L20 -- Markov Chain
[Jeff Phillips - Utah - Data Mining]

Graph G = (E,V)
  V = vertices {a,b,c,d,e,f,g,h}
  E = edges   {(a,b), (a,c), (a,d), (b,d), (c,d), (c,e), (e,f), (e,g), (f,g), (f,h)}
      unordered pairs

Draw graph:

	a	b	c	d	e	f	g	h
a	0	1	1	1	0	0	0	0
b	1	0	0	1	0	0	0	0
c	1	0	0	1	1	0	0	0
d	1	1	1	0	0	0	0	0
e	0	0	1	0	0	1	1	0
f	0	0	0	0	1	0	1	1
g	0	0	0	0	1	1	0	0
h	0	0	0	0	0	1	0	0

adjacency matrix

Each v in V is a state.
If at b, represent state as
q = [0 1 0 0 0 0 0 0]^T

Can "think of" fractional state
q = [1/2 0 0 1/2 0 0 0 0]^T
1/2 at a and 1/2 at d

probability of being in each state:
each q[i] >= 0 and sum_i q[i] = 1

Transition matrix P = normalized adjacency matrix

	a	b	c	d	e	f	g	h
a	0	1/2	1/3	1/3	0	0	0	0
b	1/3	0	0	1/3	0	0	0	0
c	1/3	0	0	1/3	1/3	0	0	0
d	1/3	1/2	1/3	0	0	0	0	0
e	0	0	1/3	0	0	1/3	1/2	0
f	0	0	0	0	1/3	0	1/2	1
g	0	0	0	0	1/3	1/3	0	0
h	0	0	0	0	0	1/3	0	0

```
then given a state q, we can "transition" to the next state by
 q_1 = P*q
This one "step" of a "Markov Chain".

"Markov" means that each state only depends on previous state.

next step
q_2 = P*q_1   or
    = P*P=q   or
    = P^2*q

q_n = P^n*q
  where P^n = P*P*P* ... n times ... *P

Can think of as a randomized random walk.
  + start state q=q_0.
  + each step, takes one path at random
  + q_n is probability distribution of state after i steps
  + thus each column of P^n positive, sums to 1 for all n
---------------------------------------

Markov Chain is **ergodic** if
  exists some t such that for all n>=t then
  each entry in P^n is positive.

-->  for any q, then
 q_n = P^n q
is positive in all elements
--> after t steps, always have *some* probability of being anywhere.

------
When is a chain not ergodic?
 + cyclic
   P = [0 1]
       [1 0]
   always alternates states in even/odd states
   --> can be larger and more irregular, uncommon in practice

 + has absorbing + transient states
   P based on *directed* graph
   P = [0   1/2 1/2 0]
       [1/2 0   1/2 1]
       [1/2 1/2 0   0]
       [0   0   0   0]
   state d always goes to b, but can never return to d.
   also...
```

```
   P = [0   1/2 1/2 0  ]
       [1/2 0   1/2 1/2]
       [1/2 1/2 0   0  ]
       [0   0   0   1/2]
   may stay at d (w.p. 1/2) but state "seeps" from d to b (and then a,c)

   (a,b,c) = absorbing,  d = transient

 + not connected
   P = [1/2 1/2 0   0  ]
       [1/2 1/2 0   0  ]
       [0   0   2/3 1/2]
       [0   0   1/3 1/2]
   (a,b) cannot reach (c,d) and vice-versa
----------------------------------------------------

Consider an ergodic Markov Chain (P,q)

**AMAZING** property

let P^* = P^n as n -> infty
 then q_* = P^* q
   is **NOT** dependent on q

--> That is, for all starting states q, the final state is q_*

--> as we do a random walk, we will eventually be in the same expected state.

Note that q_* = P^* q  = P^{*+1} q
       so q_* = P q_*
--> If state distribution is initially q_*, then already in final
distribution.
    q_* second eigenvector of P
        second eigenvalue determines rate of convergence
          -->  smaller <-> faster convergence



------------------------------------------------------
------------------------------------------------------

Metropolis Algorithm  (MCMC)
  Metropolis, Rosenbluth, Rosenbluth, Teller, Teller 1953
    (Boltzman dist, Manhattan project)
  Hastings 1970
    (more general)

each state v in V has weight associated with it
```

w(v)   sum_{v in V} w(v) = W

Want to land in state v w.p. w(v)/W

--> V might be very large, and W unknown.
--> V can be "continuous"
"probe-only" can only measure w(v) at any one state

Strategy: design special Markov Chain so q_*[v] = w(v)/W

Start v_0 in V   (q = [0 0 0 ... 1 ... 0 0]^T)

choose neighbor u   (proportional to K(v,u))
if (w(u) >= w(v_i))   --> v_{i+1} = u
else w.p. w(u)/w(v)   --> v_{i+1} = u
else   --> v_{i+1} = v_i

if ergodic:
there exists some t s.t. for i >= t
Pr[v_i = v] = w(v)/W

NOTE: not in limit, but for some finite t (even for continuous) V
through AMAZING "coupling from past"
But t is hard to find.

Often goal is to create many samples:
formal: run for t+ steps, take sample, ...
run for another t+ steps, take sample, ... repeat

in practice: run for 1000 steps (burn in),
take next 5000 steps as random samples

has "auto-correlation" but eventually more time efficient than tN steps for N samples
and t unknown.

******
"inherently sequential" makes very hard to parallelize

******
Applies even if V is continuous