LaValle Ch 5.4: Incremental Search & Sampling

2/25/18

General Framework

(1) Initialization: $G(V,E)$ as (undirected) search graph, where E is empty list of edges and V is a non-empty set of pts in C_{free}.

$V = \{q_I, q_G, \text{and possibly some other points}\}$

(2) Vertex Selection Method (VSM):

Choose element of V for expansion; $q_{cur} \in V$

Plays a role similar to priority queue in Section 2.2

(3) Local Planning Method (LPM):

Main difference to search from Ch2. Action is replaced by a path.

For some $q_{new} \in C_{free}$ (possibly not in V) attempt to construct a path $\tau_s : [0,1] \to C_{free}$ such that $\tau(0) = q_{cur}$ and $\tau(1) = q_{new}$.

If connection fails (due to collision) go to step 2.

(4) Insert edge in G: Insert τ_s into G connecting q_{cur} to q_{new}. If $q_{new} \notin V$, insert it.

⑤ Check for Solution: Determine whether ② G encodes solution path. If G is a tree, this part is trivial. Otherwise not trivial.

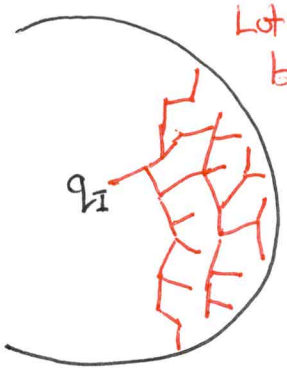
⑥ Return to step 2: If ~~no~~ solution found in ⑤ or stopping criterion has been met, stop. Otherwise return to step ②.

Recall, G is a topological graph, ie., every $q \in V$ is in C_{free} and every $\tau \in E$ is in C_{free}.

LPM is "local" planning method since it only operates close to q_{cur}.

Many possible planners can be created by pairing different methods for VSM and LPM

Avoiding Local Minima



Imaging guiding search with heuristic as done in A* or Best-First. Also imaging beginning w/ high-resolution grid for V. Then much time wasted in local min.

2/25/18
(3)

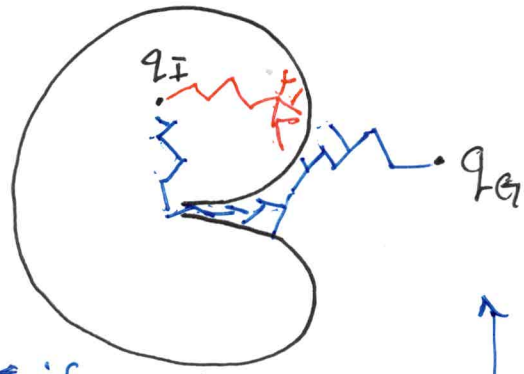
Classes of algorithms base on number of search trees

Unidirectional (single-tree) methods:

Very similar to discrete search methods in Chapter 2.

Could possibly be easy to solve with a single tree searching backward, if greedy search pushes search toward q_{ap}.

Difficult to solve w/ fwd search tree.



Bidirection (two trees):

Idea is that search frontiers of two trees will meet with less exploration than would occur when using one tree.

VSM alternates between the trees to choose q_{cur}.

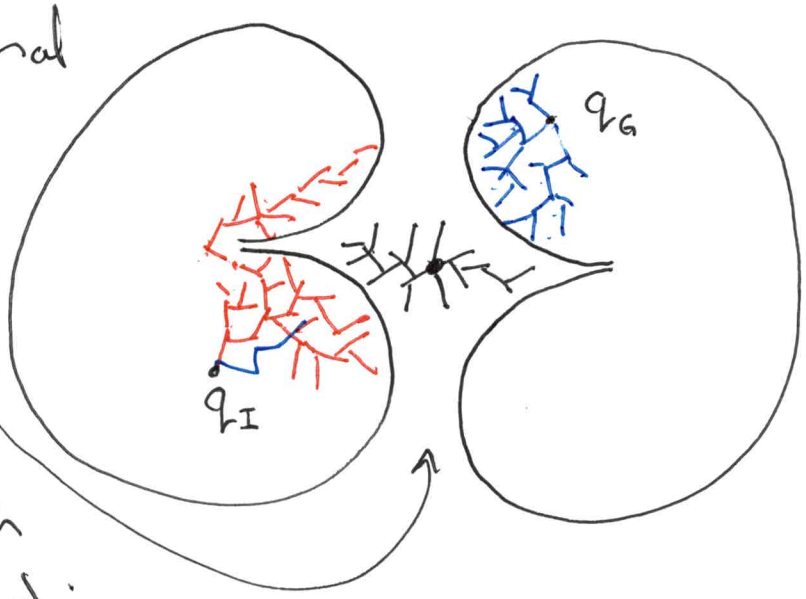
LPM " " connecting the trees and growing the trees by choosing $q_{new} \in$ other tree or q_{new} not in a tree.

2/25/18

④

Trouble for bi-directional search.

Maybe an extra tree can help.



Multi-directional search (more than two trees):

- More trees could help find paths into traps, but interleaving seach and tree connection becomes more complicated.

One has to accept that some problems just won't be solvable by sampling-based methods unless some structural info about C-space is available!

5.4.2 Adapting Discrete Search Algs

2/25/18
⑤

Map all joint motions & other dof's onto $[0,1]^n/\sim$

identifications where needed

revolute jnts that rotate 2π

quaternions $h = -h$

$$q_{i,min} \le q_i \le q_{i,max} \quad \forall\, i = 1, \ldots, n$$

where n = # of dof.

Discretization: grid on $[0,1]^n/\sim$

$$\Delta q_i = [0 \cdots 0 \; 1/k_i \; 0 \cdots 0]$$

where k_i is # of grid cells in direction i.

Grid points are expressed as $\sum_{i=1}^{n} j_i \Delta q_i = q$

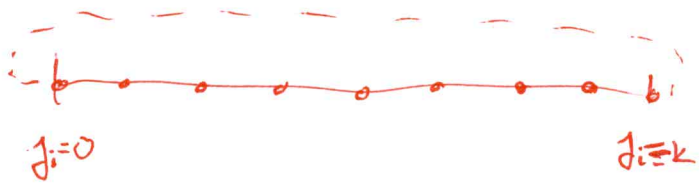
where $j_i \in \{0, 1, \ldots, k_i\}$ so that

$j_1, j_2, \ldots, j_n$ are array indices

of the grid point, q.

For dimensions that are identified at $0 \& 1$,

then points with $j_i = 0$ and $j_i = k_i$ are equal.



2/25/18

(6)

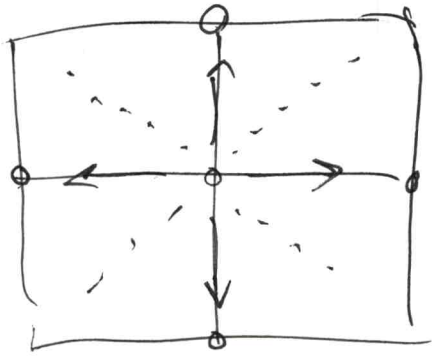
Neighborhood

The 1-neighborhood is:

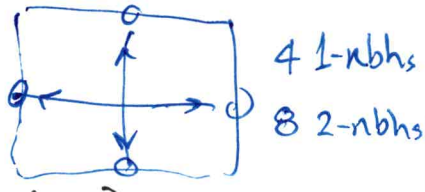
$$N_1(q) = \{q_1 + \Delta q_1, \cdots, q + \Delta q_n, q - \Delta q_1, \cdots, q - \Delta q_n\}$$

There are at most $2n$ valid ~~neighbors~~ 1-neighbors.

Using 1-neighbors keeps # of neighbors to check in search alg. linear in dimension of prob.



If neighbors included diagonal neighbors, then # of neighbors increases (~~which~~ ~~exponentially~~ ~~quadratically~~ quadratically) with dimension.



The 2-neighborhood is:

$$N_2(q) = \{q \pm \Delta q_i \pm \Delta q_j \mid 1 \le i, j \le n, i \neq j\} \cup N_1(q)$$

LaValle says q has $3^n - 1$ neighbors in an nthhbhd — LaValle eq. (5.38)

→ How many 2-neighbors?

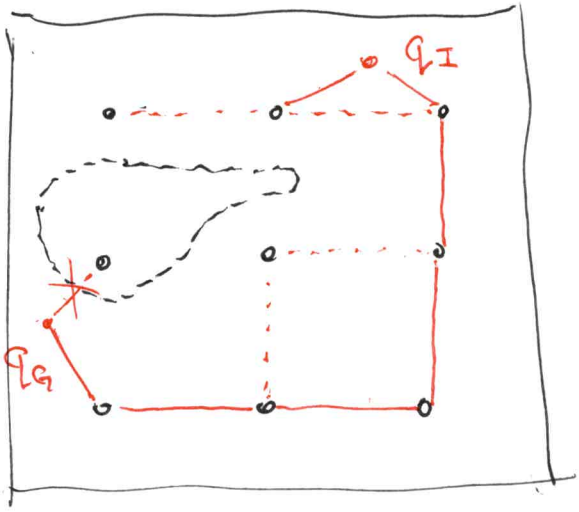
$q \pm \Delta q_i \pm \Delta q_j \rightarrow 4$ How many ways to choose $i \neq j$? $n(n-1)$

Total #2neighbors = $4(n(n-1)) = \mathcal{O}(n^2)$.

Obtaining a discrete planning problem

2/25/18

⑦



- Grid defines V
- Use LPM to connect q_I & q_G to nearest points on grid
- Not all edges need to be explored (necessarily) → RDT or RRT algorithms (section 5.5)
- All vertices & edges could be explored in advance to yield a roadmap (section 5.6).

Grid resolution issues

Too much grid resolution slows planning

Too little grid resolution misses solutions.

Two ways to deal w/ this problem:

- Interleave search & grid refinement
- Abandon grid/discrete search ~~with~~ using algs designed for the continuous problem

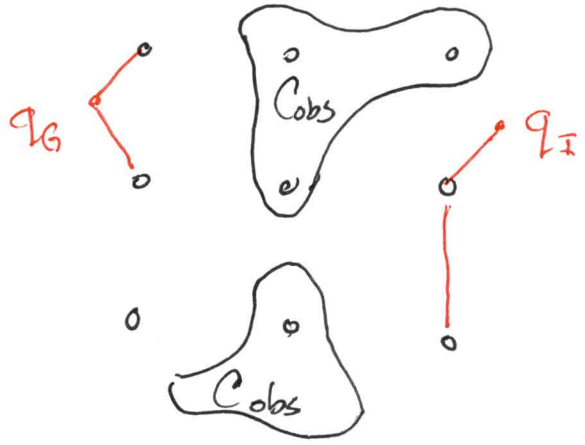
grid

Interleaved search and refinement

2/25/18

(8)

- Start with 2 points per dimension
- search
- double resolution

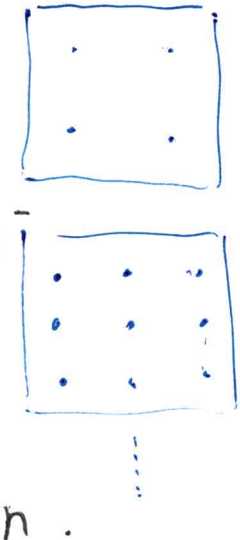


Problem is that # pts grows too rapidly if problem dimension is high, ~~$O(2^{rn})$~~ where n is dimension of C-space.

resolution

dimension

$O((2^r)^n)$



Better approach would be like iterative deepening -

use 2^n pts, search, toss results

then 3^n pts, search, toss, 4^n pts, ...

Amount of work lost is negligible for large n.

Possibly better. Increase resolution in one dimension at a time. (set of nearest neighbors changes), then search

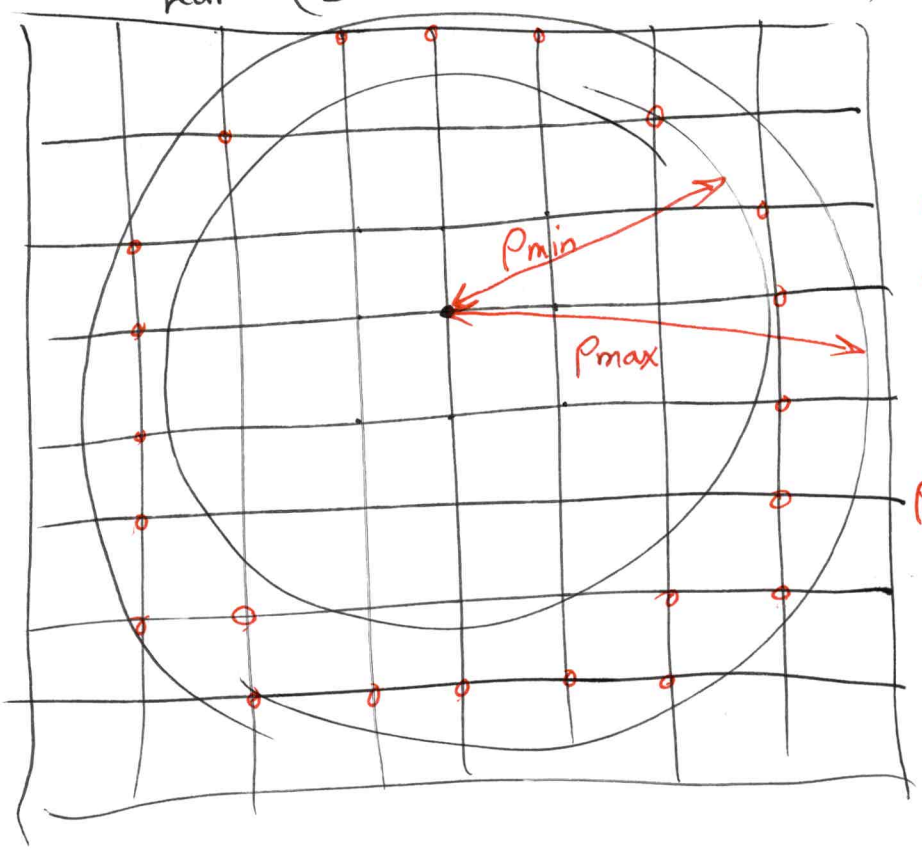
Maybe best. Keep track of connected components of $\mathcal{G}$ w/ addition of every point. Search for path only when q_I & q_G in same component of $\mathcal{G}$.

Previous grid discussion approach tries to keep |V| small.

Opposite approach.

Declare outrageous resolution (100") at start & then hope search (e.g., A*, Best-First) has good heuristic to avoid visiting many points!

One possible approach: LPM selects "neighbor" far from q_{cur}. (I've not seen this done.)



Could

Perhaps choose neighbors such that

$\rho_{min} < \rho(q_{cur}, q_{new}) \neq \rho(q_{cur}, q_{new}) < \rho_{max}$

Could do something like simulated annealing.

ρ_{min} & ρ_{max} large.

If solution progress slows, make ρ_{min} & ρ_{max} smaller

Could also use large Δ's in one-neighbor selection.

Best-first & A* could solve w/o visiting many points and ∴ might not have to create a large $\mathcal{G}$.

One way to guide BestFirst is by constructing and artificial potential field.

VSM
Chooses current point

LPM
follows gradient

Potential Fields

(show transparency)

Force field pushes robot away from obstacles

Designed to be global minimum

Attractive force field at q_G

Local Minima

Very hard to design potential field w/o local minima and global minimum at goal.

"navigation functions" by Koditschek applies to special geometries.

It would be helpful to know shape of C_{obs} to

design potential field, but that won't be available

From Latombe,

"Motion Planning"

1992.

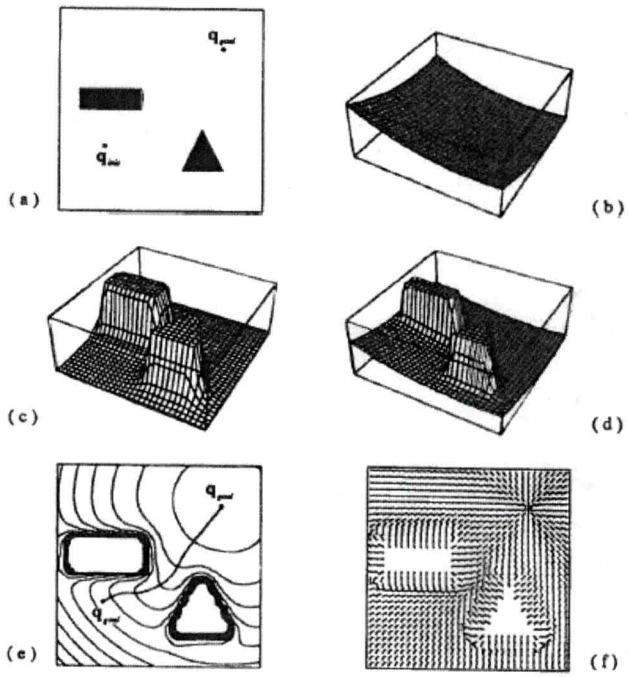
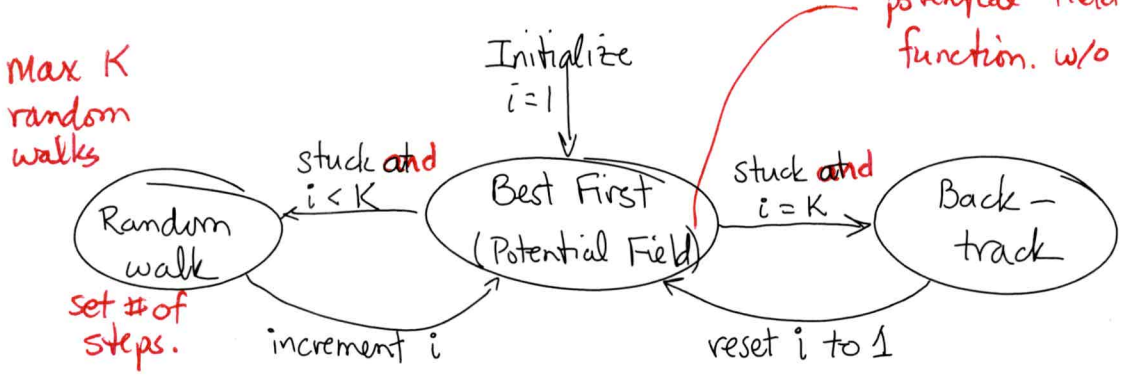


Figure 8. This series of diagrams illustrates the potential field approach. A simple two-dimensional configuration space with two polygonal C-obstacles depicted in Figure a. Figure b shows an attractive potential generated by goal configuration q_{goal}. Figure c shows a repulsive potential generated by the C-obstacles. Figure d shows the sum of the two potentials. Figure e displays equipotential contours of the total potential and a path obtained by following its negated gradient. Figure f shows orientations of the negated gradient of the

Randomized Potential Field Method

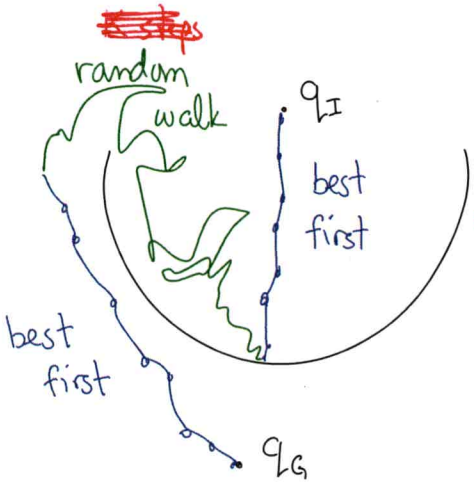


Follow best first search

random walk ; q_I

Follow best first search until all neighbors make negative progress.

Switch to random walk for a "while."



Random walk until
1) cost is improved
or 2) N steps performed

If K walks fails
Start Best First from random $q \in V$

Key

You must know how to find neighboring configurations!

This is a vote in favor of grids.

Questions:

How do we design good pot. fields?

Value of K?

what resolution?

From which node should walk begin?

More detail

Let $g = g_r + g_a$
(g_r: repulsive, g_a: attractive)

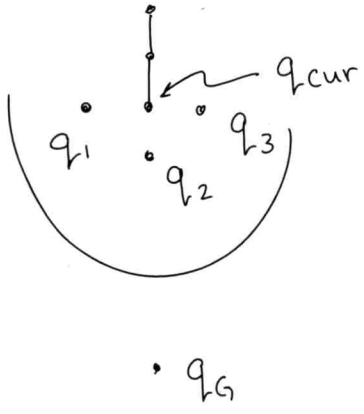
e.g. $g_r = a/\psi_n(q) = a\frac{1}{dist}$

$g_a = b\|q - q_G\|$

Let $g(q)$ denote the potential of q.

Currently at q_{curr}

Move next to q_i ∋

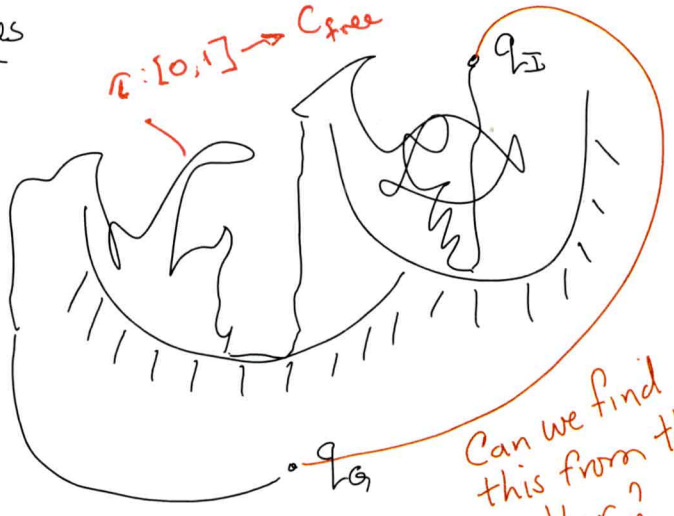


$g(q_i) < g(q_{cur})$

If $g(q_i) \geq g(q_{cur}) \; \forall i$, then best first is stuck.

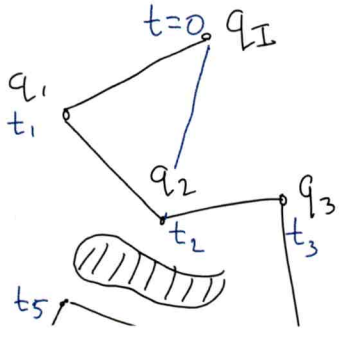
Random walks make very bad paths!

Smoothing issues

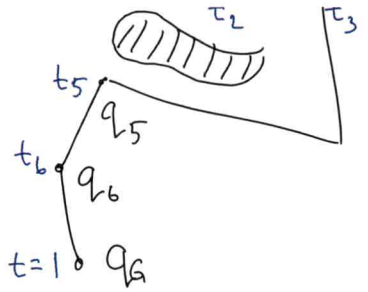


One way:
Recursively smooth the path, τ,
is to pick pairs of points along the path & connect them with a straight line in C. Then the new path is:

$$\tau' = \begin{cases} \tau(t) & 0 \leq t \leq t_1 \\ a\tau(t_1) + (1-a)\tau(t_2) & t_1 \leq t \leq t_2 \\ \tau(t) & t_2 \leq t \leq 1 \end{cases}$$



Try $q_I \rightarrow q_2$. If successful, cut out q_1 (and cost-to-come lower)



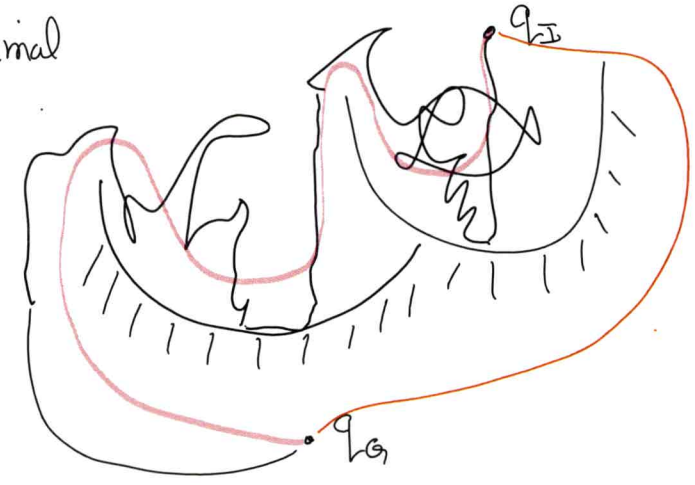
What if we tried $q_1 \rightarrow q_6$?

we could potentially cut out q_2, q_3, q_4, q_5

The order of pairs to connect is not cut and dry. A path metric would be helpful. What's important?

# of turns, energy expended, path length, ...

Suppose we suspect that our smoothed path is from a sub-optimal equivalence class (homotopy class)?



Maybe the red one is better.

You could continue searching until you believe you have a path from every homotopy class, but this would be very hard to determine definitively. Determining this is likely to be as hard as computing Cobs exactly.

In general much parameter tuning is need for good performance. And one tuning may not work well for all problems.

Other methods

Ariadne's Clew ~~Use 2 graphs G_I & G_G~~

Key idea: interleave search & exploration

Algorithm:

"e" ⇒ exploration

① VSM – choose vertex q_e in G at random

② LPM – find new point q_{new} maximally far from ~~q_e~~ all vertices in G. that can be "easily" connected.

③ Try to connect q_{new} to the other component of G

Drawback – finding q_{new} is a difficult optimization problem. Optimization method have to be tuned to each motion planning problem.

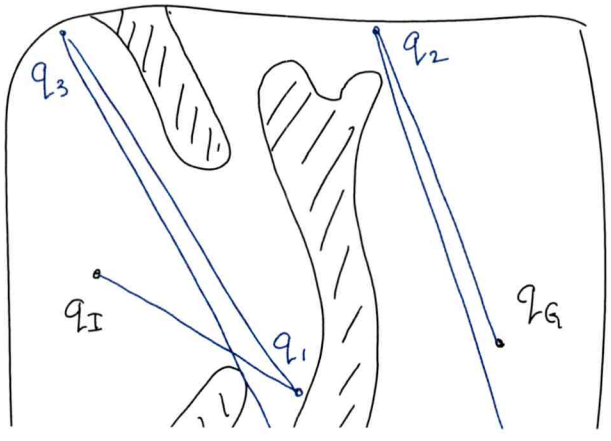
Init w/ ~~q_I~~ & q_G

~~Attempt connect (q_I, q_G) fail.~~

~~Choose $q_e = q_I$~~

~~find q_1~~

~~Connect (q_1, q_G) = fail~~



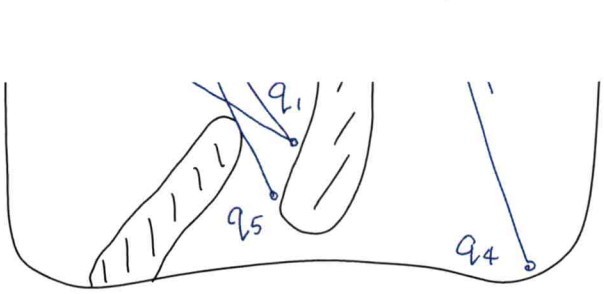
Connect (q_1, q_G) = fail

Choose $q_e = q_G$

Find q_2

Connect (q_2, q_I) = fail

Choose $q_e = q_1$, Find q_3, Connect (q_3, q_G) = fail.



Alg fails in this example, since only 5 new q's will ever be found. That is choosing any $q_e \in S$ will find q_{new} already in S.

Expansive space planner

Goal: generate samples in unexplored regions of C_{free}

① VSM – select q_e from S w/ probability inv. proportional to its # of connected neighbors

② LPM – expand q_e to get q_{new} w/in a nbhd of q_e

③ Insert q_{new} into S w/ probability inv. prop. to # of neighbors of q_{new}

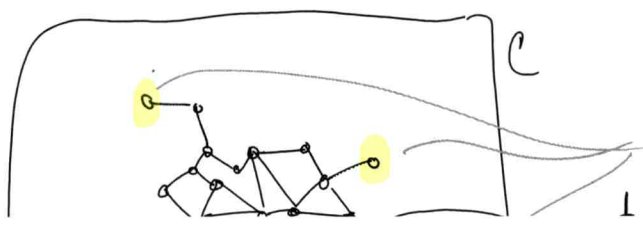
← Don't let the graph get high degree.

Three parameters to tune.

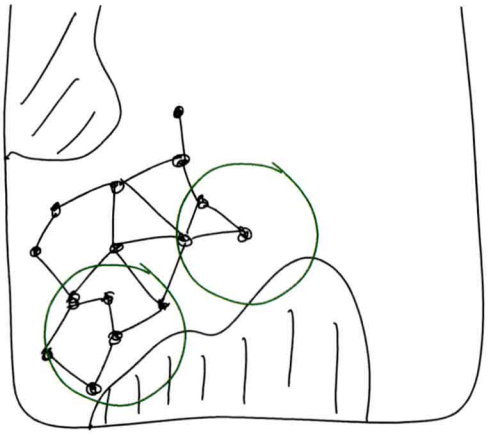
Sizes of nbhds important:
- Too small yields uniform prob. over S
- Too large " " " " S.

What's best?

} Size of nbhd should be tuned to feature resolution

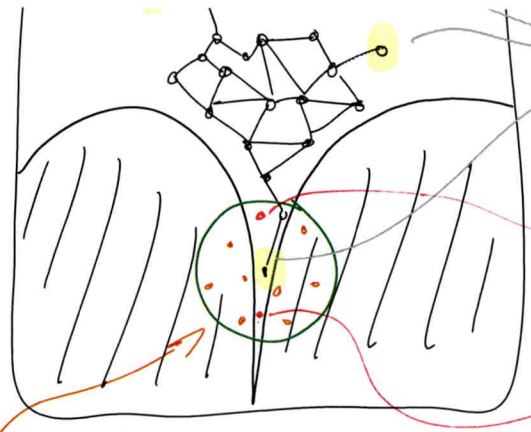


most likely



Bias search to explore from nodes on bndry of graph.

Best performance if size of nbhd was tuned to sizes of feature of C_{obs}, but this is unknown.

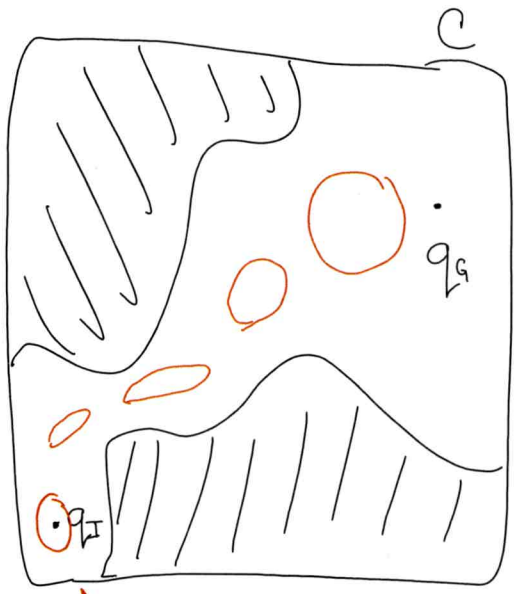


Bias search from node with little hope of success. Could keep failed points, but that would be bad if there was a narrow passage.

Random Walk Planner

Choose new points from a pdf that adapts to C_{obs}.

For example new points could be drawn from a multivariate Gaussian with changing mean and covariance matrix



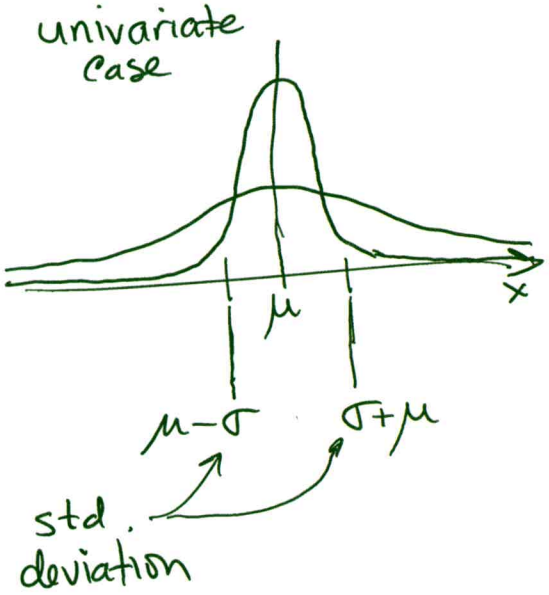
As search progresses in the example, the pdf narrows & aligns in passages and becomes less

narrows & aligns in passages and becomes less biased in wide open areas.

Recall the definition of a multivariate gaussian

$$f(q) = \frac{\exp\left(-\frac{1}{2}(q-\mu)^T \Sigma^{-1} (q-\mu)\right)}{2\pi^{N/2} \, |\Sigma|^{1/2}}$$

where μ is mean of p.d.f.
Σ is covariance matrix
N is length of vector q
$|\Sigma|$ is determinant of Σ



$$\mu = \sum_{i=1}^{k} x_i \Big/ \frac{1}{k}$$

Use moving average to create changing mean over time.

$$\Sigma'_{ij} = \sum_{i=1}^{k} \sum_{j=1}^{k} (x_i - \mu)^T (x_j - \mu)$$

summation

not summation, unfortunate choice of variable name

This method also has difficulty w/ winding narrow passages.

So you could use a sum of gaussians, but would be more complicated to compute the distribution