Motion Planning for a Point Robot (2/2)

1

- Class scribing
- Position paper

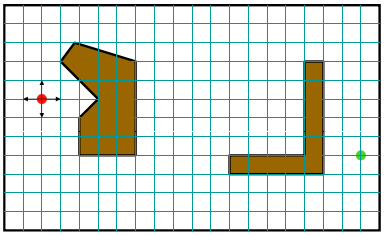
2

Planning requires models

- The Bug algorithms are "reactive motion strategies"; they are not motion planners
- To plan its actions, a robot needs a (possibly imperfect) **predictive model** of its actions, so that it can choose among several possible courses of action

3

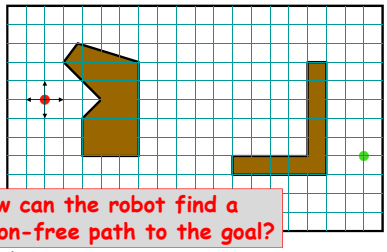
Point Robot on a Grid



Assumptions:
- The robot perfectly controls its actions
- It has an accurate geometric model of the environment (i.e., the obstacles)

4

Point Robot on a Grid



How can the robot find a collision-free path to the goal?

Assumptions:
- The robot perfectly controls its actions
- It has an accurate geometric model of the environment (i.e., the obstacles)

5

8-Puzzle Game

8	2	
3	4	7
5	1	6

Initial state

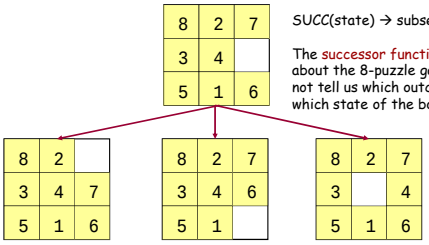
1	2	3
4	5	6
7	8	

Goal state

State: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board

6

8-Puzzle: Successor Function



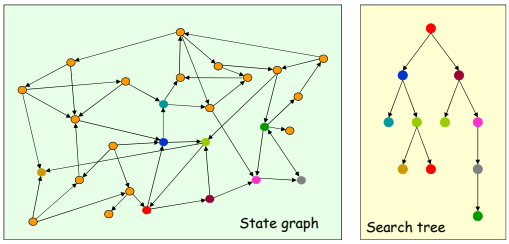
SUCC(state) → subset of states

The successor function is knowledge about the 8-puzzle game, but it does not tell us which outcome to use, nor to which state of the board to apply it.

Search is about the exploration of alternatives

7

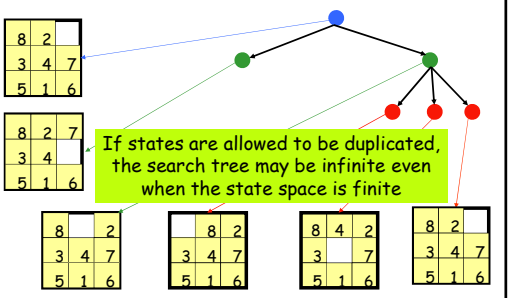
State Graph and Search Tree



Note that some states may be visited multiple times

8

Search Nodes and States



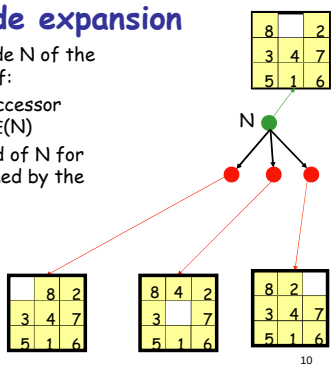
If states are allowed to be duplicated, the search tree may be infinite even when the state space is finite

9

Node expansion

The expansion of a node N of the search tree consists of:

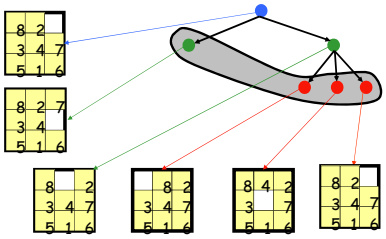
1) Evaluating the successor function on STATE(N)
2) Generating a child of N for each state returned by the function



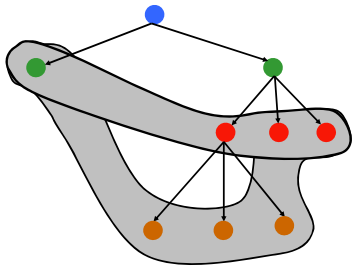
10

Fringe of Search Tree

- The fringe is the set of all search nodes that haven't been expanded yet



11



12

Search Strategy

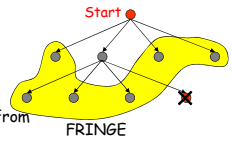
- The fringe is the set of all search nodes that haven't been expanded yet
- The fringe is implemented as a priority queue FRINGE
  - INSERT(node,FRINGE)
  - REMOVE(FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy

13

Search Algorithm

SEARCH(Start, Finish)

1. INSERT(Start,FRINGE)
2. Repeat:
   a. If FRINGE is empty then return failure
   b. q ← REMOVE(FRINGE)
   c. If q = Finish then return a path from Start to Finish
   d. For every **new** state q' in SUCCESSORS(q)
      i. Install q' as a child of q in the search tree
      ii. INSERT(q',FRINGE)



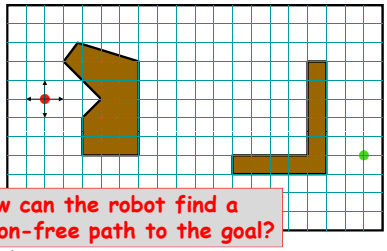
14

Blind Search Strategies

- Breadth-first
  New positions are inserted at the end of FRINGE What is the effect?
- Depth-first
  New positions are inserted at the beginning of FRINGE What is the effect?

15

Point Robot on a Grid

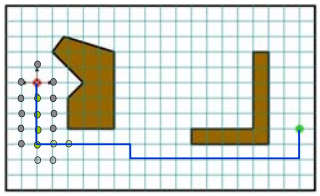


How can the robot find a collision-free path to the goal?

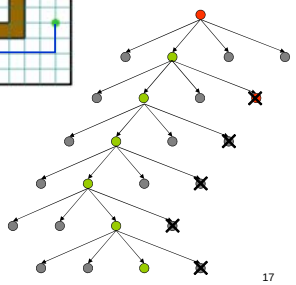
Assumptions:

- The robot perfectly controls its actions
- It has an accurate geometric model of the environment (i.e., the obstacles)

16

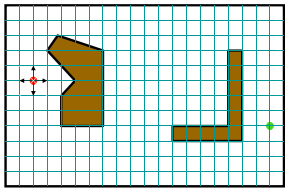


Search tree



Now, the robot can **search** its model for a collision-free path to the goal

17

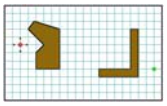


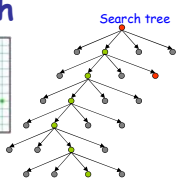
1000×1000 grid → 1,000,000 configurations
In 3-D → 10^9 configurations
In 6-D → 10^{18} configurations!!!

→Need for smart search techniques or sparse discretization

18

Smart Search



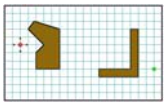


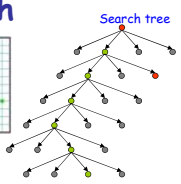
SEARCH(Start$_1$,Finish)
1. INSERT(q_{start},FRINGE)
2. Repeat:
   a. If FRINGE is empty then return failure
   b. q ← REMOVE(FRINGE)
   c. If q = Finish then return a path from Start to Finish
   d. For every **new** state q' in SUCCESSORS(q)
      i. Install q' as a child of q in the search tree
      ii. INSERT(q',FRINGE)

What can smart search be? How can it be done?

19

Smart Search





SEARCH(Start$_1$,Finish)
1. INSERT(q_{start},FRINGE)
2. Repeat:
   a. If FRINGE is empty then return failure
   b. q ← REMOVE(FRINGE)
   c. If q = Finish then return a path from Start to Finish
   d. For every **new** state q' in SUCCESSORS(q)
      i. Install q' as a child of q in the search tree
      ii. INSERT(q',FRINGE)

→ Smart ordering of the configurations in FRINGE
(best-first search)

20

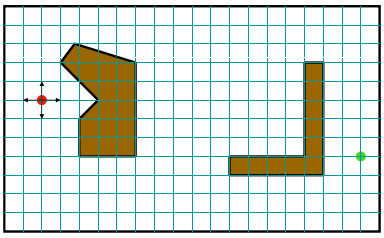
Evaluation function

- f : node N → real positive number $f(N)$
- For instance $f(N)$ may be the Euclidean distance (straight-line distance) to the goal
- Sort the fringe by increasing value of f

21

Application

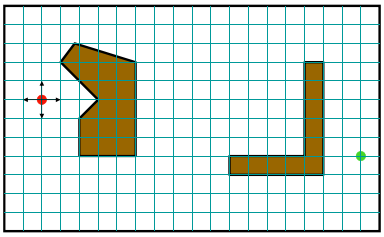
f(N) = Euclidean distance to the goal



22

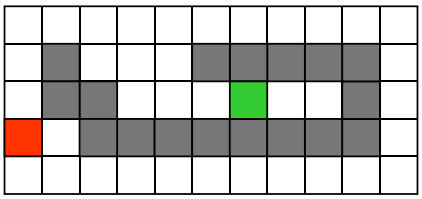
Application

f(N) = Manhattan distance to the goal



23

Another Example



24

Another Example

f(N) = h(N), with h(N) = Manhattan distance to the goal

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

25

Another Example

f(N) = Manhattan distance to the goal

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

26

Another Example

f(N) = g(N)+h(N), with h(N) = Manhattan distance to goal and g(N) = distance traveled to reach N

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

27

A* Search

$f(N) = g(N) + h(N)$, where $g(N)$ is the length of the path reaching N and **$0 \le h(N) \le$ length of the shortest path from N to goal** [h is called the **heuristic** function]

FRINGE is a priority queue sorted in increasing order of f

A*(Start,Finish)

1. Insert the initial-node (state = Start) into FRINGE
2. Repeat:
   a. If FRINGE is empty then return failure
   b. Let N be the first node in FRINGE, remove N from FRINGE
   c. Let s be the state of N and mark s as closed
   d. If s = Finish then return the path found between Start and Finish and exit
   e. For every state s' in SUCCESSORS(s)
      i. If s' is closed then do nothing
      ii. Else
         - Create a node N' with state s' as a successor of N
         - If there is a node N" in FRINGE with state s' then
           - If g(N") < g(N') then do nothing, else remove N" from FRINGE and insert N' in FRINGE,
         - Else insert N' in FRINGE

28

A* Search

$f(N) = g(N) + h(N)$, where $g(N)$ is the length of the path reaching N and **$0 \le h(N) \le$ length of the shortest path from N to goal** [h is called the **heuristic** function]

FRI

A*(

1.
2.

Both
h(N) = Euclidean distance from N to goal
and
h(N) = Manhattan distance from N to goal
verify $0 \le h(N) \le$ length of the shortest path from N to the goal

e. For every state s' in SUCCESSORS(s)

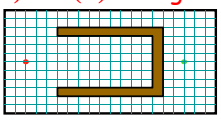
A* search is guaranteed to return a shortest path if a path exists

- If g(N") < g(N') do nothing, else remove N" from FRINGE and insert N' in FRINGE
- Else insert N' in FRINGE

29

How to Choose h?

- $f(N) = g(N) + h(N)$, where $0 \le h(N) \le$ length $h^*(N)$ of the shortest path from N to goal
- Would $h(N) \equiv 0$ be a good choice?
- Would $h(N) \equiv h^*(N)$ be a good choice?

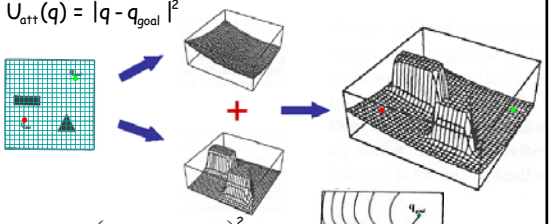


- Which one is better:
  $h(N)$ = Euclidean distance to goal?
  $h(N)$ = Manhattan distance to goal?

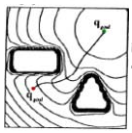
30

Attractive/Repulsive Potential Fields

$U_{att}(q) = |q - q_{goal}|^2$



$$U_{rep}(q) = \left(\frac{1}{d_{obst}(q)} - \frac{1}{d_{min}} \right)^2$$



Equipotential contours

31

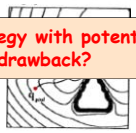
Attractive/Repulsive Potential Fields

$U_{att}(q) = |q - q_{goal}|^2$

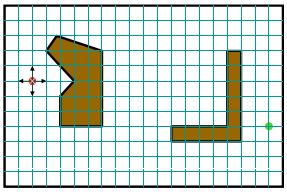
Best-first search with potential fields: Sort positions in FRINGE in increasing order of potential

What could be a Bug motion strategy with potential fields? What would be its drawback?

$\left(d_{obst}(q) \quad d_{min} \right)$



32

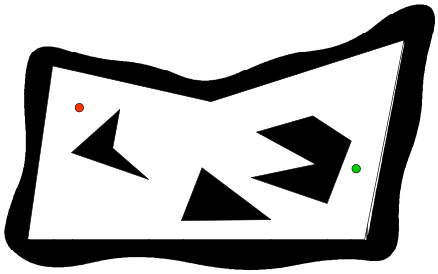


1000×1000 grid → 1,000,000 configurations
In 3-D → 10^9 configurations
In 6-D → 10^{18} configurations!!!

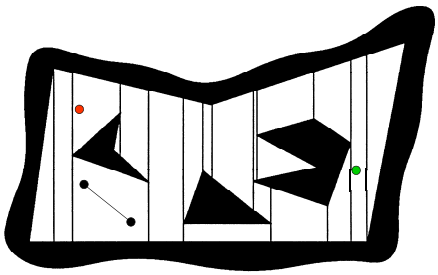
→Need for smart search techniques
or sparse discretization

33

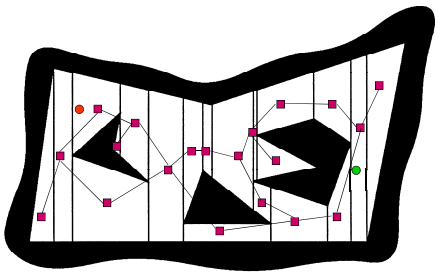
Cell Decomposition



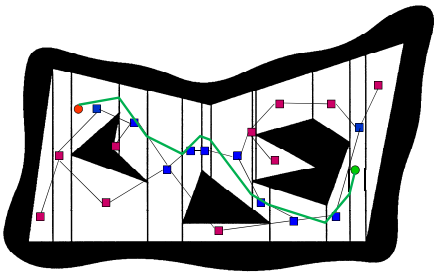
Cell Decomposition



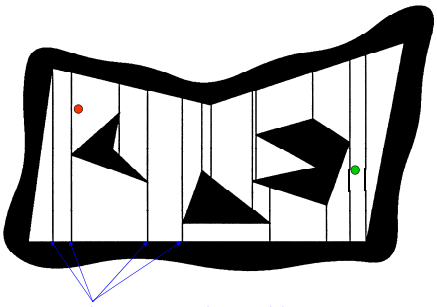
Cell Decomposition



Cell Decomposition

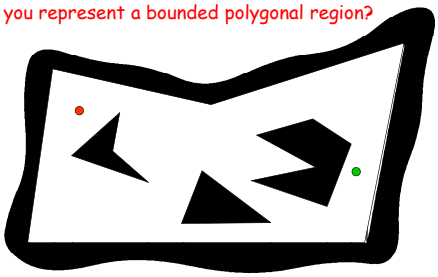


Cell Decomposition

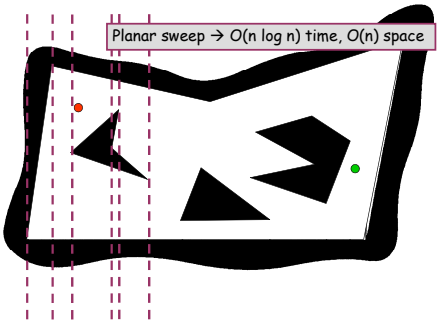


Cell Decomposition

How would you represent a bounded polygonal region?



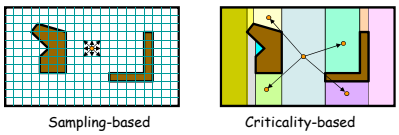
Cell Decomposition



Continuous space

↓

Discretization



↓

Search

41

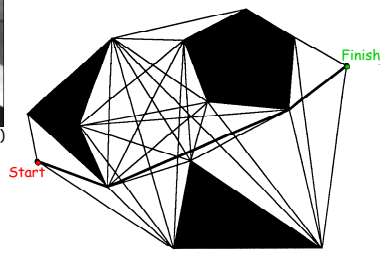
Tradeoffs

- Cell decomposition: precomputation of data structure
- Path planning: query processing for a given (Start, Finish) pair using this data structure
- Can the precomputation cost be amortized over several queries?
- What if the world changes frequently?

42

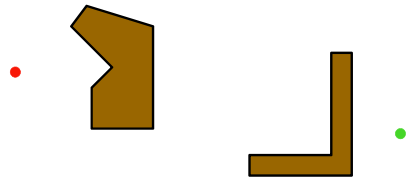
Visibility Graph

SHAKEY (SRI, 1969)



43

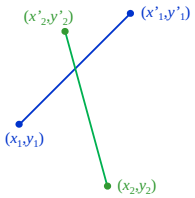
Visibility Graph



Computational complexity?

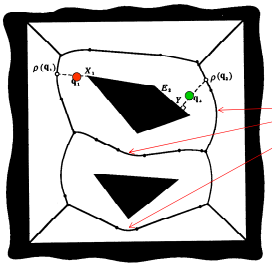
44

How would you check whether two line segments intersect?



45

Voronoi Diagram



What is this?

Advantages/drawbacks relative to visibility graph method?

46