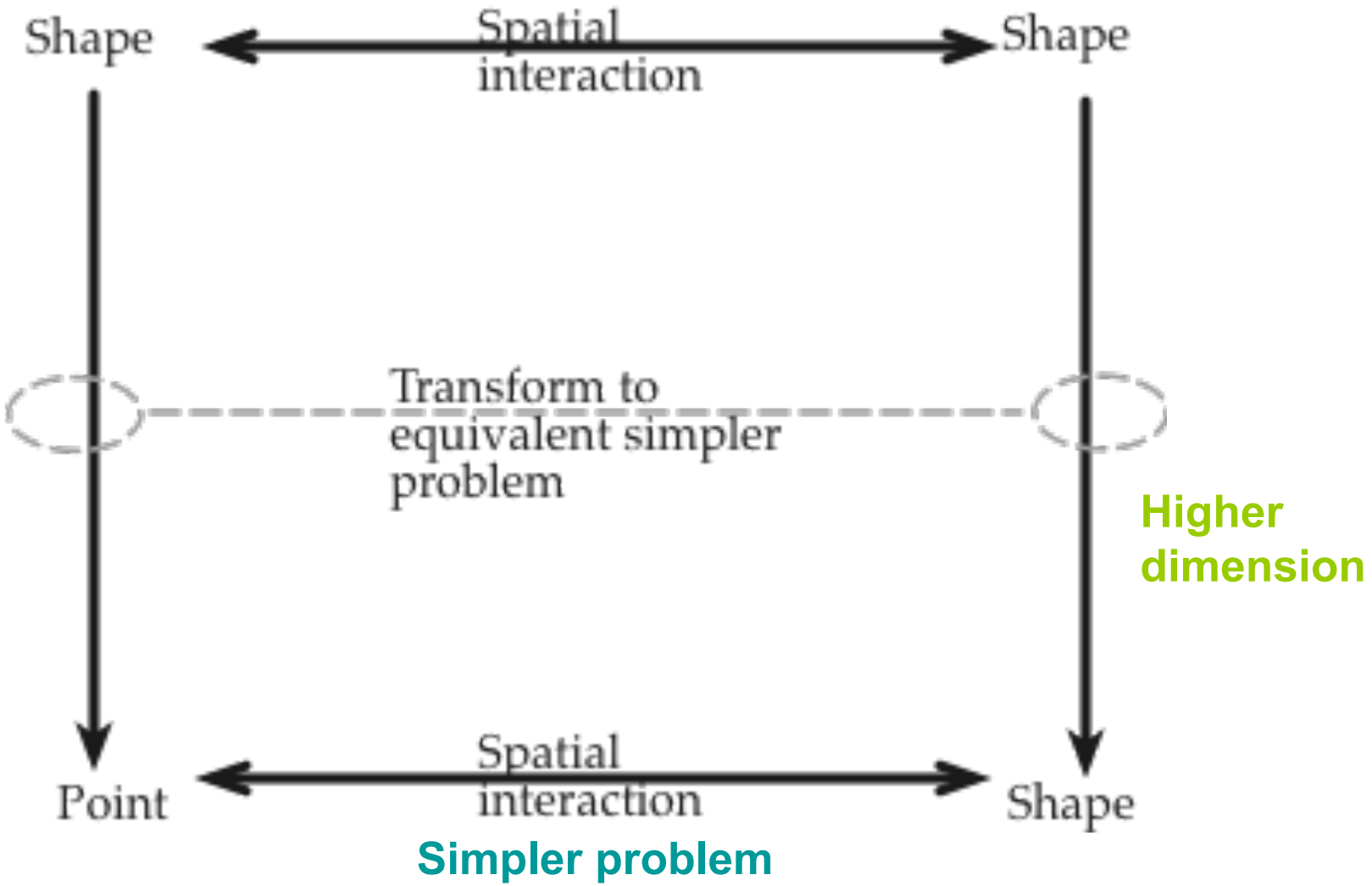
Configuration Space

- **Workspace:** spatial region in which a robot can operate
- **Configuration:** a robot state
- **Configuration space:** the set of all possible robot spaces
- **Legal configuration:** a configuration that doesn't
  - Violate any obstacles
  - Violate the physical constraints of the robot
  - Violate any operator defined constraints

Basic Definitions

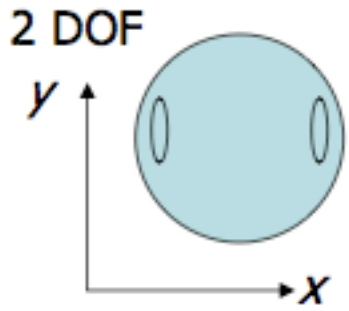
- **Q_start**: the start configuration
- **Q_goal**: the goal configuration (may have more than one)
- **C-space Transform**: transformation to a robot and obstacles that turns a robot into a single point

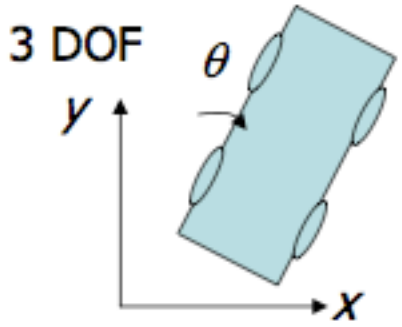
Transforming to C-Space

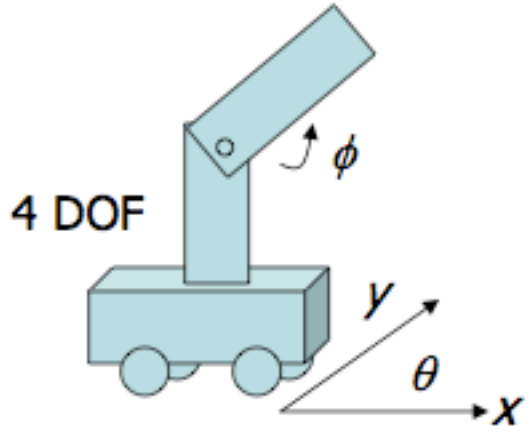


Configuration Space

- A robot has:
  - A footprint
    - The amount of space a robot occupies
  - Degrees of freedom (DOF)
    - The number of variables needed to fully describe a robot's position in space







C-Space Transform

- For a mobile robot that only translates
  - Choose a reference point on the robot.
  - Grow obstacle by translating the robot around the edge of the obstacle and tracing line made by the reference point of the robot.
  - Represent the robot only by the reference point.
  - Legal configurations now consist of all non-obstacle points.
- Robots that can rotate are usually represented by a circle.

  Once the obstacles have been grown, you can plan a path!

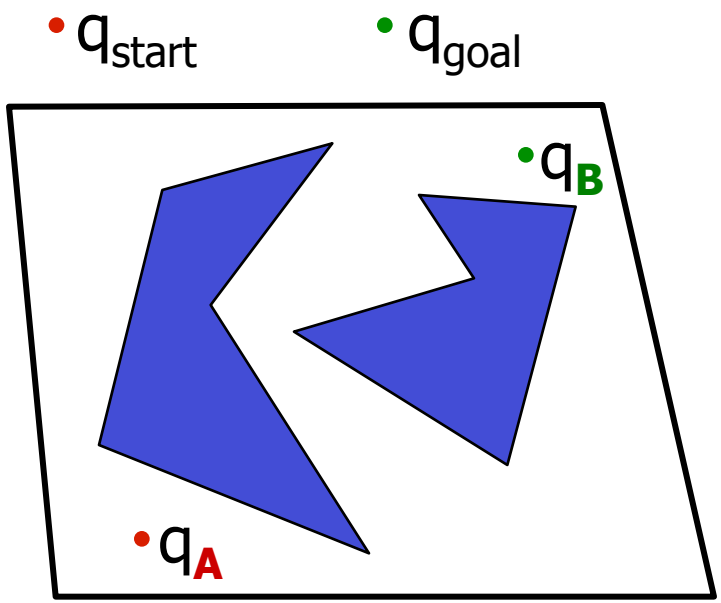
Motion Planning in C-Space

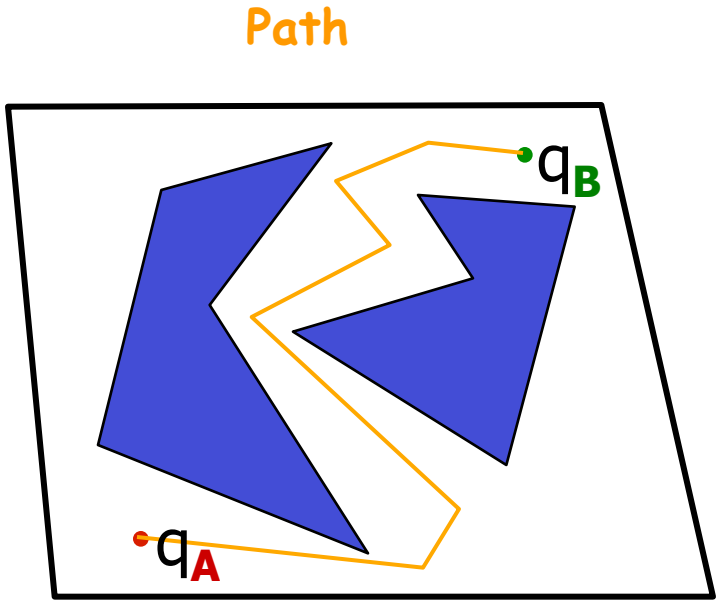
Input

- geometric descriptions of a robot and its environment (obstacles)
- initial and goal configurations

Output

- a path from start to finish (or the recognition that none exists)
- usually in configuration space

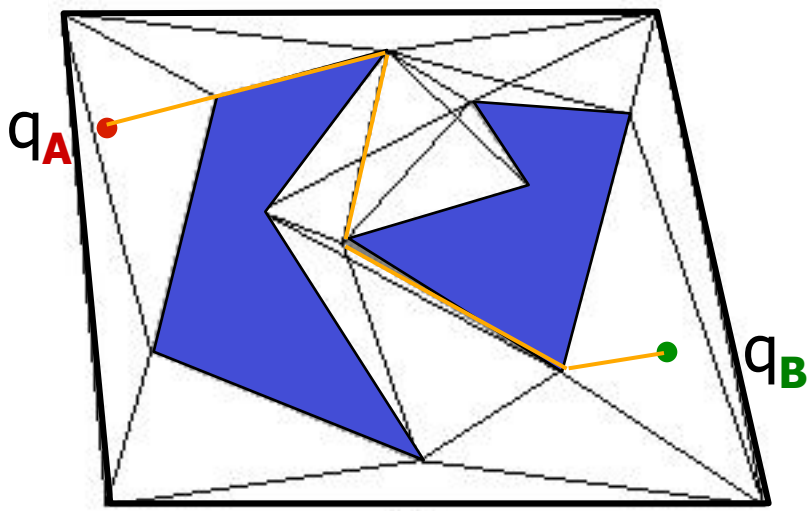




What's different?

- Before we were doing planning on an occupancy grid at single discretization level.
- Here obstacles in the world are represented by polygons.
- Planning is a 2 stage process:
  - Graph construction
  - Path planning (using A*/Djkstra's shortest path)
- Algorithms differ at the graph creation phase.

Visibility Graphs



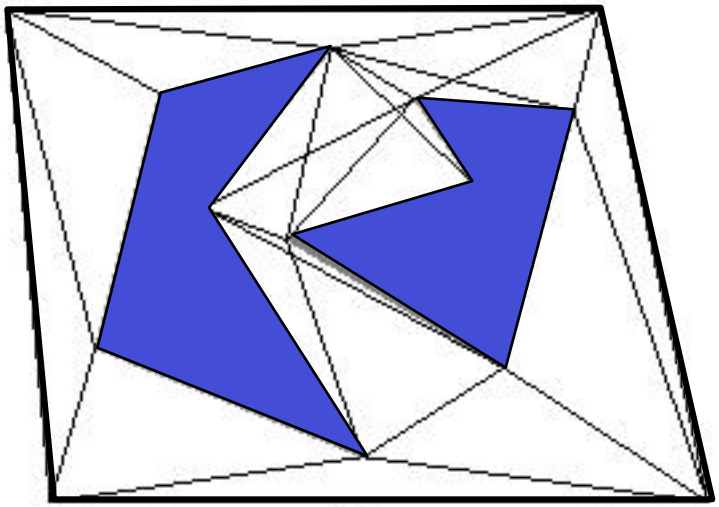
don't need the outer perimeter

In a polygonal (or polyhedral) configuration space, construct all of the line segments that connect vertices to one another…

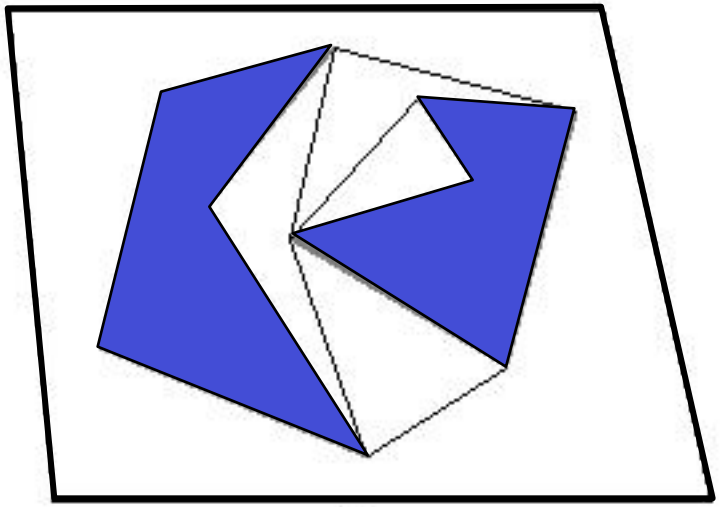
Converts the problem into one of graph search.

N = the number of vertices in configuration space

Visibility Graphs



Full visibility graph



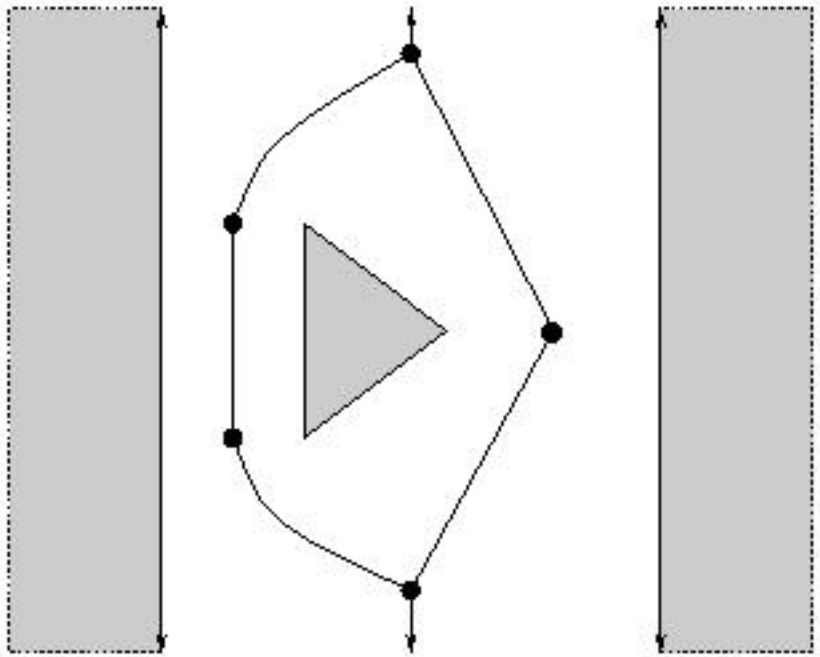
Reduced visibility graph, i.e., not including segments that extend into obstacles on either side.

benefits?
drawbacks?

Visibility Graphs

- Given polygonal obstacles, shortest path shaves corner on obstacles
- Problems
  - Can be slow to build
  - Path shaves corners
    - Not a very robust path
    - Requires some path smoothing or relaxation to improve travel time
- Strengths:
  - Easy to localize since you stay near obstacles

Generalized voronoi diagrams



The GVD (generalized voronoi diagram) is the set of points and lines that are ***maximally distant*** from obstacles.

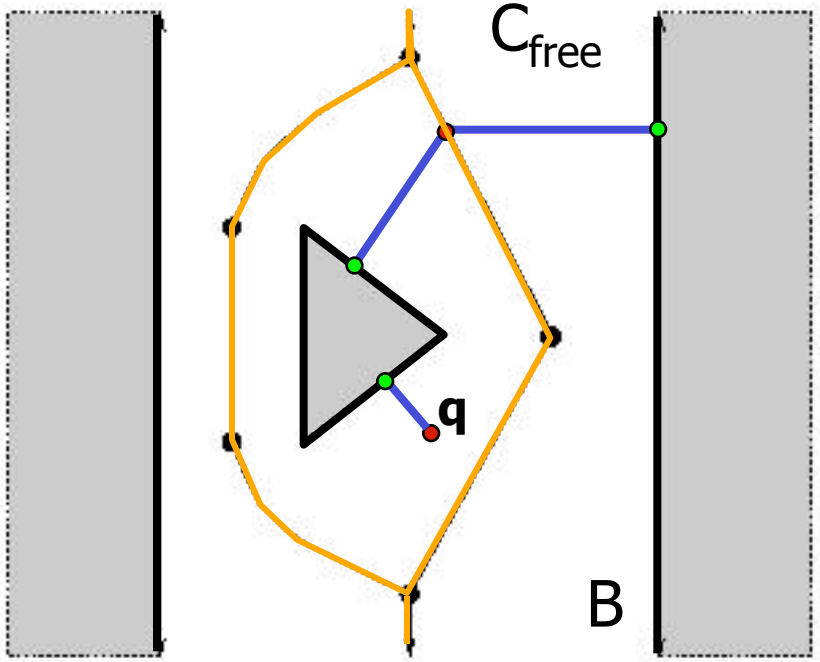
Mathematical construction

Computational construction

Voronoi diagrams

Evaluation

+ maximizes distance from obstacles
+ reduces to graph search
+ can be used in higher dimensions
- nonoptimal
- real diagrams tend to be noisy



Let B = the boundary of C_{free} .

Let **q** be a point in C_{free} .

Define *clearance*(q) = min { | q - p | }, for all p ∈ B

Define *near*(q) = { p ∈ B such that | q - p | = *clearance*(q) }

q is in the *gen. Voronoi diagram* of C_{free} if | *near*(q) | > 1

number of set elements

Voronoi Graph

Drawbacks:

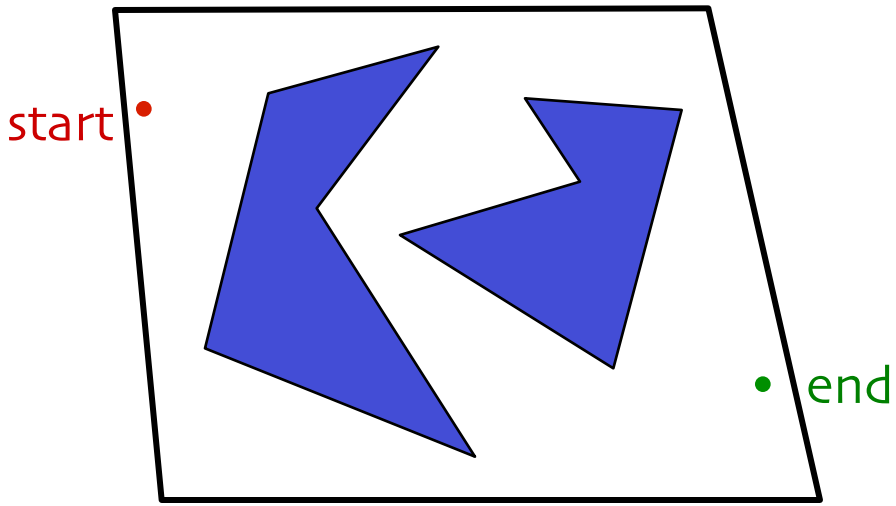
- Requires complex algorithms to calculate graph
- Can be very complex (and slow)
- Can produce non-optimal results depending on configuration of objects

Strengths:

Large safety margin for robot

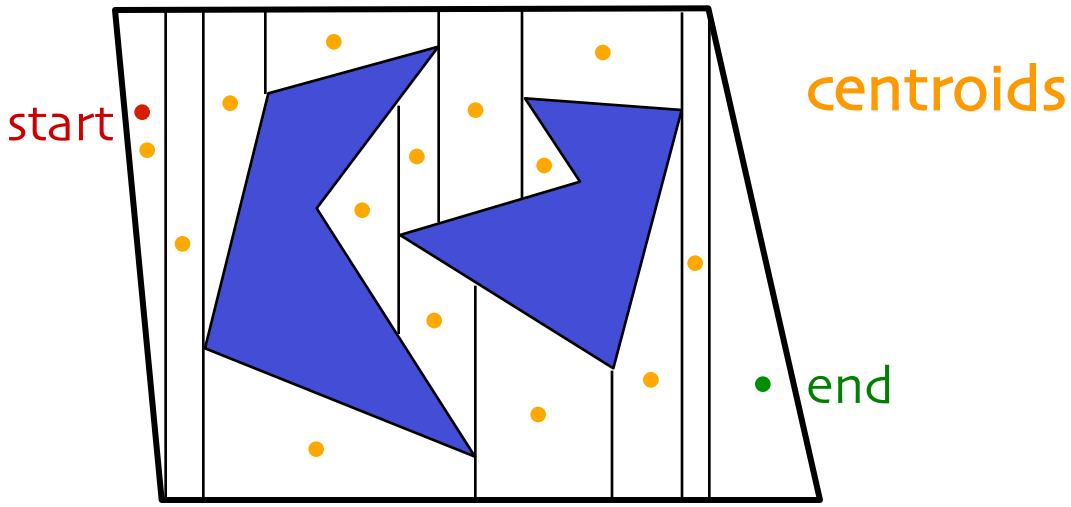
Spatial decompositions

Dividing free space into pieces and using those...



Spatial decompositions

At each vertex *v*, draw 2 segments: upper and lower vertical extensions.



Exact cell decomposition

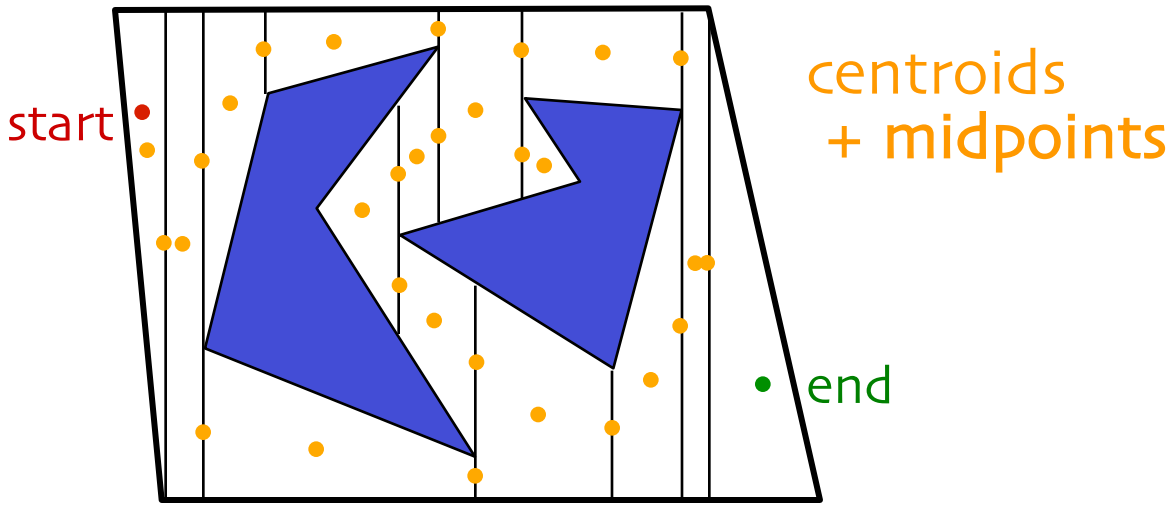
sweepline
algorithm

Running time?

O(N log(N))

Spatial decompositions

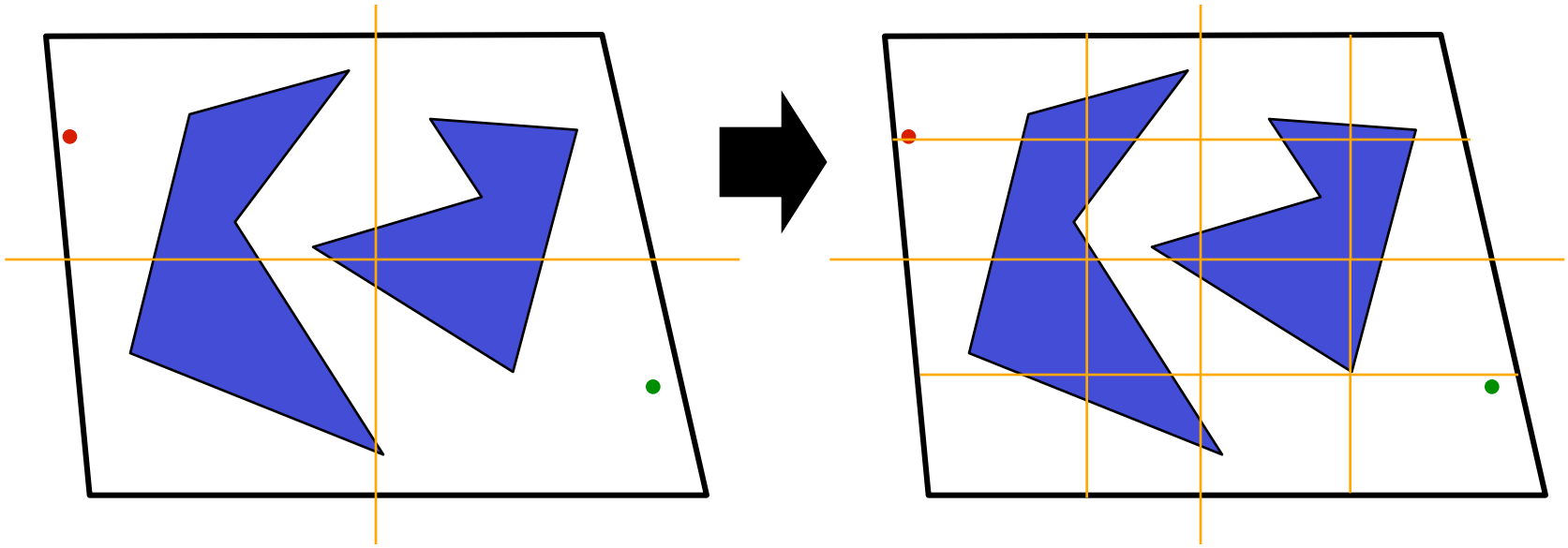
Path plan by connecting the midpoints of the vertex extensions with the centroids of the trapezoids.



	Running time?	Path?
Exact cell decomposition sweepline algorithm	O(N log(N))	via edge midpoints !

further decomposing...

Approximate cell decomposition

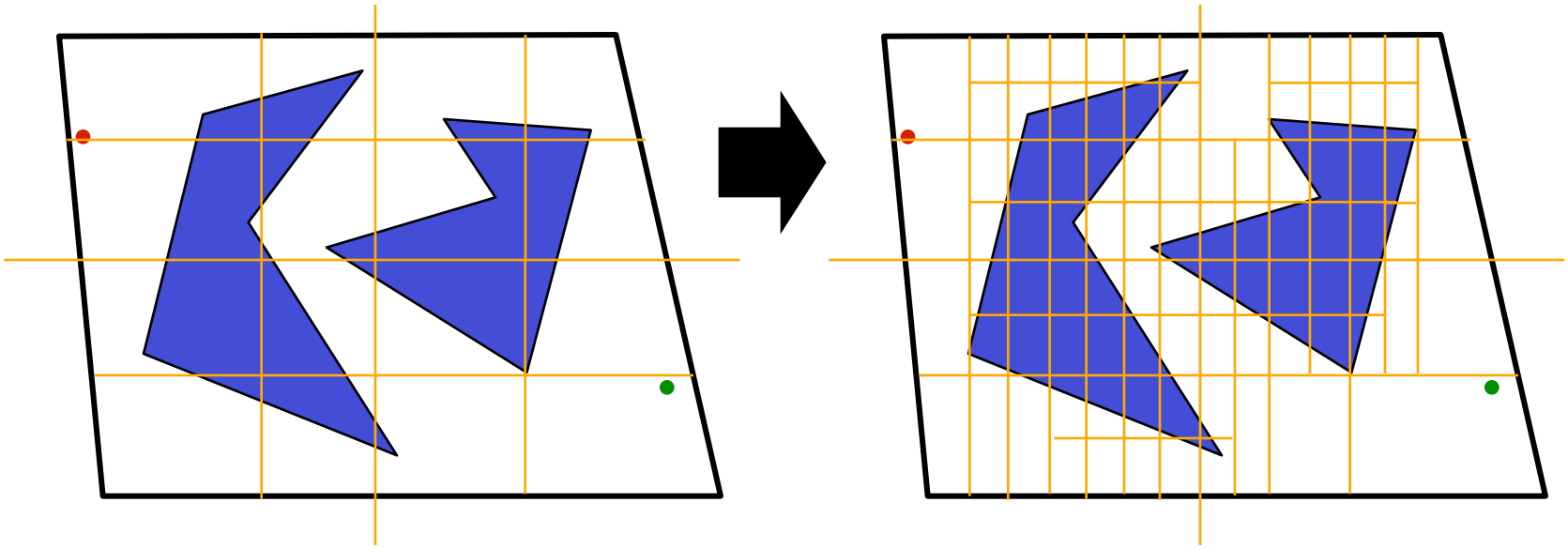


any of these we should leave alone?

Quadtree: recursively subdivides each *mixed* obstacle/ free (sub)region into four quarters...

further decomposing...

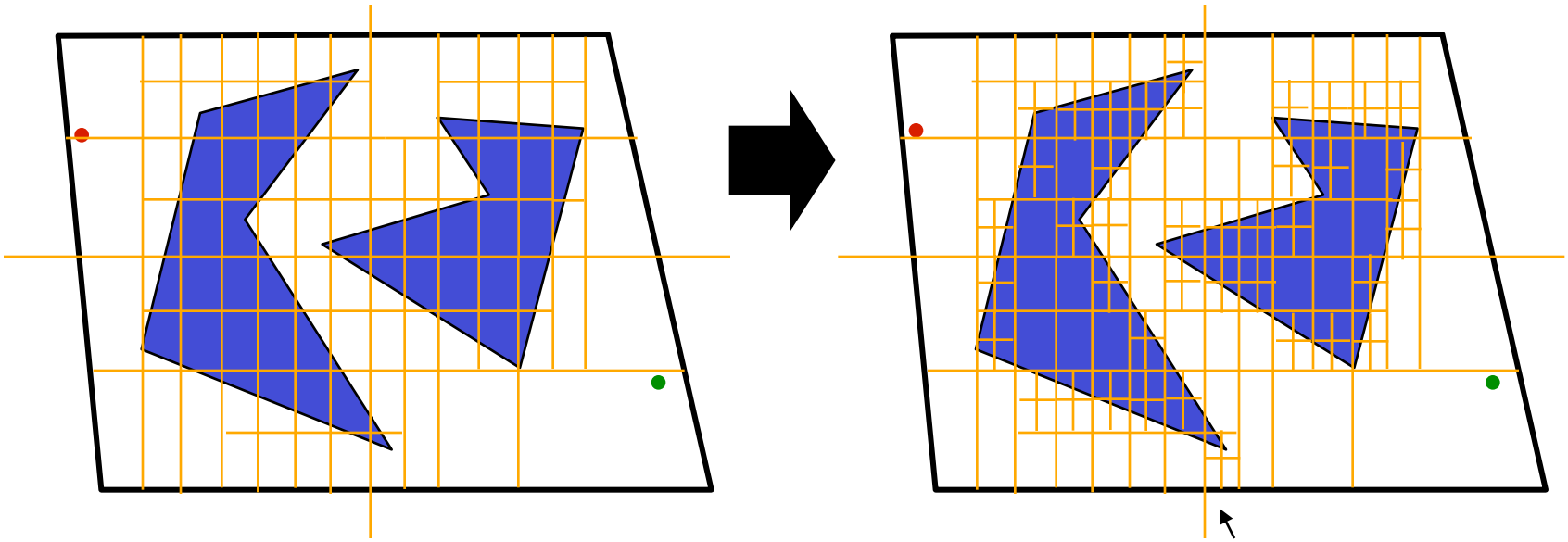
Approximate cell decomposition



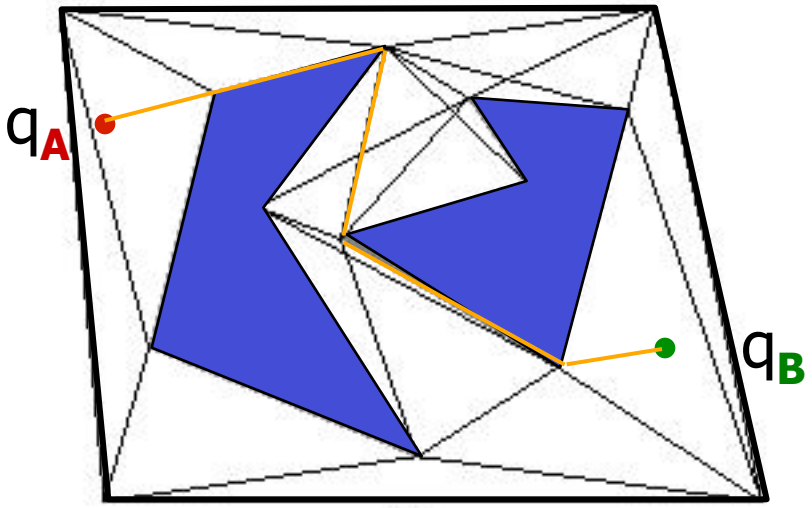
Quadtree: recursively subdivides each *mixed* obstacle/ free (sub)region into four quarters...

further decomposing...

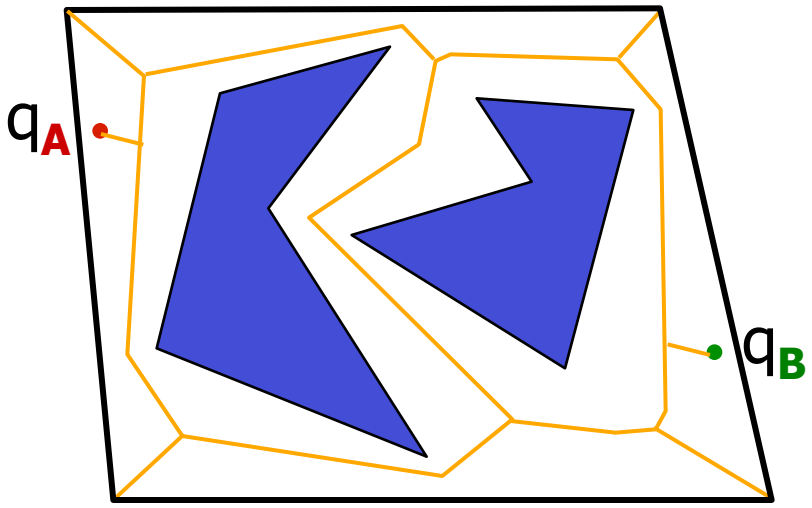
Approximate cell decomposition



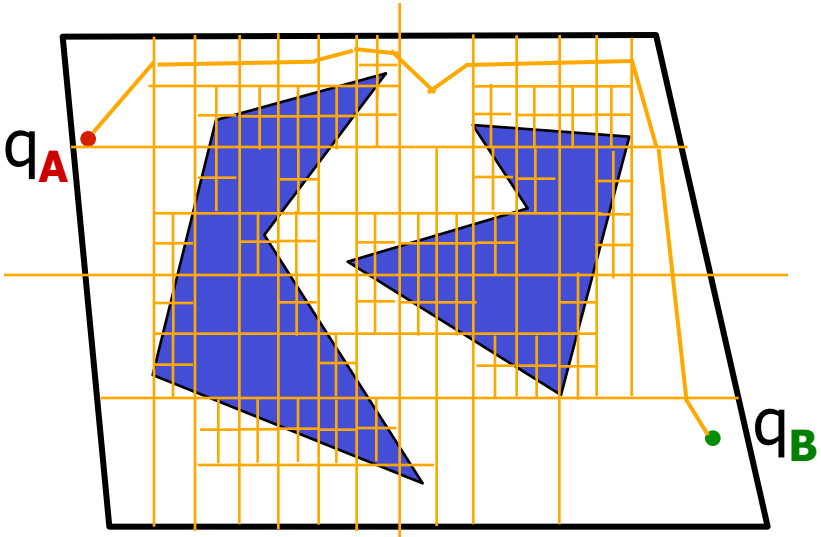
Quadtree: recursively subdivides each *mixed* obstacle/ free (sub)region into four quarters...



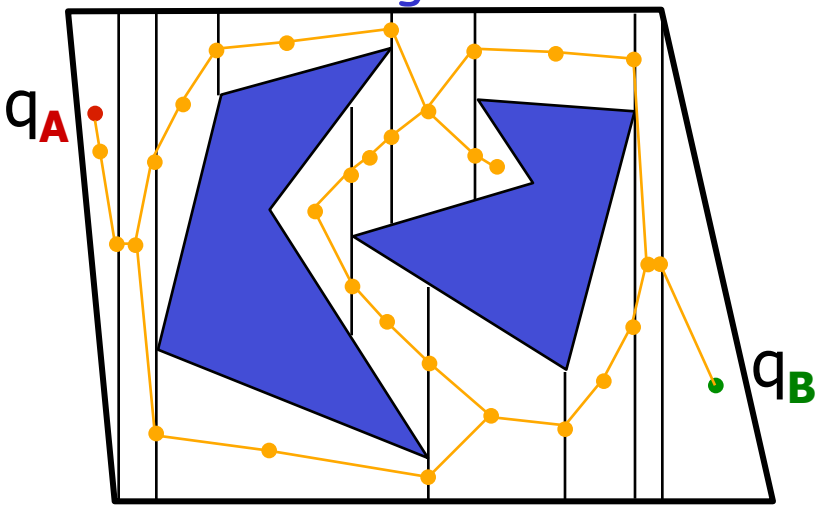
Visibility Graph Search



Search via Voronoi Diagrams



Quadtree (Approximate) Cell Decomposition



Trapezoidal Cell Decomposition

Sampling Based Planners

- In high dimensional spaces, exact geometric planners are impractical
- Sampling-based planners can operate in high-dimensional spaces efficiently
- Examples:
  - Roadmap methods
  - Rapidly exploring trees

Probabilistic Road Maps (PRM) for finding paths [Kavraki at al 96]

C-space

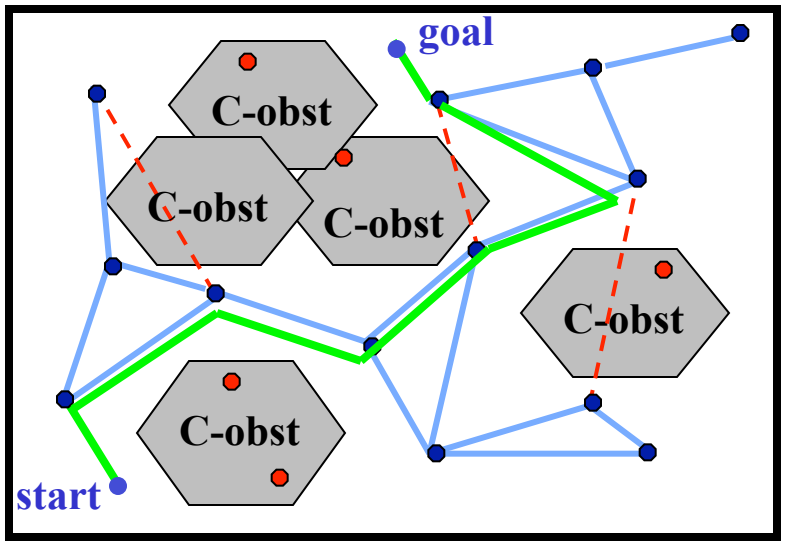


Image courtesy of Nancy Amato

Roadmap Construction (Pre-processing)

1. Randomly generate robot configurations (nodes)
   - discard nodes that are invalid
2. Connect pairs of nodes to form **roadmap**
   - simple, deterministic *local planner* (e.g., straightline)
   - discard paths that are invalid

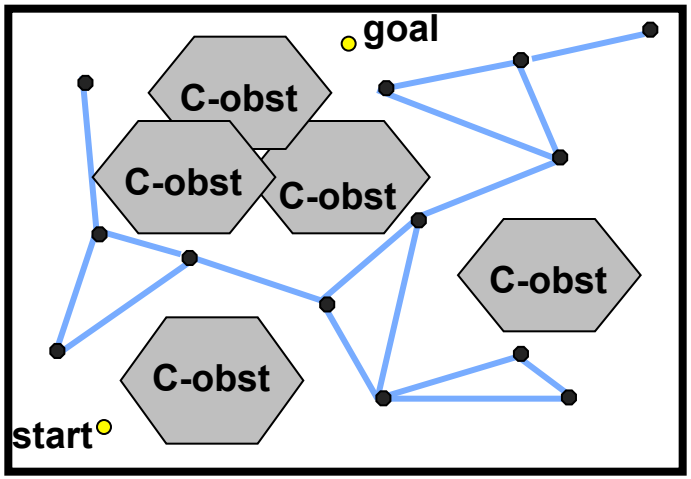
Query processing

1. Connect *start* and *goal* to roadmap
2. Find path in roadmap between *start* and *goal*
   - regenerate plans for edges in roadmap

Primitives Required:

1. Method for Sampling points in C-Space
2. Method for `validating' points in C-Space

More PRMS

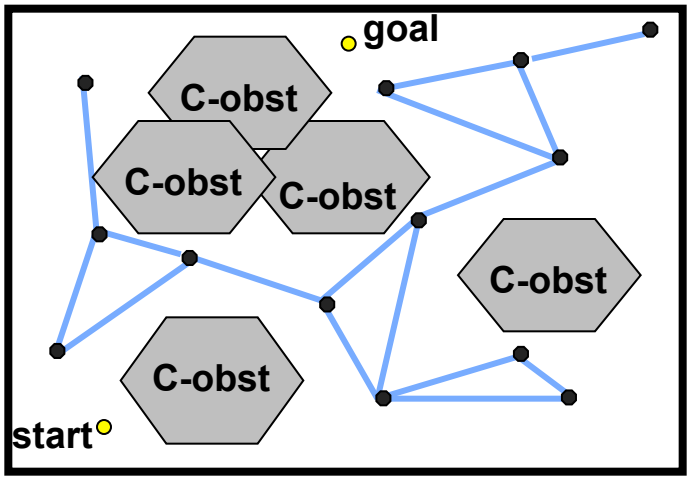


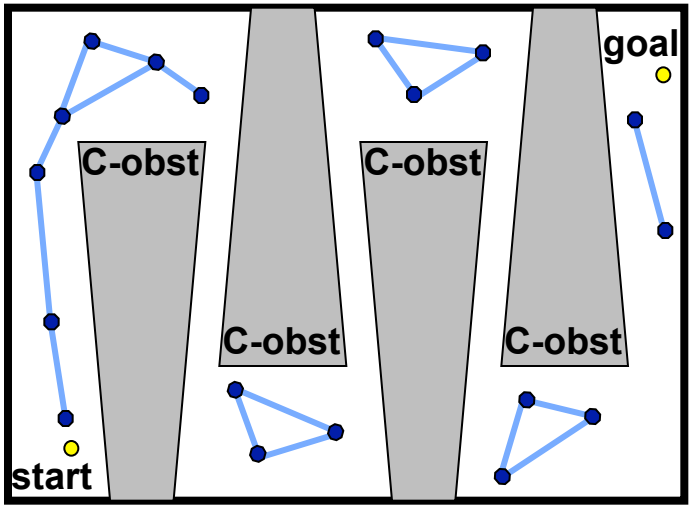
PRMs: Pros

1. PRMs are *probabilistically complete*
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems

More PRMS





PRMs: Pros

1. PRMs are *probabilistically complete*
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems

PRMs: Cons

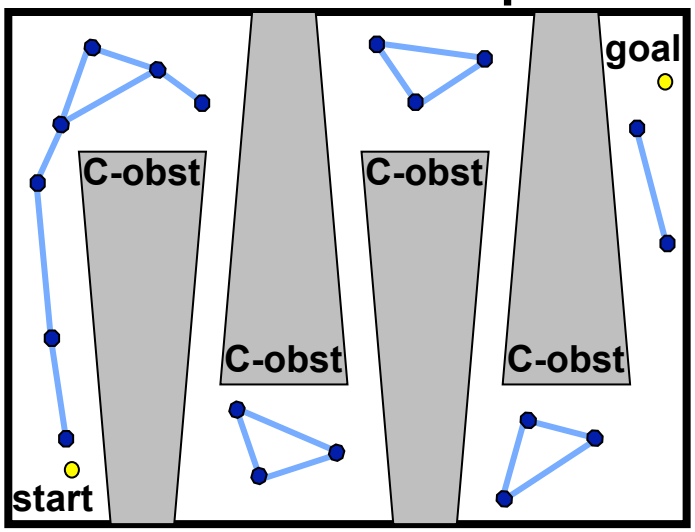
1. PRMs don't work as well for some problems:
– unlikely to sample nodes in *narrow passages*
– hard to sample/connect nodes on constraint surfaces

Sampling Around Obstacles
[Amato et al 98]

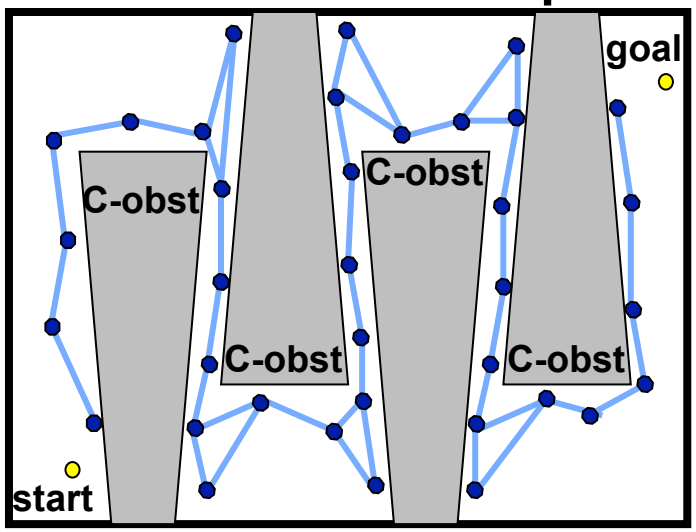
To Navigate Narrow Passages we must sample in them

- most PRM nodes are where planning is easy (not needed)

PRM Roadmap



OBPRM Roadmap



Images courtesy of Nancy Amato

Idea: Can we sample nodes near C-obstacle surfaces?

- we cannot explicitly construct the C-obstacles...
- we do have models of the (workspace) obstacles...

Repairing Paths [Amato et al]

Even with the best sampling methods, roadmaps may not contain valid solution paths

- may lack points in narrow passages
- may contain approximate paths that are nearly valid

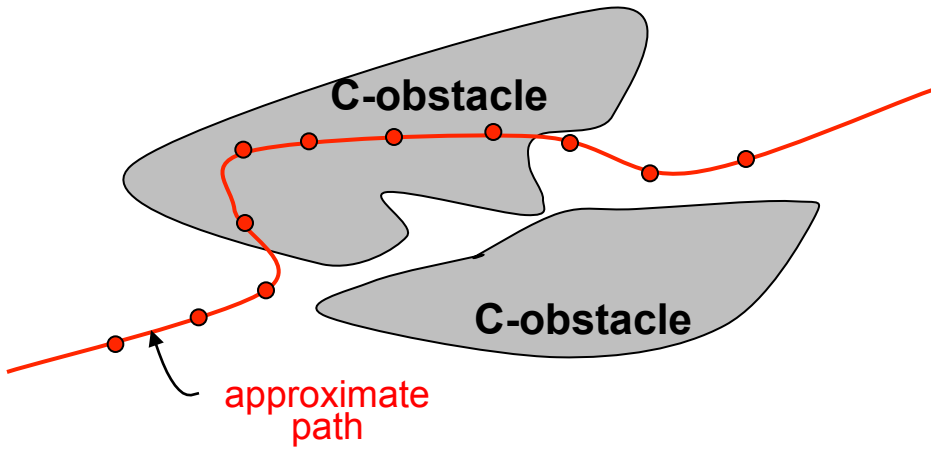


Image courtesy of Nancy Amato

Repairing Paths [Amato et al]

Even with the best sampling methods, roadmaps may not contain valid solution paths

- may lack points in narrow passages
- may contain approximate paths that are nearly valid

Repairing/Improving Approximate Paths

1. Create initial roadmap
2. Extract *approximate path P*
3. Repair P (push to C-free)
   - Focus search around P
   - Use OBPRM-like techniques

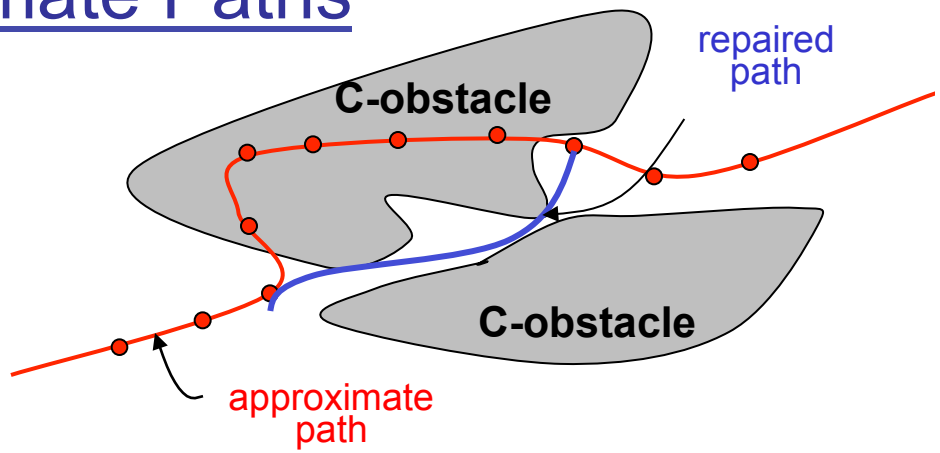


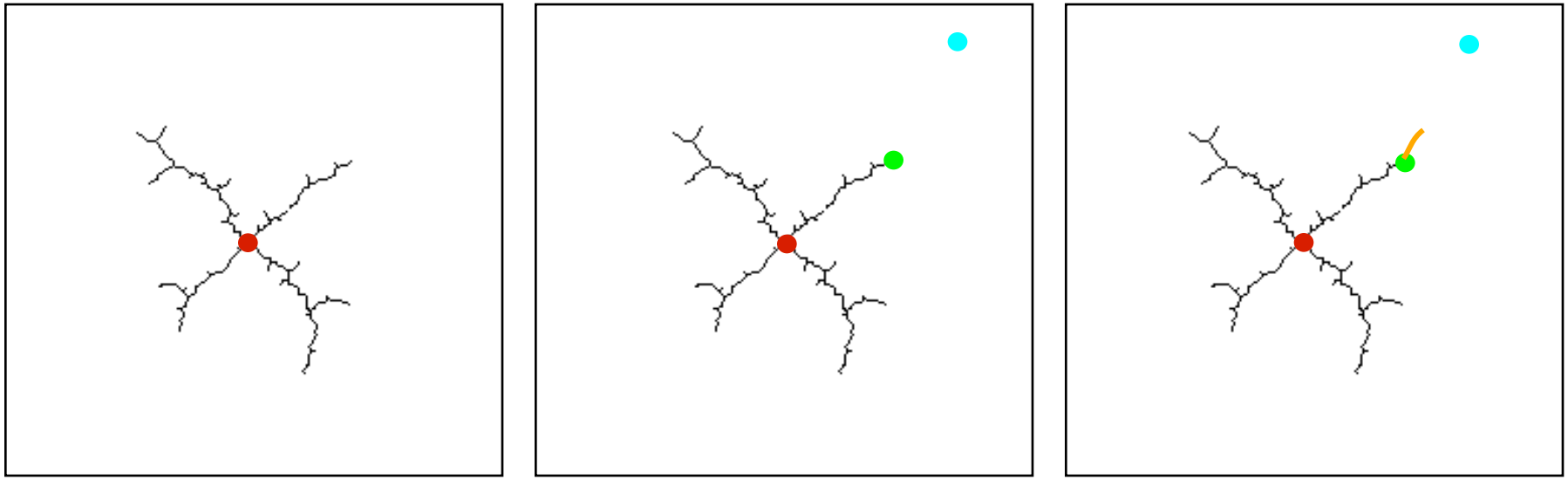
Image courtesy of Nancy Amato

Example

https://www.youtube.com/watch?v=voC56tR48ac&feature=youtu.be

RRT

RRTs -- “Rapidly-exploring random trees”



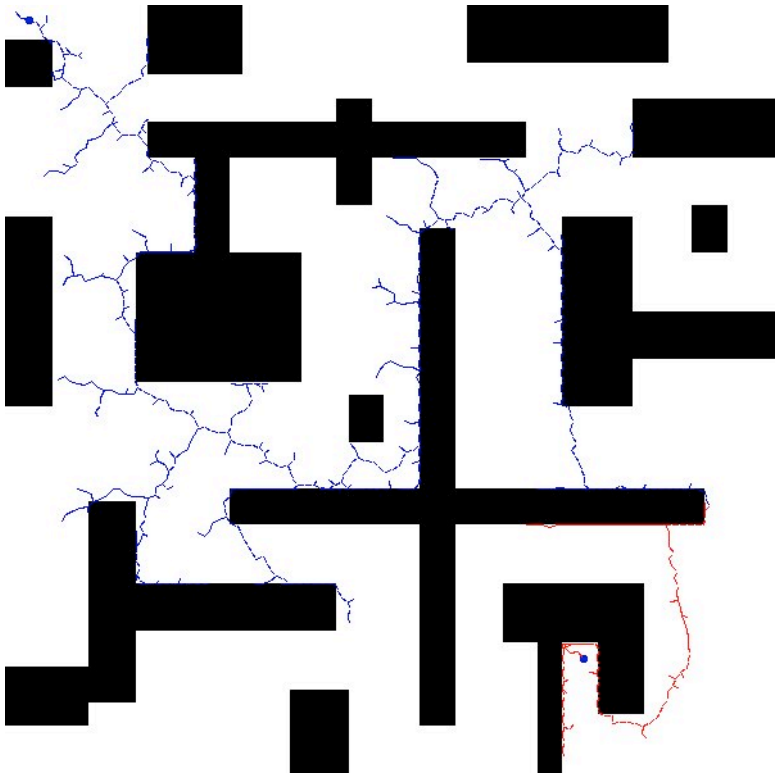
1) Maintain a tree of configurations reachable from the starting point.

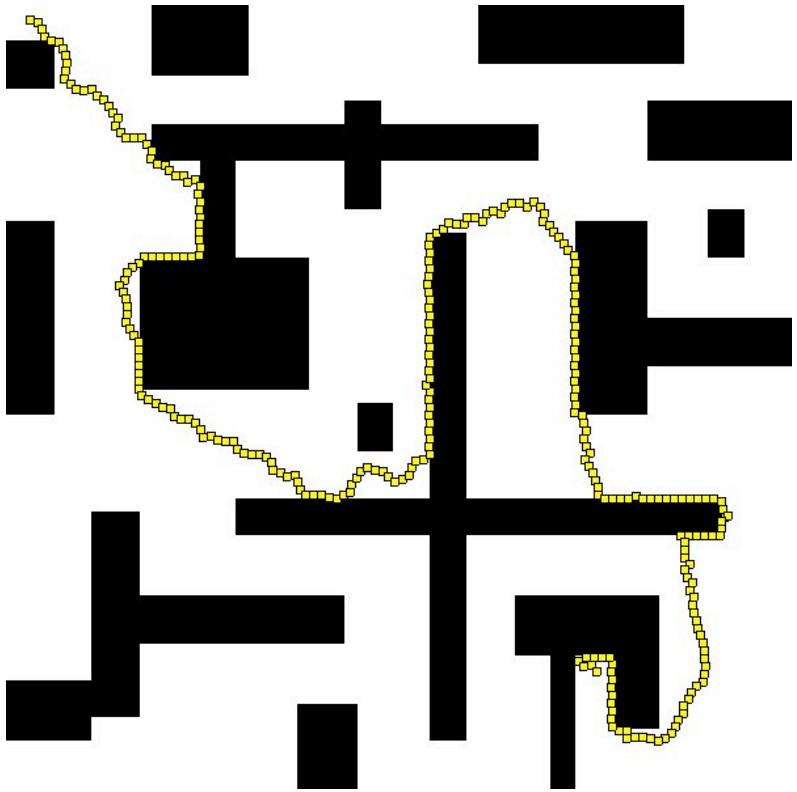
2) Choose a point at random from ***free space***

3) Find the closest configuration already in the tree

4) Extend the tree in the direction of the new configuration /“EXTEND” step

RRT Maze





Tree Growth

- http://msl.cs.uiuc.edu/rrt/gallery_2drrt.html

X-Wing Fighter

- http://msl.cs.uiuc.edu/rrt/gallery_xwing.html

Conclusion

Motion planning in high-dimensional space is a computationally hard problem.

Even planning in low-dimensions with incomplete information can be difficult due to non convex shapes.

Next time….on to machine learning!