LaValle Ch 2: Discrete Planning

1/19/18

For simplicity assume all models are completely known & predictable, and the number of possible states is finite.

2.1.1 Problem Formulation

x = state of robot

X = state space, set of all x.

u = action, e.g., move fwd 1 m, apply 1 Nm to joint 1.

U = action space = $\bigcup_{x \in X} U(x)$

$U(x)$ = action space when in state x.

$x' = f(x, u)$ = state transition function

- x': new state obtained as a result of action u. i.e. $x' \in X$
- f
- x: current state
- u: current action

Note $x' \in X$.

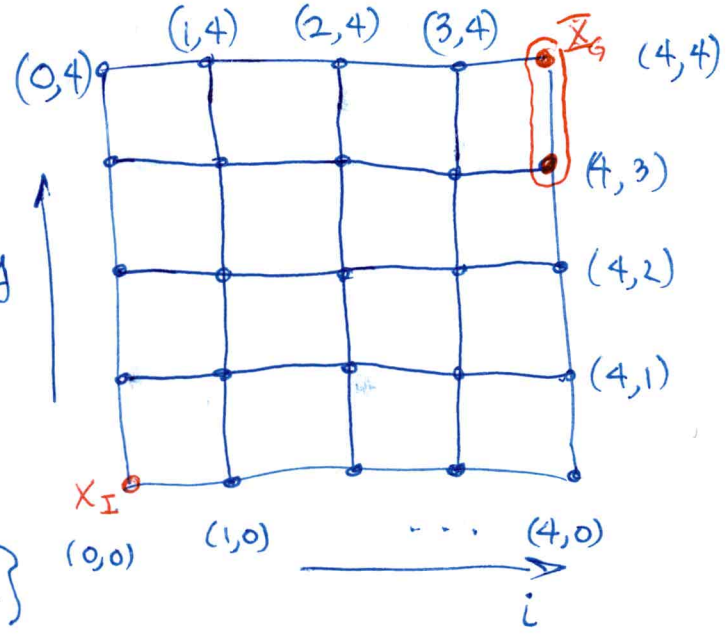
Example

$X = \{0,1,2,3,4\} \times \{0,1,2,3,4\}$

Cartesian product

$x = (i, j)$;

$i, j \in \{0,1,2,3,4\}$



$U = \{\text{right}, \text{up}, \text{left}, \text{down}\}$

$U((0,0)) = \{\text{right}, \text{up}\}$.

other bndry nodes are constrained too.

State transition function

$$f(x,u) = \begin{bmatrix} i \\ j \end{bmatrix} + \left\{ \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{R}, \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{U}, \underbrace{\begin{bmatrix} -1 \\ 0 \end{bmatrix}}_{L}, \underbrace{\begin{bmatrix} 0 \\ -1 \end{bmatrix}}_{D} \right\}$$

x

Choose one to predict outcome of action

Task definition

$X_I = (0,0)$

$X_G = \{(4,4), (4,3)\}$

Possible Solution Algorithms

Assume no obstacles

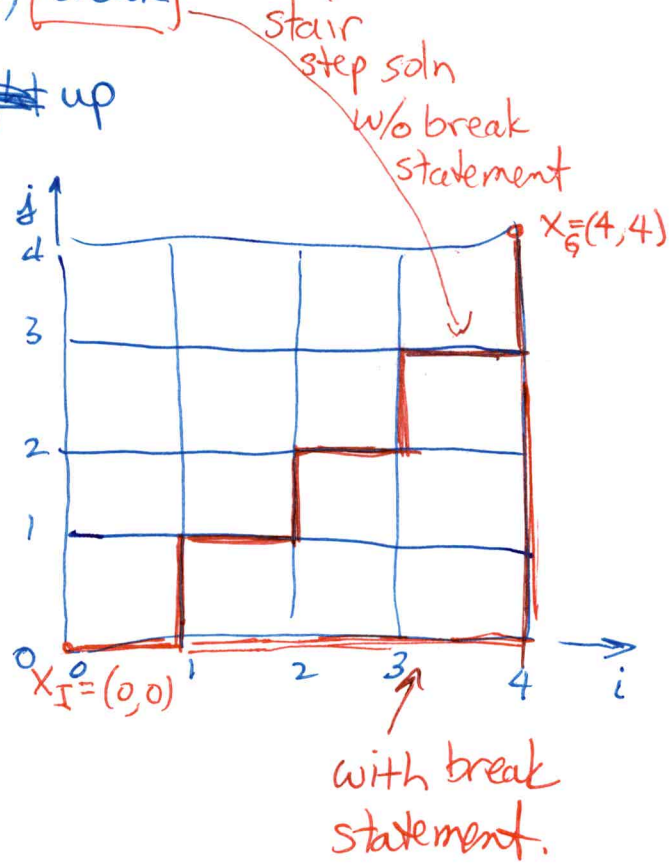
Alg 1

1. If $i < 4$, choose $u = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ← move right
2. If $j < 4$, choose $u = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ← move up.

More precisely

while $(i,j) \neq X_G = (4,4)$

- if $i < 4$, move ~~right~~, break
- if $j < 4$, move ~~right~~ up



Why so easy to solve?

- State space small
- State space structure known exactly
- State space simple structure.

Even a random walk alg will work

Alg 2

while $x \neq x_G$

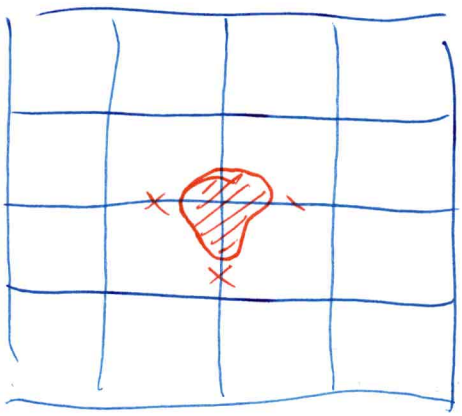
if $i<4$ && $j<4$, choose u = up or right at random

else if $i<4$, choose u = right

else choose u = up.

What if we change structure of X?

Suppose some arcs are missing and robot doesn't know it?!



- If soln exists, Alg 1 & Alg 2 will fail sometimes

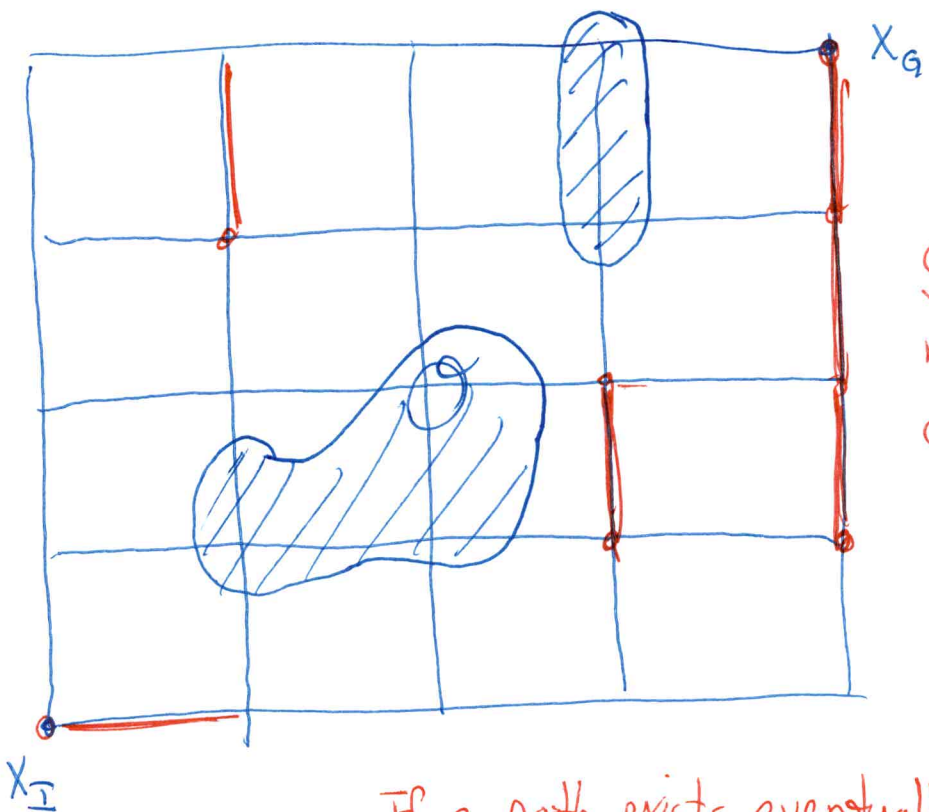
∴ Algs 1 & 2 are not complete & ∴ not desirable.

Alg 3 (next page) always succeeds (eventually), if a path exists.

Alg 3

1. while path not found
2. Choose node at random
3. connect to neighbor if possible
4. if x_I and x_G in same component of G, return solution
5. else, return to 1

G - graph.



G has multiple components.

If a path exists eventually one is found.

Note: Sampling must be uniform to ensure visiting all nodes eventually for this to work!

2.2: Searching for Feasible Plans

1/20/18

Robot planning algs are just graph search algs.

Key distinction - graph is not fully specified in advance, because that is too costly.

We want complete algorithms.
How do we get them?

Completeness is guaranteed if search is systematic

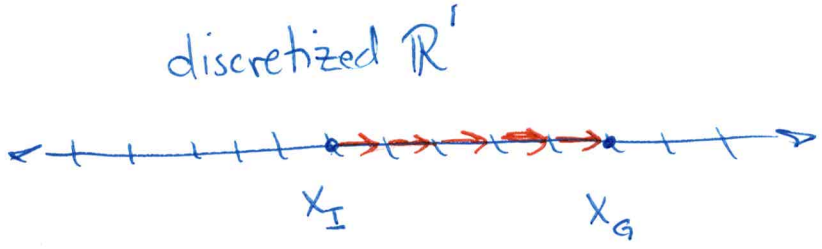
Systematic - as time $\to \infty$ every node will be explored and every action at each node will be tested. AND no redundant exploration (to avoid infinite loops).

To be most useful algs must run in finite time.

$\therefore$ It is common to use finite representations of infinite spaces.

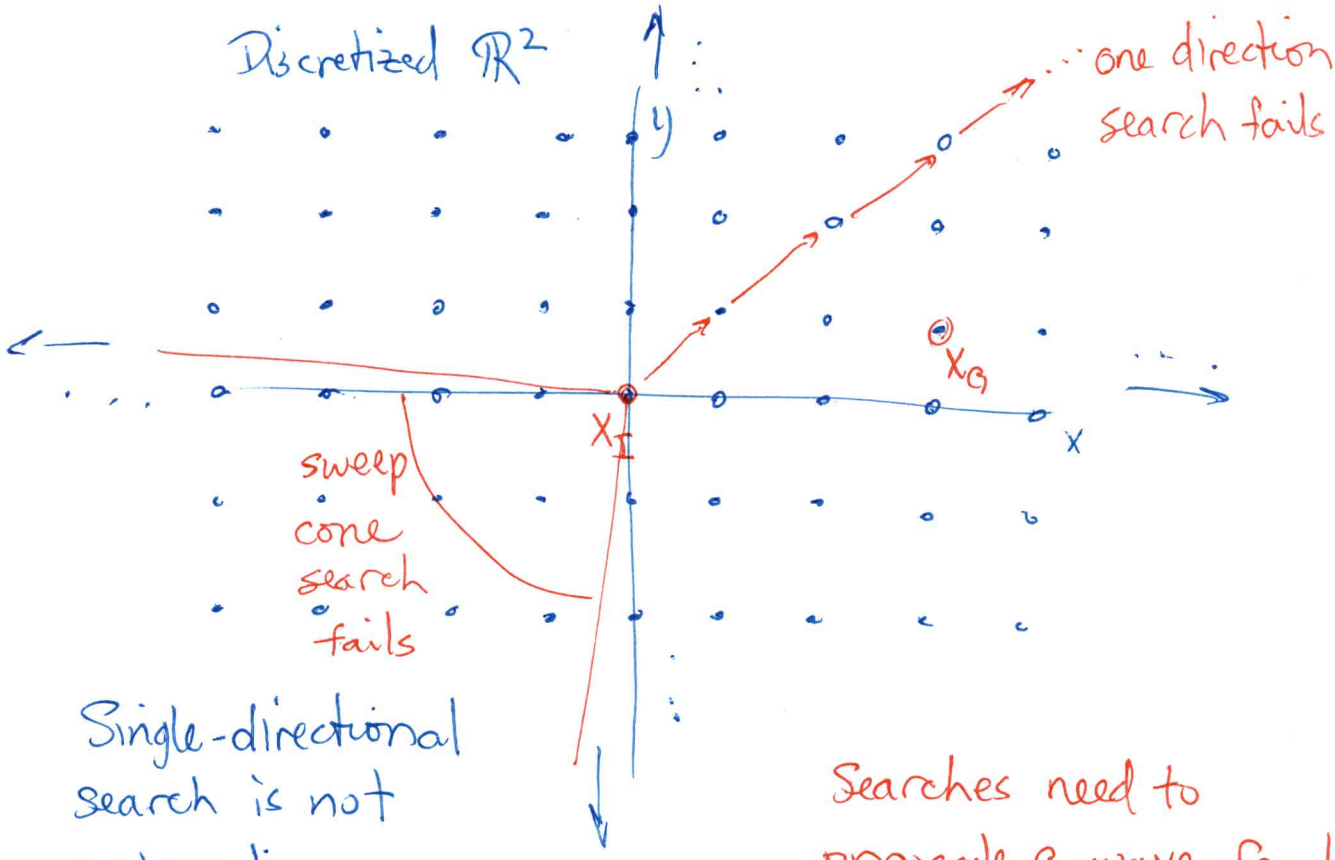
Note: If graph is countably infinite, we require systematic to mean soln found in finite time, but alg will run forever if soln does not exist.

Example of countably infinite graph.



If exploration is in one direction only, it is not systematic.

x_G to the left of x_I will never be found.



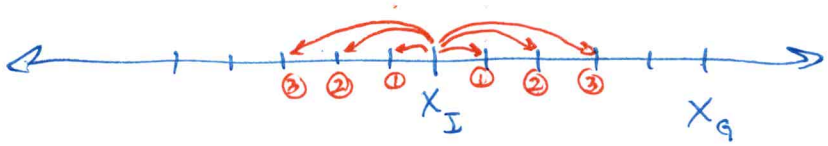
Single-directional search is not systematic.

Searches need to propogate a wave front.

1/20/18

(8)

1D wave front search



If a solution exists, it will be found in a finite # of steps.

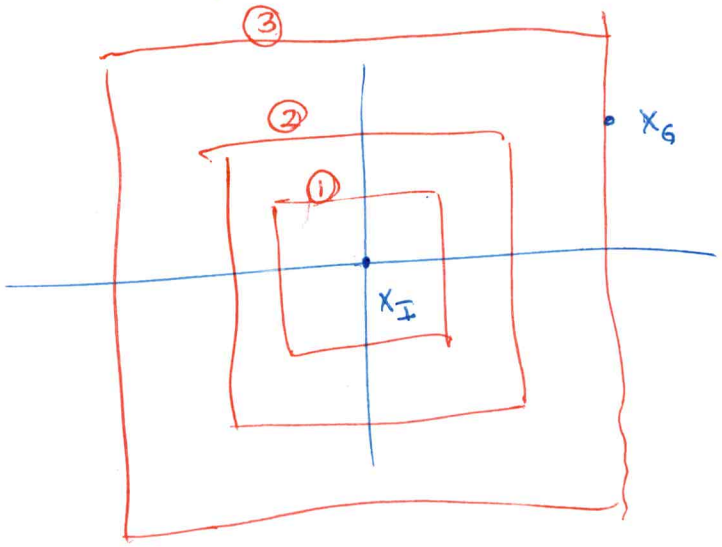
Suppose X_I is at $x=0$. and X_G is at $|x|=N$.

X_G will be found in $2N$ steps

soln exists → If $N<\infty$, alg terminates w/ soln in finite time (assuming each expansion requires finite time).

~~If N<∞,~~

2D wave front



If solution exists, one will be found in finite time.

2.2.1: General Fwd Search

1/20/18
(9)

We need 3 kinds of states (nodes):

1.) Unvisited: Initially all states except x_I.

2.) Dead: States that have been visited and for which every possible next state has also been visited. A next state of x is x' for which $u \in U(x)$ exists such that $x' = f(x,u)$.
All available info about solution has already been extracted.

3.) Alive: Visited states for which not all possible next states have been visited.

Dead states stored in list, D.
Alive states stored in priority queue, Q.

LaValle Ch 2:

Basic search (on graph) Algorithms

10/16/17

Let D be a list of explored nodes

Let Q be a priority queue, with first element being the most "promising"

```
D.init()      % Make empty list
Q.insert(x_I) & mark x_I as "visited"
while Q is not empty
    x ← Q.getFirst()
    if x ∈ X_G
        extract plan & return success
    for all u ∈ U(x)
        x' = f(x,u)
        if x' not visited
            mark x' as visited
            Q.insert(x')
        else
            resolve duplicate path to x'
    Q.remove(x)
    D.insert(x)
return failure
```

LaValle figure 2.4 (with addition of D)

(good hashing scheme or clever data structure helps)
must compare x' w/ all x∈D ∪ x∈Q
could be time consuming

set back pointer from x' to x & remember u.

update back ptr if path would be better.

Alg is in Fig 2.4 LaValle.

Some forward search algorithms

Breadth-first search ← Breadth-First is systematic!

Q is First-In, First Out (FIFO)

Search progression

D = {}
Q = {0}

---

x = 0
Q = {0, 1, 5}
D = {0}

---

x = 1
Q = {1, 5, 2, 6} ← don't insert 0, since 0 ∈ D
D = {0, 1}

---

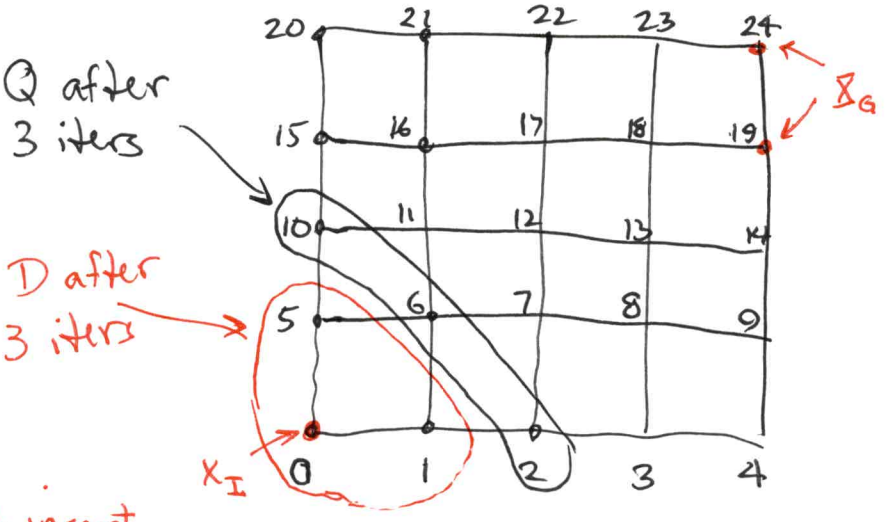
x = 5
Q = {5, 2, 6, 10} ← 6 inserted twice. In breadth-first, nothing to resolve for duplicate path just keep 1 copy
D = {0, 1, 5}

⋮

---

x = 14
Q = {14, 18, 22, 19}
D = {… 14}

Note: 18 & 22 will be processed before soln returned even though 19 ∈ X_G

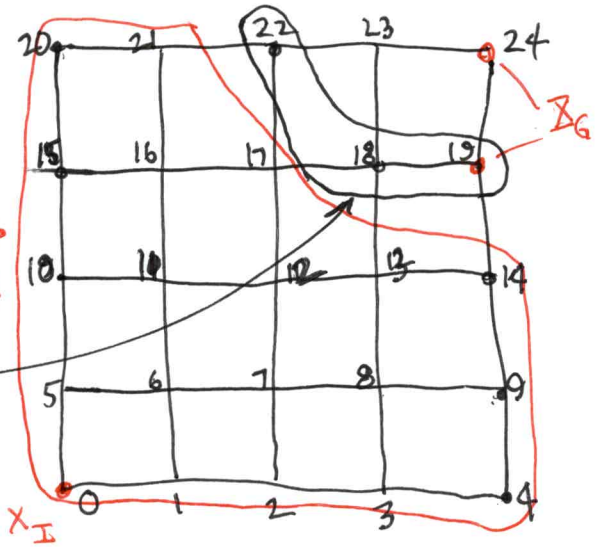


$X_I = \{0\}$

$X_G = \{19, 24\}$

U = {right, up, left, down}

constraint: can't leave graph



Worst-case running time

Let E & V denote the sets of edges & vertices.

Assume that basic operations can be checked in const. time.

Then running time is $\mathcal{O}(|V| + |E|)$

where $|\cdot|$ is cardinality of a set

$|V|$ = # distinct states

$|E| = \mathcal{O}(|U| \cdot |V|)$

In planar problems,

$|V| = n_x \cdot n_y$ = # of distinct states

where n_x is # of discretized values of x-coord.

n_y " " " " " " y-coord.

In problems with more d.o.f., $|V|$ grows exponentially with the number of d.o.f.

e.g. 7-jointed arm on quadrotor or ground vehicle

# of states, $|V| = n_{\theta_1} \cdot n_{\theta_2} \cdot n_{\theta_3} \cdot n_{\theta_4} \cdot n_{\theta_5} \cdot n_{\theta_6} \cdot n_{\theta_7} \cdot n_x \cdot \ldots$

$\ldots \cdot n_y \cdot n_z \cdot n_\alpha \cdot n_\beta \cdot n_\gamma$

If only 100 distinct values per dof assumed, then we have 10^{26} states!

This complexity motivate cell decomposition & sampling-based methods.

Depth-First Search ← systematic if $|\mathbb{X}| < \infty$

Q is Last In, First Out (LIFO).

Everything else works the same way.

$D = \{\ \}$

$Q = \{0\}$

$x = 0$

$Q = \{0, 1, 5\}$

$D = \{0\}$

$x = 5$

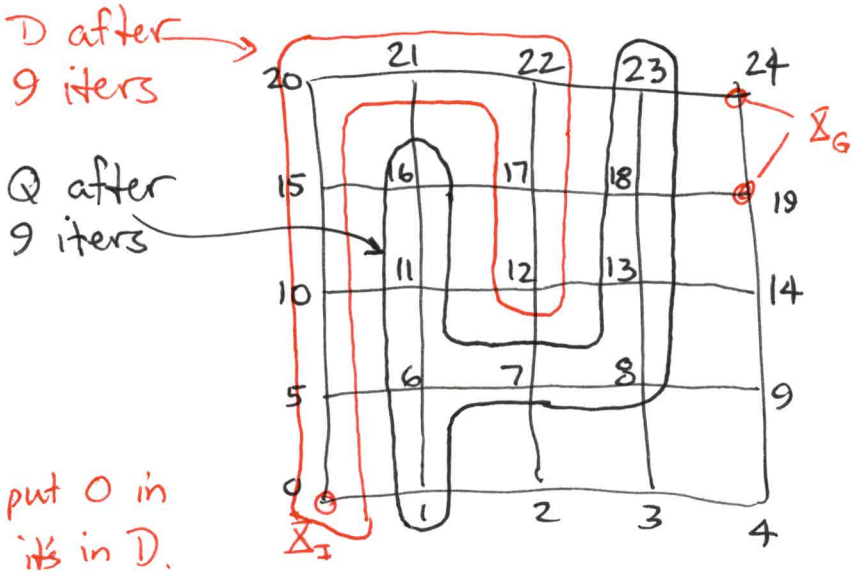
$Q = \{1, 5, 6, 10\}$ ← don't put 0 in since it's in D.

$D = \{0, 5\}$

$x = 10$

$Q = \{1, 6, 10, 11, 15\}$

$D = \{0, 5, 10\}$



Solution is found in 16 iterations.

Worst-case running time is $O(|V| + |E|)$

If grid were infinite, search would go in preferred direction & never return to x_I.

What path is returned? Could be D!

Dijkstra's Algorithm

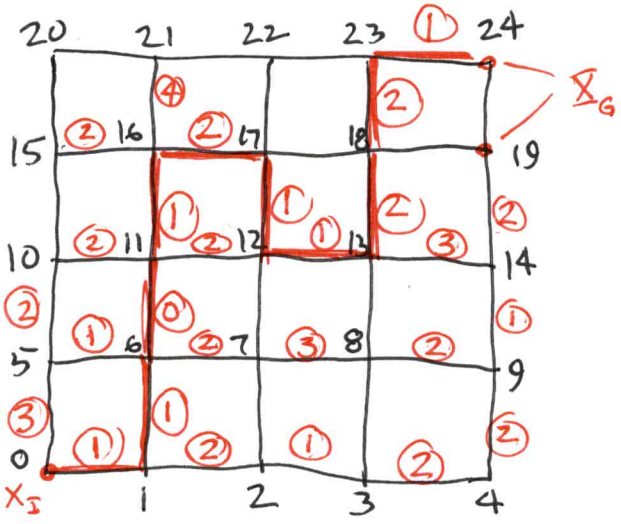
Good for finding optimal paths

Let $\ell(e)$ denote "cost" to traverse edge e, $e \in E$.

$\ell(x,u) = \ell(e) \geq 0$

Associate a cost with each link.

40 ~~cost~~ links for this simple graph



Let $C(x)$ = cost of path from X_I to X, a.k.a. cost-to-come

$C^*(x)$ = ~~minimum~~ optimal cost of path from X_I to X.

Q is sorted on $C(x')$

where $C(x') = C^*(x) + \ell(e)$

Note: If you want to allow negative weights, then use the Bellman-Ford algorithm. (B-F can identify negative cycles.)

Important: once $x \in D$, $C^*(x)$ has been computed!

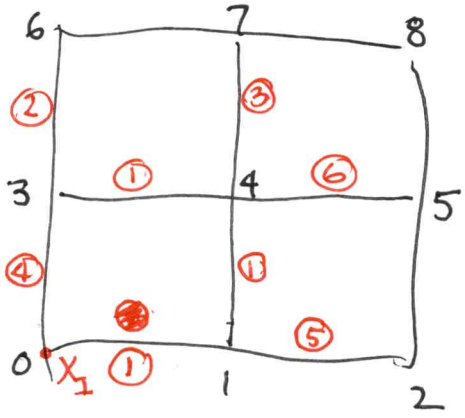
Resolve duplicate path to x'.

Initial condition
$c^*(0) = 0$

$D = \{\}$
$Q = \{0\}$

$x = 0$
$Q = \{0, 1, 3\}$
$D = \{0\}$

$C(1) = 1, \to 0$
$C(3) = 4, \to 0$



$x = 1$
$Q = \{1, 3, 2, 4, \cancel{0}\}$
$D = \{0, 1\}$

$C(4) = 2, \to 1$
$C(2) = 6, \to 1$
$C(0) = 2$ — but $C^*(0) = 0$ is given, so ignore this path. Also $0 \in D$, so don't consider

---

$x = 4$
$Q = \{3, 2, 4, \cancel{0}, 5, 7, 3, \cancel{4}\}$
$D = \{0, 1, 4\}$

$C(3) = 4$

$C(3) = 3, \to 4$
$C(5) = 8, \to 4$
$C(7) = 5 \to 4$

keep $C(3) = 3$ (not in D yet)
update ptr & cost, since smaller cost found

$x = 3$
$Q = \{3, 2, 5, 7, \cancel{4}, 6, \cancel{0}\}$
$D = \{0, 1, 4, 3\}$

in D

$C(6) = 5$

$0, 4 \in D$ so don't mess with them.

Optimal planning isn't free!

Worst-case running time is $O(|V| \lg |V| + |E|)$

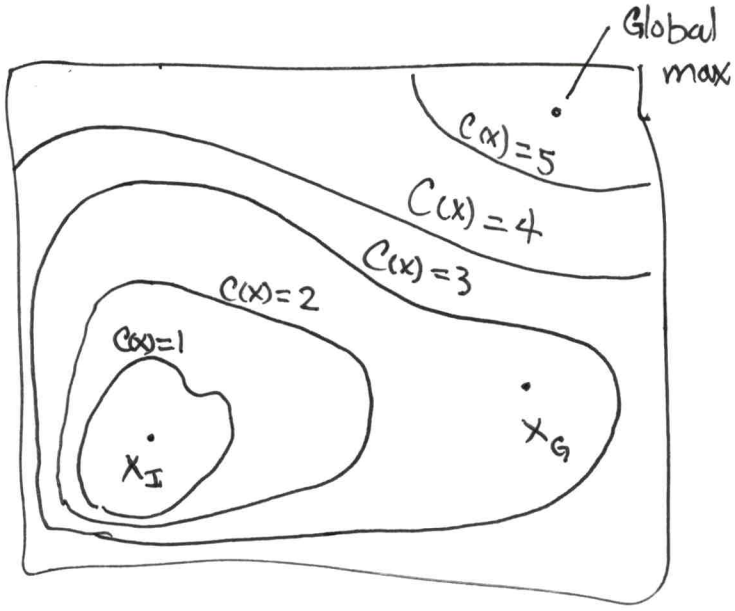
(Q must be implemented as a Fibonacci heap.)

Dijkstra's Alg is greedy w.r.t. cost.

It explores cheapest paths first.

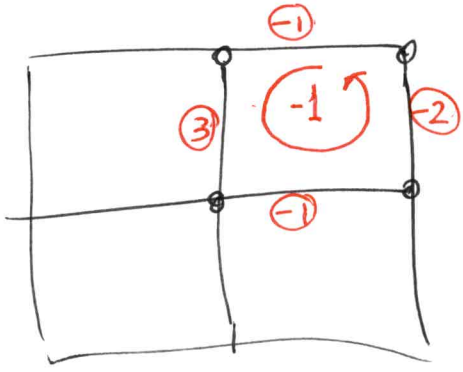
Topo map analogy

It will explore inside each iso-cost line before "spilling over" to the next region.



Aside: Bellman-Ford

Worst-case complexity of B-F is $\mathcal{O}(|E| \cdot |V|)$



If there are no negative weights, B-F gives same results as Dijkstra, but less efficiently.

Dijkstra is $\mathcal{O}(|E| + |V| \log |V|)$

A* Algorithm

A* attempts to reduce # of states visited by using an optimistic cost-to-go heuristic.

Let $G(x)$ = cost-to-go

$G^*(x)$ = optimal cost-to-go
(can't know this in advance)

$\hat{G}^*(x)$ = under-estimate of optimal cost to go ← required to guarantee path optimality

Only change to above algorithms is that Q is sorted on $C^*(x) + l(x,u) + \hat{G}^*(x')$

Worst heuristic is $\hat{G}^*(x') = 0$. Then A* = Dijkstra

Perfect heuristic is $\hat{G}^*(x') = G^*(x')$. Then A* is depth-first "straight" to X_G

Example:
Walking in NYC.
Let cost be path length.
$\hat{G}^*(x)$ = Euclidean distance between current location & goal.

Best-First Search

Sort priority queue on $\hat{G}^*(x)$.

$\hat{G}^*(x)$ does not need to be an under-estimate.

Paths found are not necessarily optimal.

Iterative Deepening

Make depth-first search systematic.

Good candidate on graphs with high branching factors

Idea:

Use depth-first ~~search~~ limited to i steps away from x_I.

If x_G not found, throw away old work and start from scratch limiting "depth" to $i+1$ steps from x_I.

Once alg has committed to depth += 1, work lost is negligible.

```
set depth to 1
while x_G not found
    ~~set limit to~~ i
    Constrained DF search
    if x_G found
        extract path
    else
        depth += 1
```

2.2.3: Backward search

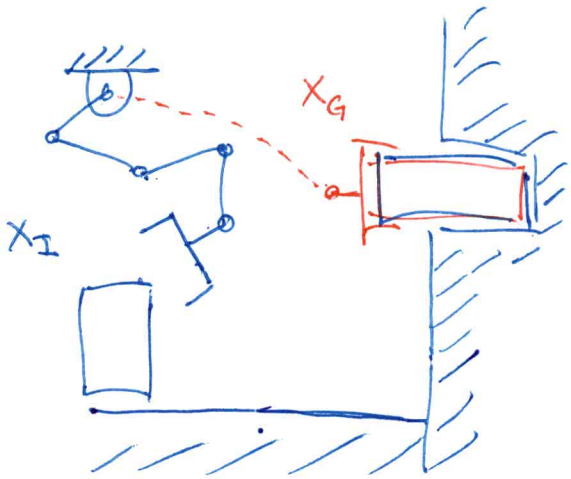
1/21/18

(14)

If branching factor is high near x_I
and low near x_G, then
it may be more efficient to search from $x_G \to x_I$.

Example:

- Only one direction to move from x_G.
- Many from x_I.



We have this already

Fwd Search

$x' = f(x,u)$

$x \xrightarrow[f(x,u)]{u \in U(x)} x'$

We want something like this

Bkwd Search

$x = f^{-1}(x', u^{-1})$

$x \xleftarrow[f^{-1}(x',u^{-1})]{u^{-1} \in U(x')} x'$

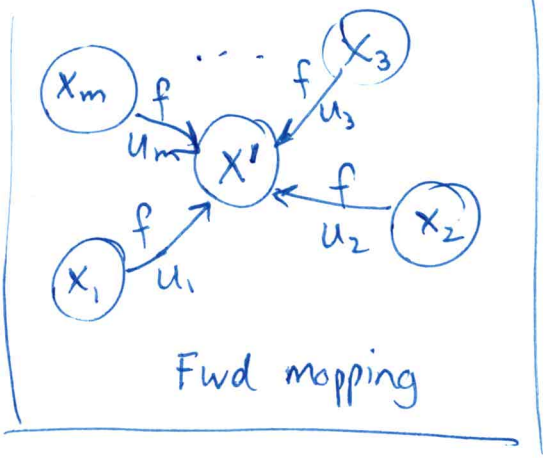
~~Note: f^{-1} is not an inverse function in the usual mathematical sense!~~

Let $\mathcal{U}^{-1} = \{(x,u) \in \mathbb{X} \times \mathcal{U} \mid x \in \mathbb{X}, u \in \mathcal{U}(x)\}$

↑ all state-action pairs, which can be considered the domain of f.

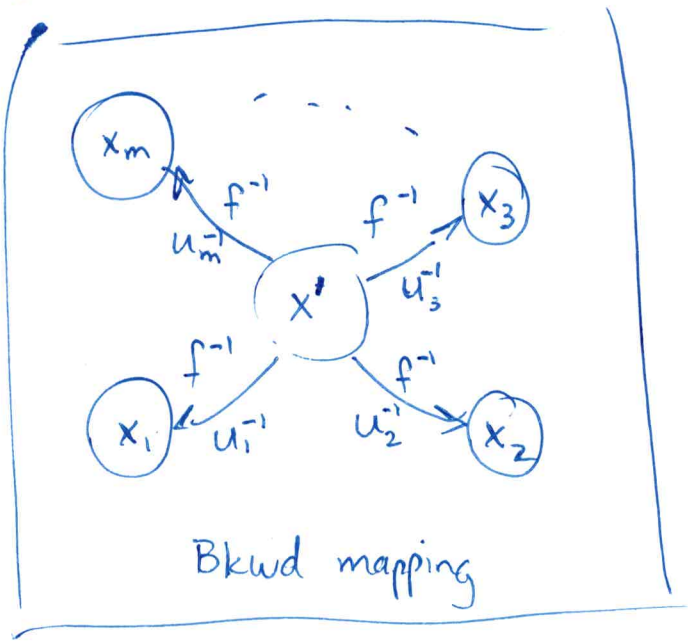
Let $\mathcal{U}^{-1}(x') = \{(x,u) \in \mathcal{U}^{-1} \mid x' = f(x,u)\}$ ← backward action space

Denote $(x,u) \in \mathcal{U}^{-1}$ as $u^{-1} \in \mathcal{U}^{-1}$



Let f^{-1} denote the backward state transition function

$x = f^{-1}(x', u^{-1})$



If f^{-1} is defined all fwd search algs can be run bkwd w/o changes!

Adapting Fwd Search Alg

1/21/18

(16)

swap x_I & x_G

swap x & x'

replace u with u^{-1}

replace f with f^{-1}

replace U with U^{-1}

initialize Q with all $x_G \in X_G$

rather than x_I

see Alg in Figure 2.6 LaValle

Bidirectional search

1/21/18

(17)

Q_I.insert(x_I), mark x_I as visited

Q_G.insert(x_G), mark x_G as visited

while Q_I not empty AND Q_G not empty

FWD

if Q_I not empty

$x \leftarrow Q_I$.GetFirst()

if $x \in Q_G$

return success

for all $u \in U(x)$

usual processing from fwd search

BKWD

if Q_G not empty

$x' \leftarrow Q_G$.GetFirst

if $x' \in Q_I$

return success

for all $u^{-1} \in U^{-1}(x')$

usual processing, but reversed

Return FAILURE

General Steps of Search in Robotics

1/21/18

(18)

(1) Initialize graph $G(V,E)$.

$V = \{x_I\}$ or $\{x_G\}$ or $\{x_I, x_G\}$ or more!

$E = \emptyset$ (empty)

There can be more than 2 trees

(2) Select vertex for expansion

(3) Apply actions to find new states (or to connect to existing states)

(4) Insert edges into trees

(5) Check for solution

Can be expensive if 2 or more trees!

(6) Return to step (2).

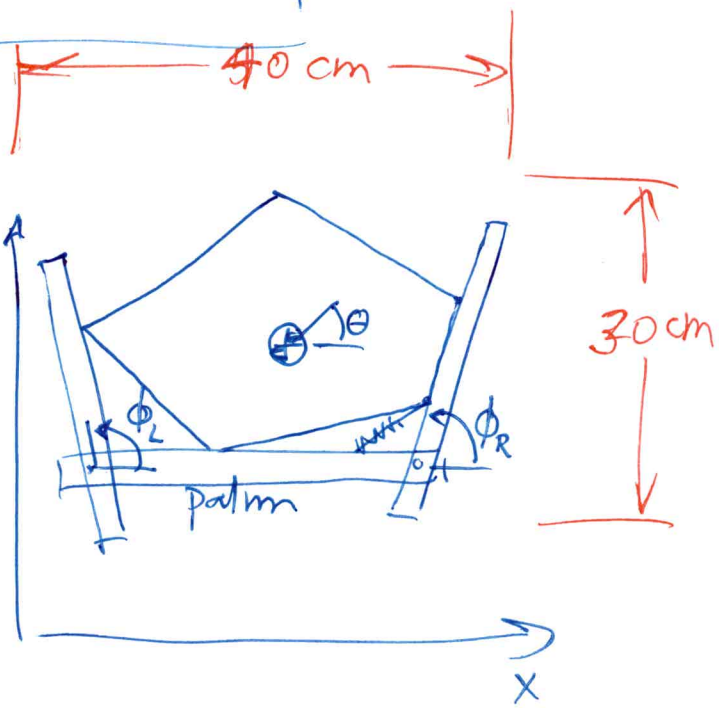
2.3 Discrete optimal planning

Skip for now. Return if necessary.

1/20/18

Complexity reduction by representation (and modeling and control). (10)

Show in-hand manipulation video



x_I = initial grasp

$(x, y, \theta, \phi_L, \phi_R)_I$

x_G = goal grasp

$(x, y, \theta, \phi_L, \phi_R)_G$

$X = \mathbb{R}^2 \times S^3$

5 dimensional

Discretize state space accurately, because contact vs no contact make a big difference in outcome of actions.

$U = \{ \phi_{L,min} \le \phi_L \le \phi_{L,max}, \ \phi_{R,min} \le \phi_R \le \phi_{R,max} \}$?

$\simeq$ 1 Radian $\simeq$ 3 Radians $\simeq$ 0 Rad $\simeq$ 2 Rad

1/20/18

(11)

How will we represent a plan?

Perhaps $\phi_L(t)$, $\phi_R(t)$ $\quad 0 \le t \le T$

Should these be continuous functions?

Could the controller execute them accurately?

Should we discretize state & including ϕ_L & ϕ_R ?

Say 0.01 R for angles
0.1 cm for transl

Size of state space

$400 \times 300 \times 628 \times 200 \times 200 = 3.0144 \times 10^{12}$

HUGE

(Minimum reasonable branching factor is 10 (2 * # DOF).

Suppose each number is double precision - 8 bytes

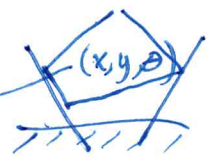
∴ The full graph requires ≃ 24 terabytes

Very inefficient!

What can be done to reduce this?

Can we use physics to reduce dimension of representation?

ie. Given ϕ_L & ϕ_R, can we compute ?



Can we limit the problem to simplify?

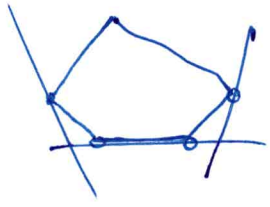
Can we improve the controller?

Computer used was RS/6000
with ~64 MB RAM.
Very slow risc processor.
Solved problems in 3-5 minutes!

Assume: no throwing.
no dynamics (just quasistatic)
no friction

Always at least two contacts,

but we can control it to
maintain 4 contacts.



Maintaining 4 contacts removes
4 dof from state space.

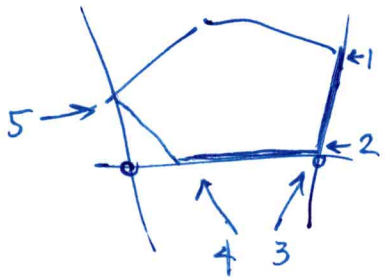
Controller need to be hybrid.

ϕ_L, ϕ_R position controlled? ← not good

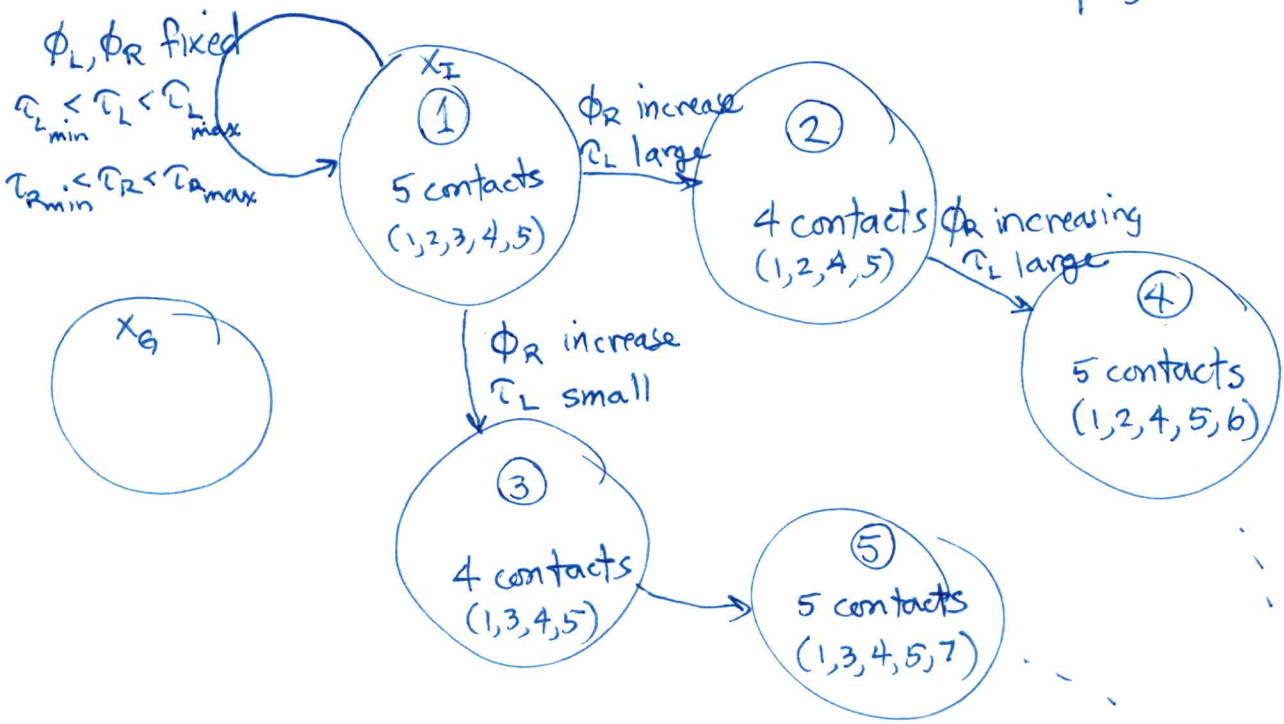
or ϕ_L, τ_R

or τ_L, ϕ_R } ← better

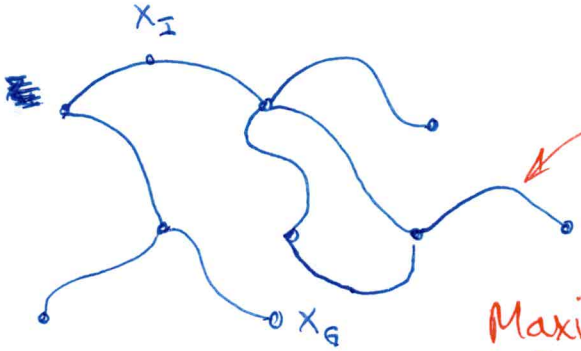
$\mathcal{X}$ is still 5D, but we only have to try to explore a 1D subspace



Graph - $\mathcal{G}$ - looks like



View in $\mathcal{X}$ (projected to $\mathbb{R}^2$)



RAM save space by no discretization

Maximum branching factor is $\binom{5}{4} = 5$