5.4 Incremental Searching & Sampling

Single query search algorithm.

Key steps

1. Initialize $G(V,E), E = \emptyset, V = \{q_{1}, q_{a}\}$

   $V$ could contain many samples

2. VSM. Choose $q_{cur} \in V$

3. LPM. Attempt to connect $q_{new}$ to $q_{cur}$.

   ($q_{new}$ not necessarily in $V$.)

   Create path $c: [0,1] \rightarrow C_{free}$

   $c(0) = q_{cur}, c(1) = q_{new}$

4. Insert edge if step 3 succeeds

5. Check for solution. (Complicated if multiple graphs or trees)

6. If no solution found, go to step 2.

   If after many iterations, no soln has been found,
   add more samples to $V$ and continue,
   or declare failure.

Questions:
Choose grid points up front?
   How many? What resolution?
   exponential growth with dimension of C

Which points should I attempt to connect next?
   Points far from each other? Nearby points?

Hard to choose all points up front, so consider
   interleaving sampling & searching for connections
   Let \( \alpha \) be a sequence of points dense on \( C \).
   Maybe try searching after each 100 samples?
   Maybe search every time the # of components
   of \( G \) drops? (Initially every point is a separate
   component of the graph, \( G \)).

Another option: Choose an unreasonably high # of
   points, and count on heuristic to solve w/o
   visiting a large fraction of the points.

Difficult problems:
Problems living on complicated varieties.
e.g. parallel manipulators

Problems living in high-dimensional spaces

e.g. dynamic systems

Problems with dynamics and contact avoidance can be solved using something like dVC as the LPM.

Use forces of contact in controller to
deflect robot away from contact

If graphs are not too large, one can solve probs. with alg in section 2.2.

\( G \) is small if:
- dimension of \( C \) is low
- resolution is low
- heuristic is very good

E.g. Suppose the robot has:
- 6 dof (flying robot)
- 7 dof (some industrial arms)

If we want 100 points per dof:
\[ \Rightarrow 10^6, 10^4 \text{ points}. \]
That's a big graph! What about humanoid robots w/ 50+ joints?

Another high-dimensional system
- 24D C-space
- maybe velocity matters too!

\( \text{quadcopter} \)

\( \text{Payload} \)
Best-first * A* could solve w/o visiting many points and :: might not have to create a large G.

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

Potential Fields
(show transparency)

Force field pushes robot away from obstacles

Designed to be global minimum

Attractive force field at $q_s$

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


Randomized Potential Field Method

Figure 8. This series of diagrams illustrates the potential field approach. A simple two-dimensional configuration space with two polygonal C-obstacles is depicted in Figure a. Figure b shows an attractive potential generated by the 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 total potential.
Follow best first search until all neighbors make negative progress.

Switch to random walk for a "while."

**Key**

You must know how to find neighboring configurations!

This is a vote in favor of grids.

**Questions:**

How do we design good potential fields?

Value of $K$?

What resolution?

From which node should walk begin?

**More detail**

Let $g = g_r + g_a$

where:

- $g_r = \frac{1}{\psi_n(q)}$
- $g_a = \|q - q_a\|$

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

Currently at $q_{\text{curr}}$

Move next to $q_i \in \{q_1, q_2, q_3, q_4\}$.
\[ g(q_i) < g(q_{\text{cur}}) \]

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

Random walks make very bad paths!

Smoothing issues

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

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

Try \( q_1 \to q_2 \). If successful cut out \( q_1 \).
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 $C^0$'s exactly.
In general, much parameter tuning is needed for good performance. And one tuning may not work well for all problems.

Other methods

Ariadne’s Clew

Key idea: interleave search & exploration

Algorithm:
1. VSM - choose vertex \( q_e \) in \( G \) at random
2. LPM - find new point \( q_{new} \) maximally far from \( q_e \) that can be “easily” connected.
3. Try to connect \( q_{new} \) to the other components of \( G \)

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

Init w/ \( q_I \) & \( q_{I6} \)
Attempt connect \( (q_I, q_{I6}) \)
- fail.
Choose \( q_e = q_I \)
- find \( q_e \)
Connect \( (q_I, q_{I6}) = \) fail
- return \( q_I \).
Connect \((q_1, q_2)\) = fail
Choose \(q_e = q_5\)
Find \(q_2\)
Connect \((q_2, q_4)\) = fail
Choose \(q_e = q_1\), Find \(q_3\), Connect \((q_3, q_4)\) = 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 \(\mathcal{C}\)

1. **VSM** - select \(q_e\) from \(S\) w/ probability inv. proportional to its \# of neighbors
2. **LPM** - expand \(q_e\) to get \(q_{new}\) w/ a \(\text{nbhd}\) of \(q_e\)
3. Insert \(q_{new}\) into \(S\) w/ probability inv. prop. to \# of neighbors of \(q_{new}\)

Three parameters to tune:

Sizes of \(\text{nbhds}\) important:
- Too small yields uniform prob. over \(S\)
- Too large " " " " \(S\),
  - What's best?

\{ Size of \(\text{nbhd}\) should be tuned to feature resolution \}

\[ \text{most likely I'd choose} \]
Bias search to explore from nodes on bdry of graph.

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 Costs.

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

Recall the definition of a multivariate gaussian

\[ f(q) = \frac{\exp(-\frac{1}{2}(q-u)^T\Sigma (q-u))}{2\pi^{N/2} |\Sigma|^{1/2}} \]

where \( u \) is mean of pdf.
\( \Sigma \) is covariance matrix
\( N \) is length of vector \( q \)
\(|\Sigma|\) is determinant of \( \Sigma \)

\[ u = \frac{\sum_{i=1}^{k} x_i}{k} \]

\[ \Sigma' = \sum_{i=1}^{k} \sum_{j=1}^{k} (x_i - \mu)(x_j - \mu) \]

not summation, unfortunate choice of variable name

This method also has difficulty w/ winding narrow passages.