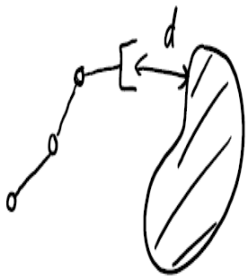
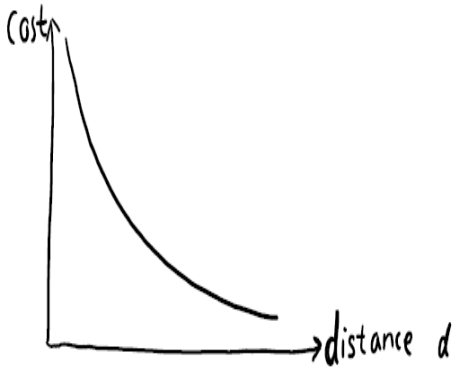
Planning With Costs

1. Define configuration space cost c: C --> R
   C[path] = cost of a path
   $\text{path} = \arg\min_{paths} C[path]$

   Want larger d to decrease cost (stay away from obstacle)

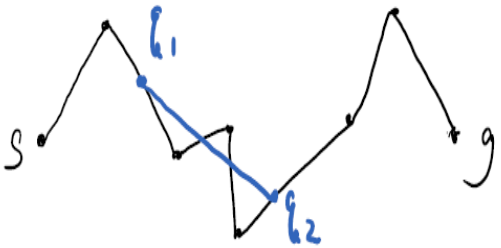


2. Cost Based Path Shortening

2.1 Original Algorithm

Connect q1 and q2 use simple planner. If succeed return it, else try again

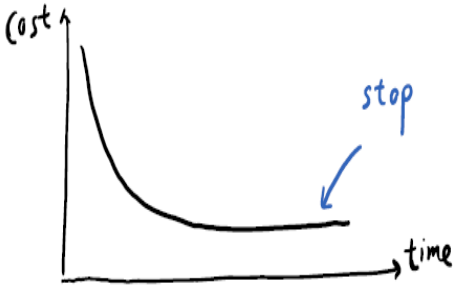



2.2 Algorithm 1:Rejection sampling

Accept only if $C[l_{new}] < C[l_{old}]$

Terminate conditions:

a. When path length < x
b. Try for one more second
c. Rate of improvement close to zero




2.3 Algorithm 2

Take random q1, q2 as a try

Mediapolis: Accept cost increases sometimes

Probability of accepting the path: P

P = 1 if $\Delta C < 0$

$$P \propto \exp\left[\frac{-\Delta C}{T}\right] \; if \; C \geq 0$$

T :Temperature, increase T -> increase P

2.4 Algorithm 3:Simulated Annealing

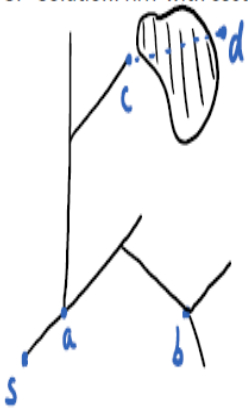
Start with high temperature (prone to explore)

Explore space early on

Reduce Temperature later

Flaw for Algorithm 1-3:RRT has no notion of cost

3. Solution: RRT with cost basis




C(a,b):Cost of path in the tree from a to be

h(c,d): heuristic cost from c to d

Heuristic:

- optimistic estimate (e.g. straight line)
- Cost less than optima cost C*(a,b)

Admissible: h(a,b)<=C*(a,b)

**Heuristically Guided RRT**

Node quality measure mq

$$C_q = c(s,q) + h(q,g)$$

Where c(s,q) is the current cost

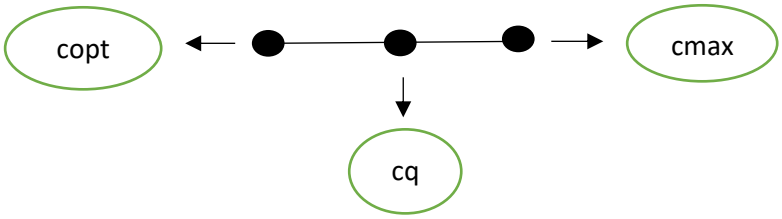
h(q,g) is the optimistic path to goal

$$c_{max} = max c_q$$

This is the worst value for the cost.

$$C_{opt} = h(s,g)$$

This is the best case value of the cost thus the optimal value.



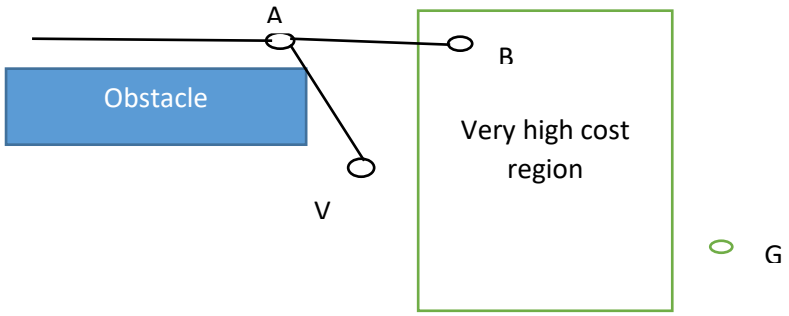

$$m_q = 1 - (c_q - c_{opt})/(c_{max} - c_{opt})$$

**Algorithm**

1. Sample qs
2. q ← NN(qs)
3. Compute mq
4. p = max(mq,pmin) Around 0.1 pmin is chosen
5. Accept if rand<p

Reject and resample

Consider the following example:




There is a very low probability of finding the point V.

A solution for this would be to consider the k nearest neighbours.

# Anytime RRT

The idea of this algorithm is gradually growing to the hard problem, that means at any time, it can give the reasonable solution at that time.

**Definition:**

Quickly find a sub-optional plan and improve it with time. At each iteration i: produce a path of cost Csi. Then, the next iteration i+1, of finding a path to the goal, is terminated only if Csi+1 < (1 − ε ) Csi = Ctarget , where ε is called the sfsf improvement factor.

Therefore, the RRT keeps growing and searching only when the new cost becomes lower than the previous cost by the improvement factor. the algorithm uses a novel node sampling, node selection and node extension technique.

**Node Sampling**

Rather than randomly sampling across all of the configuration space, the algorithm sample from those areas of the configuration space which could possibly provide a solution which satisfies the restriction. If the shortest path h(s, qs )+h(qs , q) ³ Ctarget , reject.

**Node Selection**

For selecting the node in the tree to extend towards the new sample, we define the Selection Cost of a node as:

SelCost(q) = db.Distance(q, qs) + cb.C(s, q)

Where cb is cost bias, $d_b$ is distance bias. Initially db = 1 and cb = 0 so that the node selection works just like the nearest neighbor ordinary RRT. After that, at each time step, db is decremented Dd while cb is incremented Dc. Now when a new configuration is sampled, its k-nearest neighbors are calculated and ordered in a list based on the selection cost. Then pick up the best extension that is satisfied

C(s, q) + C(q, qnew + h(qnew, qg)) < Ctarget