UNIT #3 -- Hidden Markov Model and Dynamic Programming

Natural language processing Lab
© 2011 Jason S. Chang, Department of Computer Science, NTHU

(adapted from Wikipedia and Wikibook)

According to Wikipedia (http://en.wikipedia.org/wiki/Hidden_Markov_model)

1. HMM provides a statistical account of a sequence of observations (tokens) generated by a Markov process with hidden states
2. Each (hidden) state is dependent on a fixed number of previous states
3. Each (visible) observation is dependent on the correponding state
4. Therefore, the sequence of observations generated by an HMM gives some information about the sequence of states
5. Hidden Markov models apply in many natural language processing problems, including speech recognition, ocr, part-of-speech tagging

- Hidden states Y
- Observed outputs $x_0, x_1, \ldots, x_T$
- For state i in Y
  - Initial probabilities π_i of being in state i
- For state i, j in Y
  - Transition probabilities $a_{i,j}$ of moving from state i to state j.
- For state i in Y and some observation *k*
  - Emission probabilities $b_{i,k}$ of emitting the observation *k* while in state i

Examples and connections to previous labs

1. For speech recognition or Chinese input method
    - Obs. = voice signal/key strokes
    - States = words
    - State transition = bigram language model (visible Markov model)
    - Emission = acoustic model/phone-character mapping

    Hidden states = Chinese character string, e.g., 請幫我改一下自傳
    Obs = The string of phonic symbols, e.g., ㄑㄅㄨㄍㄧㄒㄗㄓ
    Initial prob. = Unigram prob of Chinese character, e.g., P(請)
    Transition prob. = Character bigram prob of, e.g., P(幫|請)
    Emission prob. = P(ㄑ|請)

2. For part of speech tagging
    - Obs. = words
    - States = parts of speech (usually hidden, need annotation or learning)
    - State transition = pos ngram
    - emission prob. = pos to word mapping (the reverse of dictionary)

(1)The Evaluation Problem

- Given an HMM and a (short) sequence of observations, what is the probability of the observations being generated by the model?

(2)The Decoding Problem

- Given a model and a (short) sequence of observations, what is the most likely state sequence producing the observations?

(3)The Learning Problem

- Given a tentative model and a (long) sequence of observations, how should we adjust the model parameters, maximizing the prob. of the observations

Divide and Conquer to Solve the Decoding Problem

- The original problem
  - What is the most likely states $y_0, y_1, \ldots, y_T$ producing the observations $x_0, x_1, \ldots, x_T$?
- Divide and conquer (define and solve the subproblems)
  - What is the most likely states $y_0, y_1, \ldots, y_t$ for t in range(T)?
- Subproblems need the have compatible states
  - What is the most likely states $y_0, y_1, \ldots, y_t$ ending in state k?
  - Define probability $V_{t,k}$ of $y_0, y_1, \ldots, y_t\,(y_t = k)$

Decoding -- the Viterbi Algorithm (http://en.wikipedia.org/wiki/Viterbi_algorithm)

Suppose we are given

- Hidden Markov Model (HMM) with states Y
- Initial prob. π_i , transition probabilities $a_{i,j}$, and emission prob. $b_{i,k}$
- Observed outputs $x_0, x_1, \ldots, x_T$

The Viterbi path, or the most likely state sequence $y_0, y_1, \ldots, y_T$ is:

$$V_{0,k} = \mathrm{P}(x_0 \mid k) \cdot \pi_i$$

$$V_{t,k} = \mathrm{P}(x_t \mid k) \cdot \max_{y \text{ in } Y} (a_{y,k} \cdot V_{t-1,y})$$

$$\mathrm{Ptr}(t,k) = \mathrm{P}(x_t \mid k) \cdot \operatorname{argmax}_{y \text{ in } Y} (a_{y,k} \cdot V_{t-1,y})$$

The size of V is $T \times |Y|$ and each $V_{t,k}$ take $|Y|$ steps to compute

The complexity of Viterbi algorithm is $O(T \times |Y|^2)$

The Viterbi path

The Viterbi path retrieved by computing and saving back pointers

$$\mathrm{Ptr}\,(t, k) = \mathrm{P}(x_t \mid k)\ \mathrm{argmax}_{\text{y in Y}}\ a_{y,\,k}\ V_{t\text{-}1,\,y}$$

Then the path in reverse is the following

$$y_T = \mathrm{argmax}_{y\ in\ Y}\ V_{T,\,y}$$

$$y_{T\text{-}1} = \mathrm{Ptr}\,(T, y_T)$$

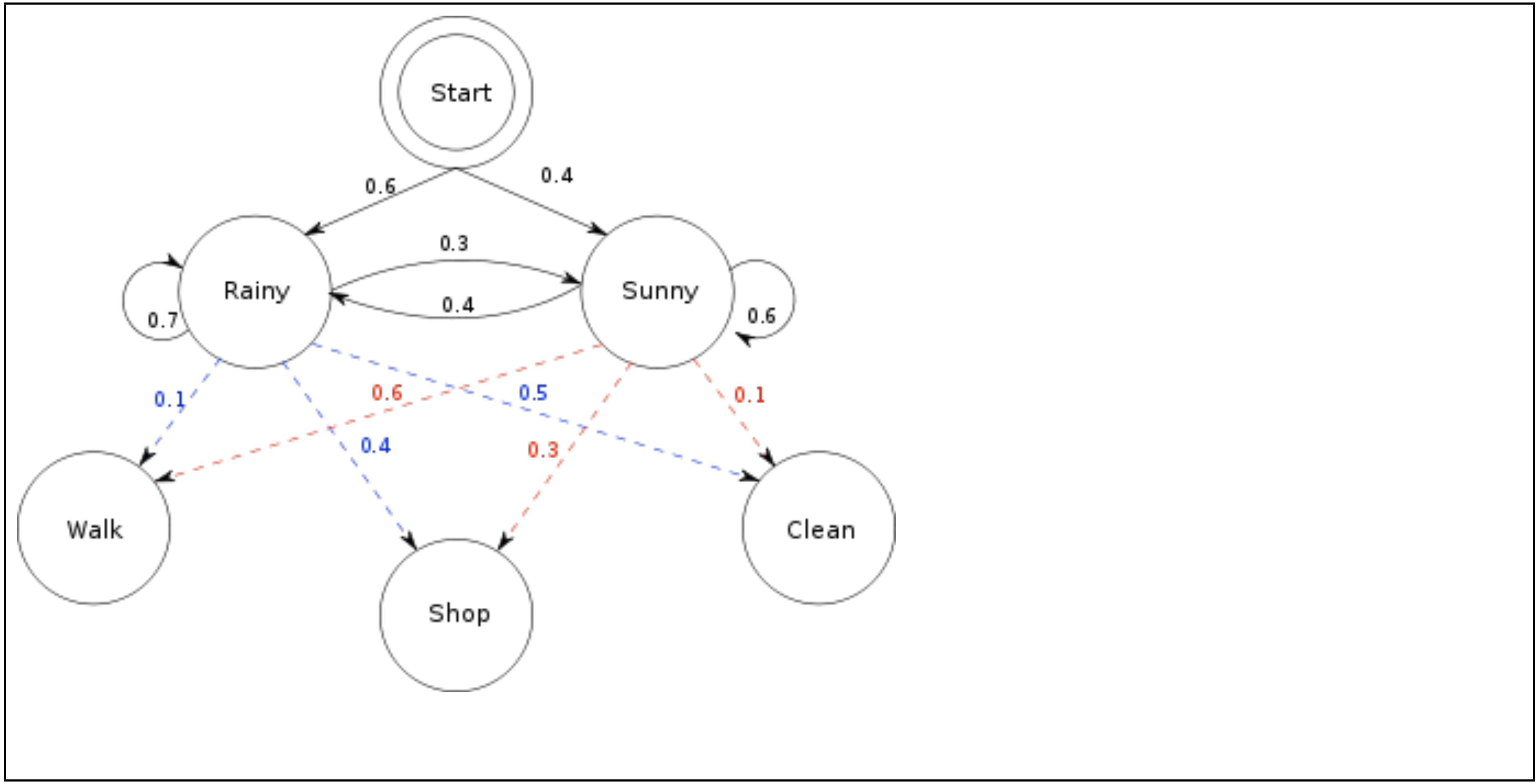
$$\ldots$$

$$y_1 = \mathrm{Ptr}\,(2, y_2)$$

$$y_0 = \mathrm{Ptr}\,(1, y_1)$$

Example of the Viterbi Algorithm in Python

```
states = ('Rainy', 'Sunny')
observations = ('walk', 'shop', 'clean')

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
   'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
   }

emission_probability = {
   'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
   }
```

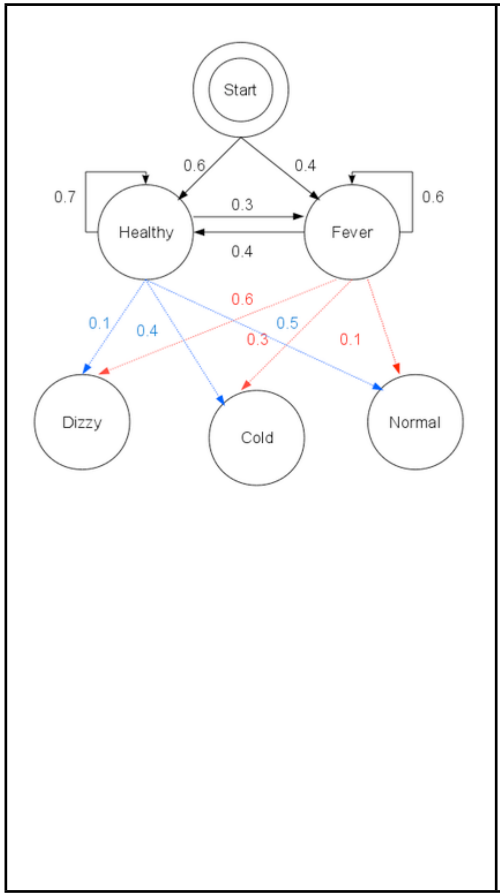

Start
0.6
0.4
Rainy
0.3
0.4
Sunny
0.7
0.6
0.1
0.6
0.5
0.1
0.4
0.3
Walk
Shop
Clean

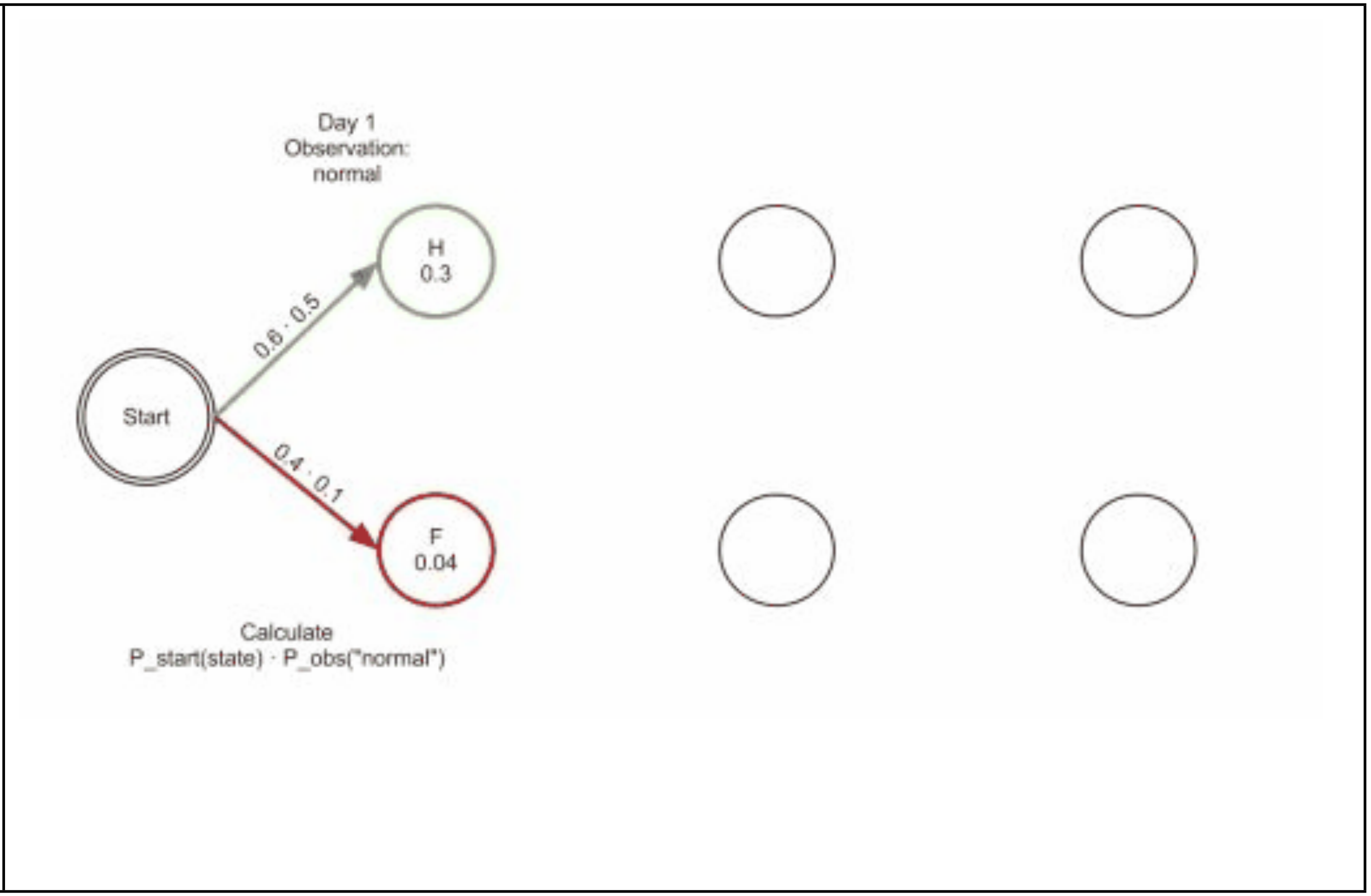
The Viterbi Algorithm in Python

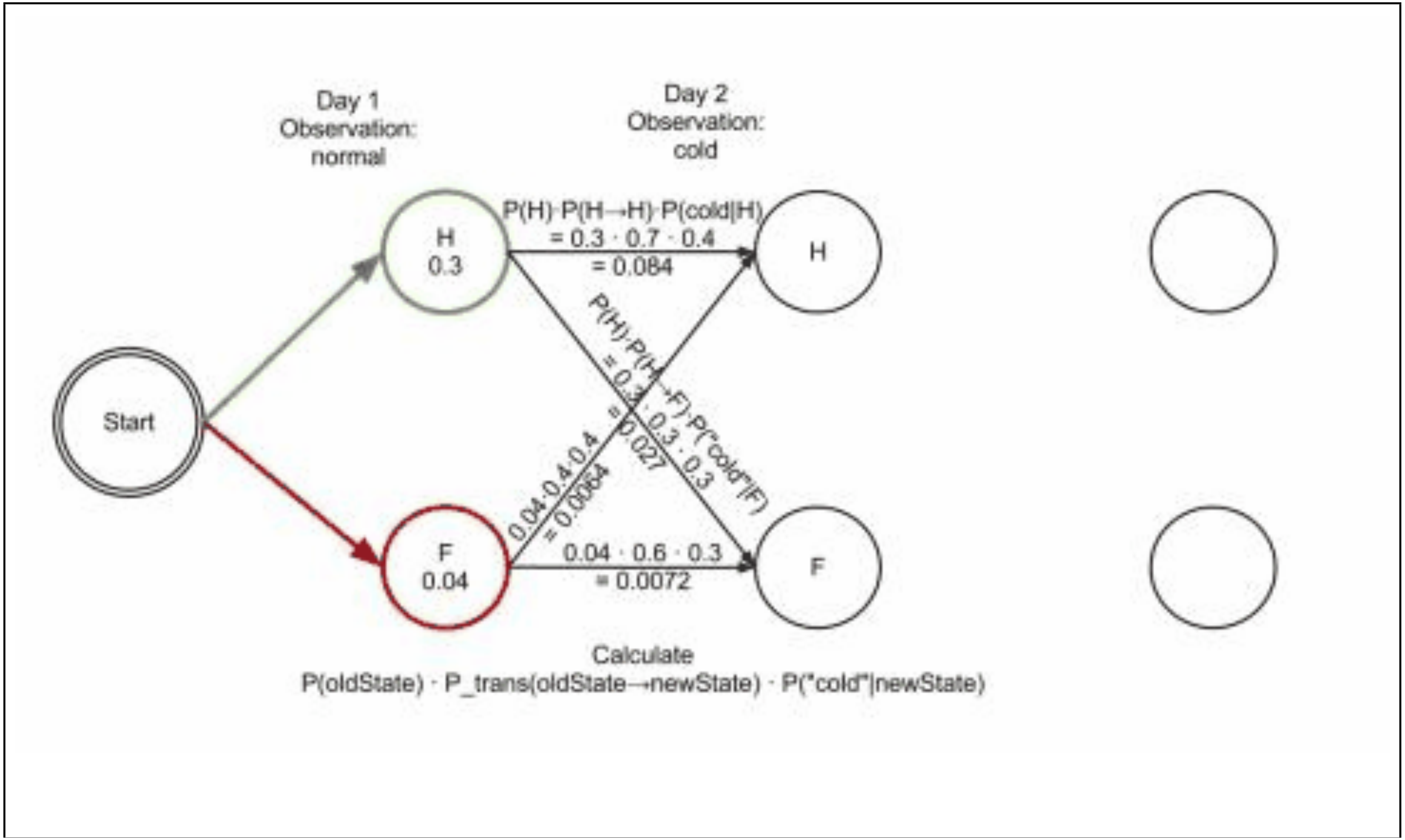
```
def viterbi(obs, states, start_p, trans_p, emit_p):
  V, path = [{}], path = {}

  # Initialize base cases (t == 0)
  for y in states:
    V[0][y], path[y] = start_p[y] * emit_p[y][obs[0]], [y]

  for t in range(1,len(obs)):
    V.append({})
    newpath = {}

    for y in states:
      (prob,state) = max([(V[t-1][y0]*trans_p[y0][y]*emit_p[y][obs[t]],y0) \
                          for y0 in states])
      V[t][y] = prob
      newpath[y] = path[state] + [y]

  path = newpath
  (prob, state) = max([(V[len(obs) - 1][y], y) for y in states])
  return (prob, path[state])
```

Animation of the Viterbi Algorithm

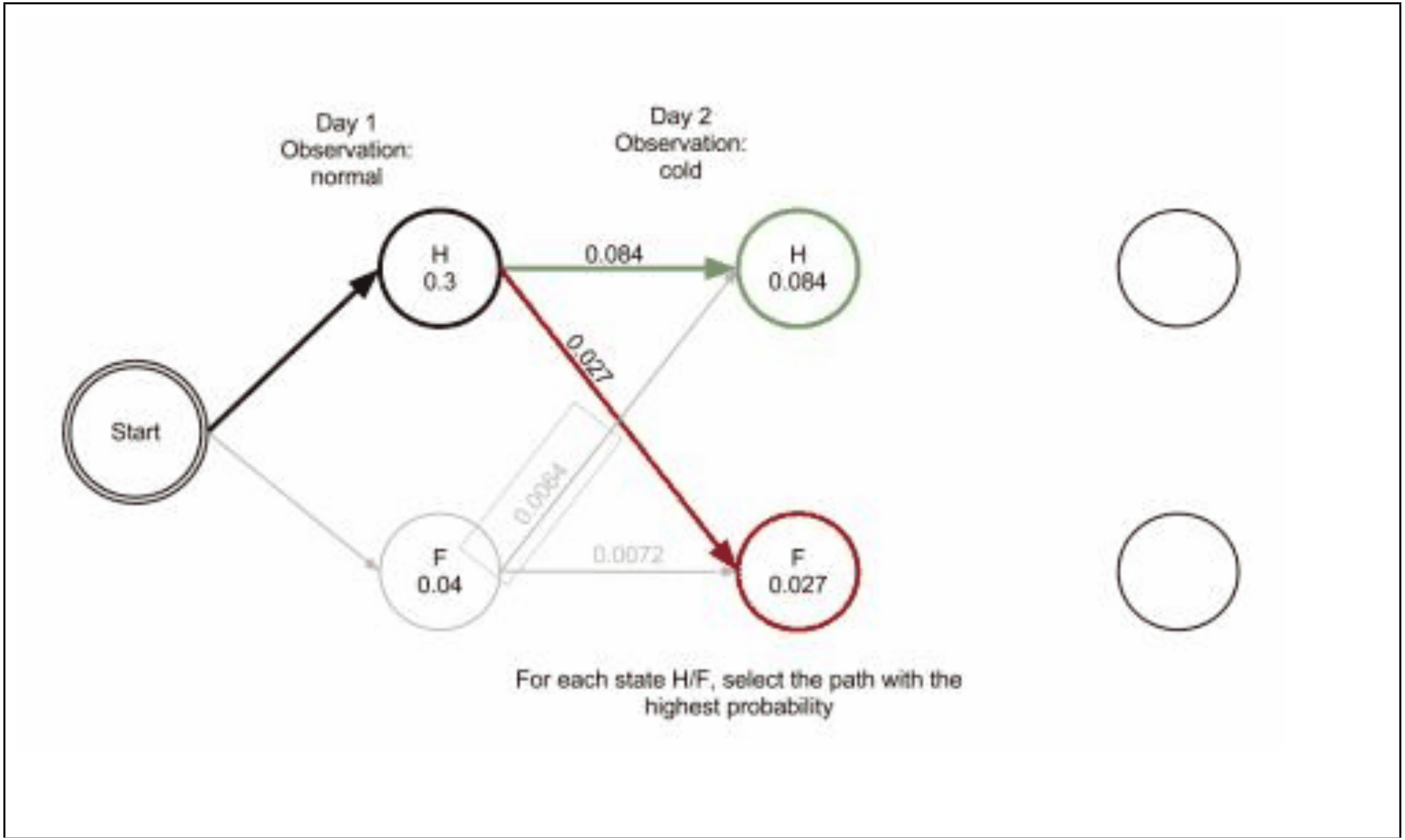
http://en.wikipedia.org/wiki/File:Viterbi_animated_demo.gif

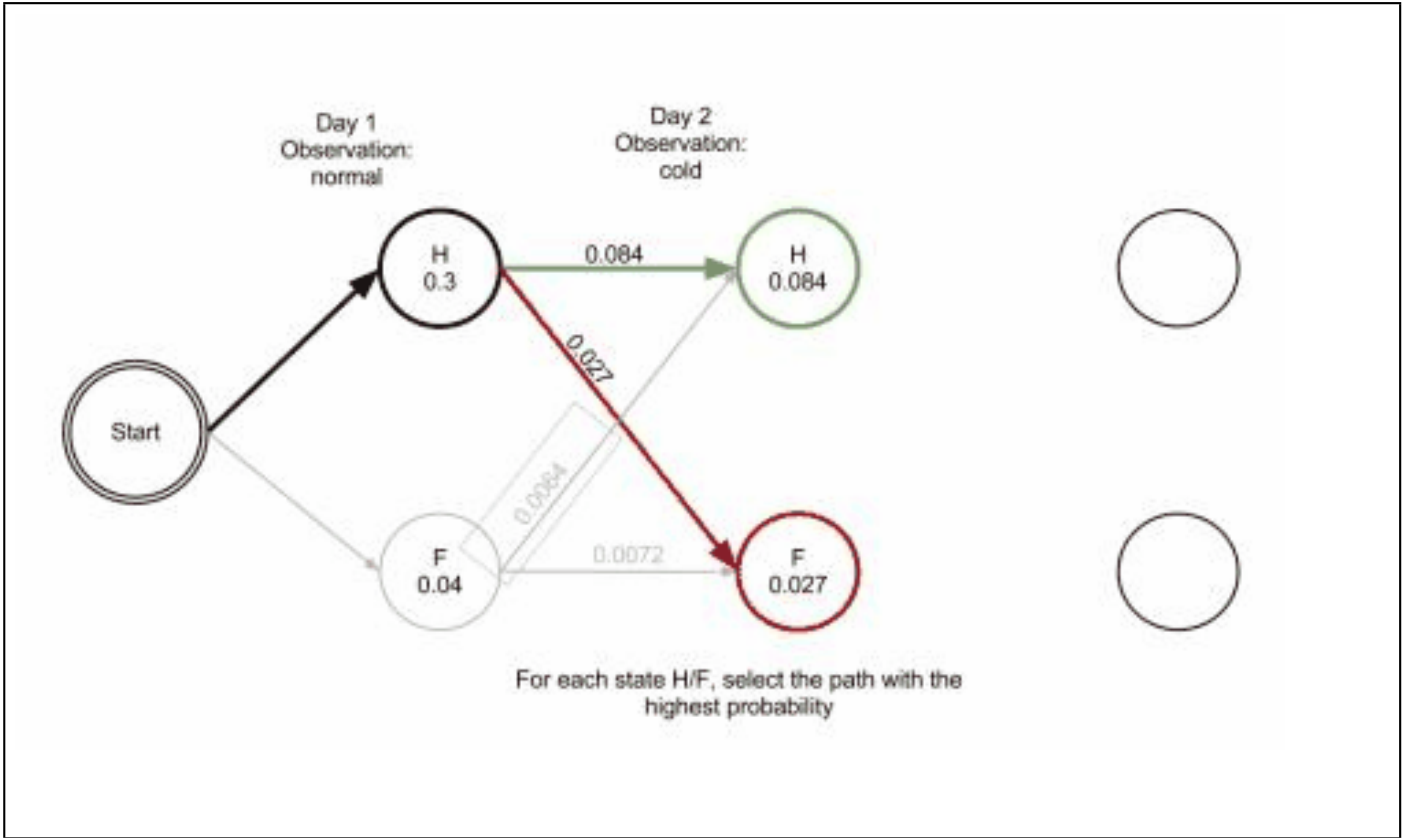




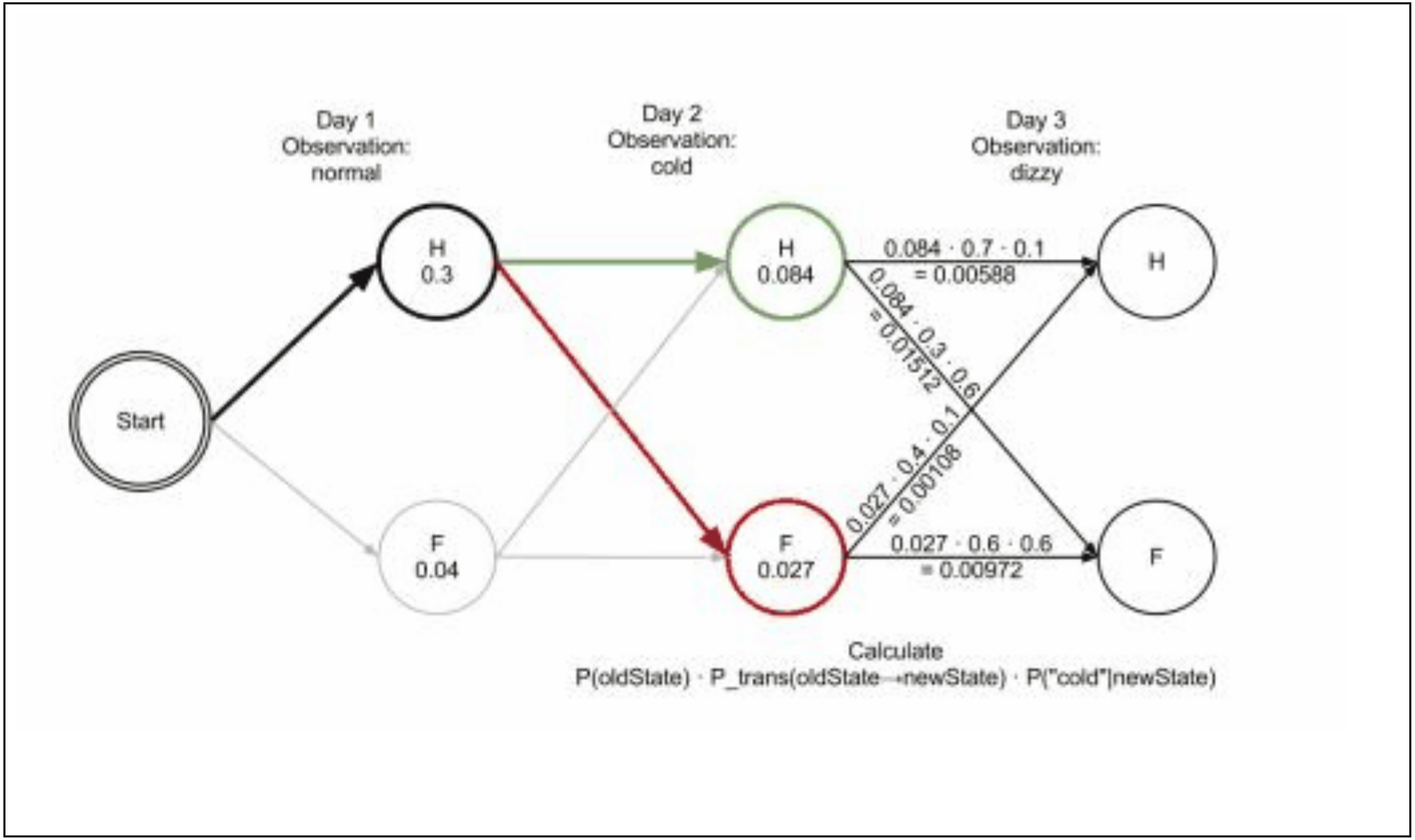


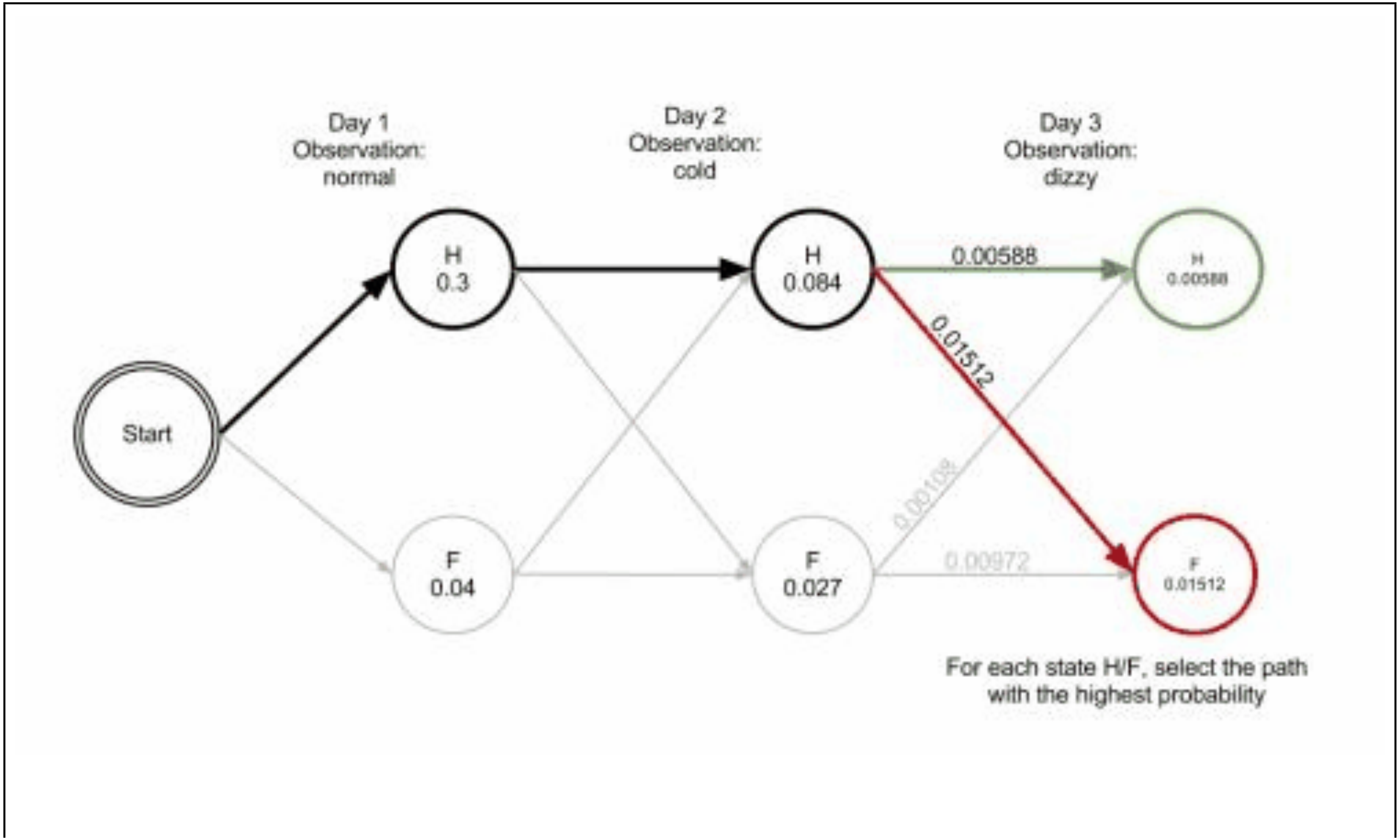
Day 1
Observation:
normal
Day 2
Observation:
cold
Start
H
0.3
F
0.04
H
F
P(H)·P(H→H)·P(cold|H)
= 0.3 · 0.7 · 0.4
= 0.084
P(H)·P(H→F)·P("cold"|F)
= 0.3 · 0.3 · 0.3
= 0.027
0.04·0.4·0.4
= 0.0064
0.04 · 0.6 · 0.3
= 0.0072
Calculate
P(oldState) · P_trans(oldState→newState) · P("cold"|newState)


Day 1
Observation:
normal
Day 2
Observation:
cold
H
0.3
0.084
H
0.084
0.027
Start
0.0084
F
0.04
0.0072
F
0.027
For each state H/F, select the path with the highest probability



Day 1
Observation:
normal
Day 2
Observation:
cold
Start
H
0.3
F
0.04
H
0.084
F
0.027
0.084
0.027
0.0084
0.0072
For each state H/F, select the path with the highest probability


Day 1
Observation:
normal
Day 2
Observation:
cold
Day 3
Observation:
dizzy
Start
H
0.3
F
0.04
H
0.084
F
0.027
H
F
0.084 · 0.7 · 0.1
= 0.00588
0.084 · 0.3 · 0.6
= 0.01512
0.027 · 0.4 · 0.1
= 0.00108
0.027 · 0.6 · 0.6
= 0.00972
Calculate
P(oldState) · P_trans(oldState→newState) · P("cold"|newState)


Day 1
Observation:
normal
Day 2
Observation:
cold
Day 3
Observation:
dizzy
Start
H
0.3
F
0.04
H
0.084
F
0.027
0.00588
0.01512
0.00108
0.00972
H
0.00588
F
0.01512
For each state H/F, select the path
with the highest probability

Visualization

```
# Helps visualize the steps of Viterbi.
def print_dptable(V):
    print "    ",
    for i in range(len(V)): print "%7s" % ("%d" % i),
    print

    for y in V[0].keys():
        print "%.5s: " % y,
        for t in range(len(V)):
            print "%.7s" % ("%f" % V[t][y]),
        print
```

```
           0       1       2
Healt:  0.30000 0.08400 0.00588
Fever:  0.04000 0.02700 0.01512
(0.01512, ['Healthy', 'Healthy', 'Fever'])
```

The Viterbi Algorithm in Python (with visualization)

```
def viterbi(obs, states, start_p, trans_p, emit_p):
  V, path = [{}], path = {}

  # Initialize base cases (t == 0)
  for y in states:
    V[0][y], path[y] = start_p[y] * emit_p[y][obs[0]], [y]

  for t in range(1,len(obs)):
    V.append({})
    newpath = {}

    for y in states:
      (prob,state) = max([(V[t-1][y0]*trans_p[y0][y]*emit_p[y][obs[t]],y0) \
                          for y0 in states])
      V[t][y] = prob
      newpath[y] = path[state] + [y]

  path = newpath

  print_dptable(V)

  (prob, state) = max([(V[len(obs) - 1][y], y) for y in states])
  return (prob, path[state])
```

Example Run

```
states, obs = ('Rainy', 'Sunny'), ('walk', 'shop', 'clean')

start_prob = {'Rainy': 0.6, 'Sunny': 0.4}

transition_prob = {'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},}

emission_prob = {'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},}

print viterbi(obs,states,start_prob,transition_prob,emission_prob)

>>>
             0       1       2
Rainy:   0.06000 0.03840 0.01344
Sunny:   0.24000 0.04320 0.00259
(0.01344, ['Rainy', 'Rainy'])
>>>
```

Lab #3

Chinese Input Method:

- Input: ㄑㄅㄨㄍㄧㄒㄗㄓ OR q b w g y x z zh (漢語拼音）
- Output: 請幫我改一下自傳 (or at least one of the suggestions)

```
states = Chinese character string, e.g., 請幫我改一下自傳
obs = The string of phonic symbols, e.g., ㄑㄅㄨㄍㄧㄒㄗㄓ
start_prob = unigram prob of Chinese character, e.g., P(請)
transition_prob = bigram prob of Chinese characters, e.g., P(幫|請)
emission_prob = P(ㄑ| 請)
```

Data

1. Chinese corpus (Giga word Chinese or Sinica Balanced Corpus)
2. Chinese phonic dictionary