Parts of Speech tagging-
Hidden Markov Models (HMM)

Sequence Labeling

- **Sequence Models:**
  - Sequence models are those where there is some sort of dependence through time between the inputs.
  - Its job is to assign a label to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels.
  - The classical example of a sequence model is the **Hidden Markov Model** for **part-of-speech tagging.**
- **Problem:**
  - Given a sequence of words, infer the most probable sequence of labels(tags) for these words

Examples :

- POS
- NER

HMM

- HMM is probabilistic function of a Markov process.
- HMM can be thought of as underlying events probabilistically generating surface events.

  For example tagging, assigning pos to words in text

POS tagging with HMM

Notations:

$W = w_1, w_2, w_3,..,w_n$ -sequence of words(Input)

$T = t_1, t_2, t_3, .., t_n$ -sequence of tags(labels)

- It is a process of finding the sequence of tags which is most likely to have generated a given word sequence.

Markov Processes

- A Markov process is a random process in which the future is independent of the past, given the present
- A Markov process satisfies the **Markov property**
- The markov property is characterized as "**memoryless**“ or **“forgetfulness”**

Markov Processes

Based on Two Properties

1. **Limited Horizon**: Given previous t states, a state i is independent of preceding 0 to t-(k+1) states

   $P(X_t = i / X_{t-1}, X_{t-2}, \ldots, X_0) = P(X_t = i / X_{t-1}, X_{t-2}, \ldots, X_{t-k})$

   - Order k Markov process

2. **Time invariance (shown for k=1):** dependence of t-state on t-1 state is same everywhere.

   $P(X_t=i/X_{t-1}=j) = P(X_1=i/X_0=j) \ldots. = P(X_n=i/X_{n-1}=j)$

Formal definition of HMM

An HMM is specified by:

1. The set S= s_1, s_2, s_3, …., s_N of hidden state
2. The start state s_0
3. The matrix A of transition probabilities:
   $a_{ij} = p(s_j/s_i)$
4. The set O of possible visible outcomes
5. The matrix B of output probabilities:
   $b_{ik}= p(o_k/s_i)$

HMM

- In the process, “**tags**” are the “**hidden states**” which produced the “**observable output**” i.e., “**words**”.
- Find out the most probable sequence of tags for the word sequence:

Best tag sequence: $T^* = \text{argmax}_T \{P(T/W)\}$

$T^* = \text{argmax}_T (P(T)*P(W/T))$

$$T^* = argmax_T \prod_i P(w_i|w_1 \ldots w_{i-1}, t_1 \ldots t_i) P(t_i|t_1 \ldots t_{i-1})$$

HMM

Best tag sequence: $T^* = \text{argmax}_T \{P(T/W)\}$

$$T^* = \text{argmax}_T (P(T)*P(W/T))$$

Simplify, using 2 assumptions:

1- **Bigram assumption** : The probability of a tag depends on the previous one (bigram model- $P(t_i/t_{i-1})$).

2- **Lexical Generation probability assumption** : The probability of a word depends only on its own POS tag, ($P(w_i/t_i)$).

$$\boxed{T^* = \text{argmax}_T(\prod_{i=1\ldots T} PROB(t_i|t_{i-1}) * PROB(W_i|t_i))}$$

Computing the probability values

1. **Tag Transition probabilities $p(t_i/t_{i-1})$**

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = 0.49$$

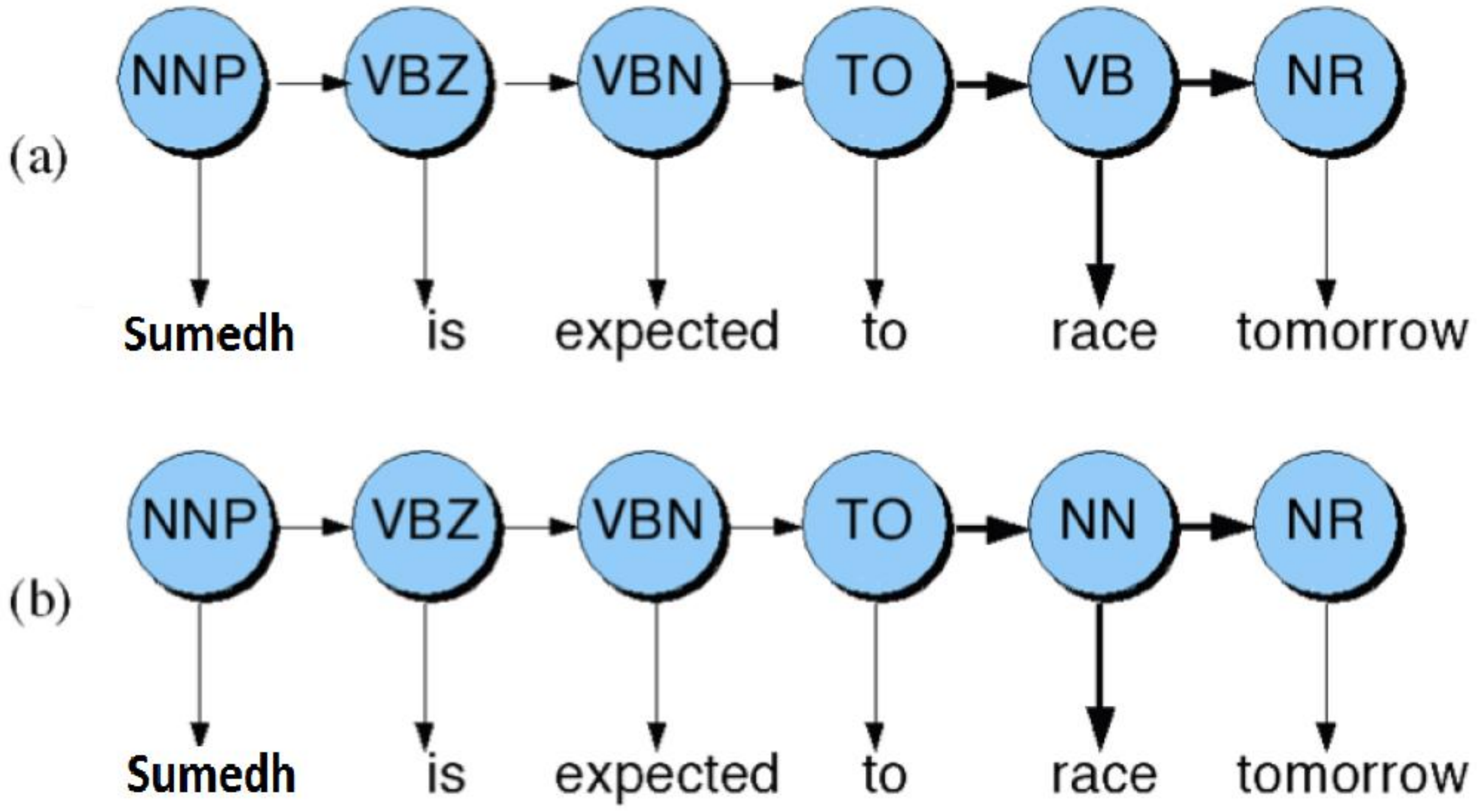
Computing the probability values

2. **Word Likelihood probabilities p(w_i/t_i)**

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = 0.47$$

Disambiguating race



Disambiguating race

Difference in probability due to

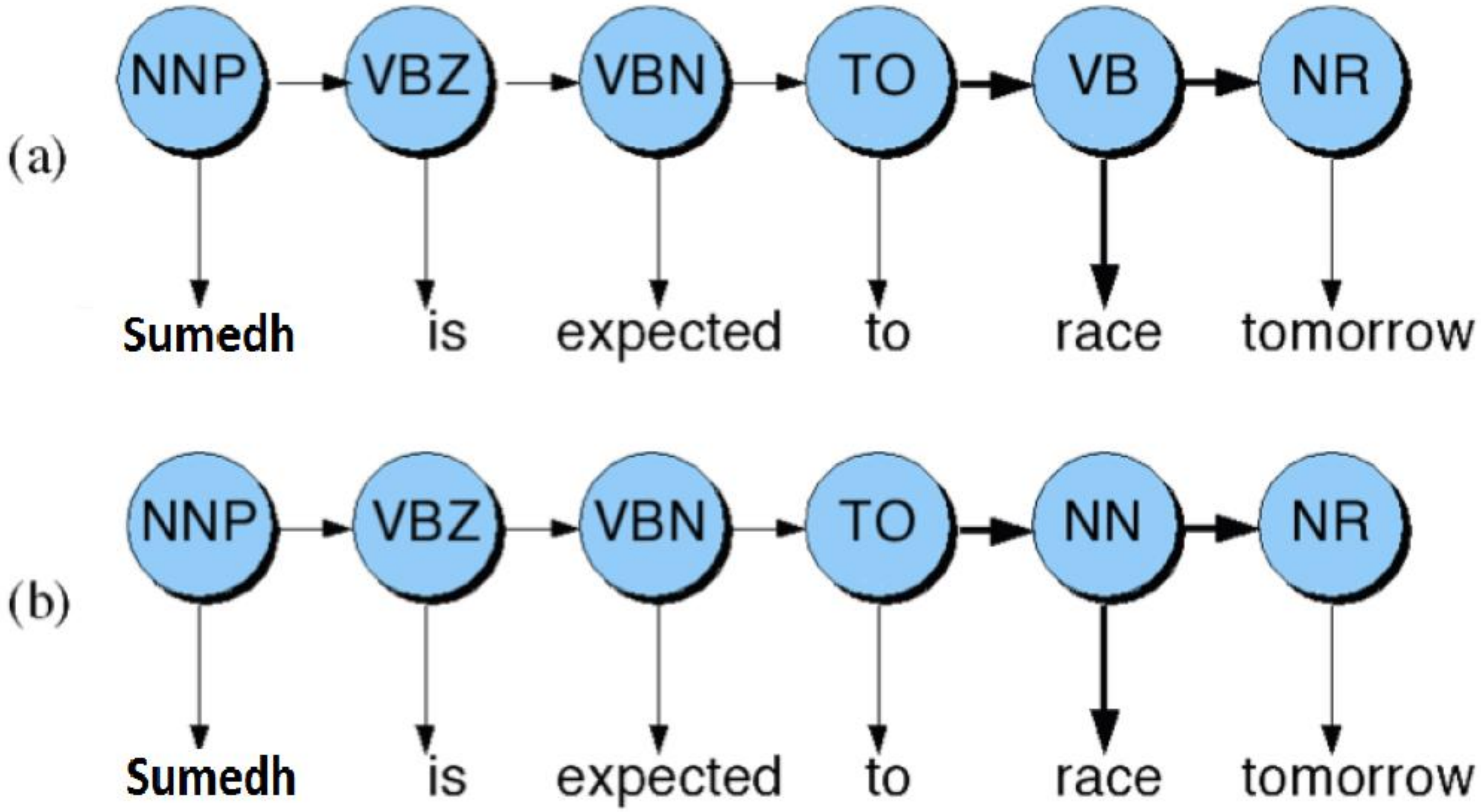
- P(VB/TO) vs. P(NN/TO)
- P(race/VB) vs. P(race/NN)
- P(NR/VB) vs. P(NR/NN)

Disambiguating race

After computing the probabilities:

- P(NN/TO)*P(NR/NN)*P(race/NN) = 0.0047*0.0012*0.00057 = 0.00000000032
- P(VB/TO)*P(NR/VB)*P(race/VB)=0.83*0.0027 *0.00012=0.00000027

What is this model?



This is a Hidden Markov Model

Hidden Markov Models

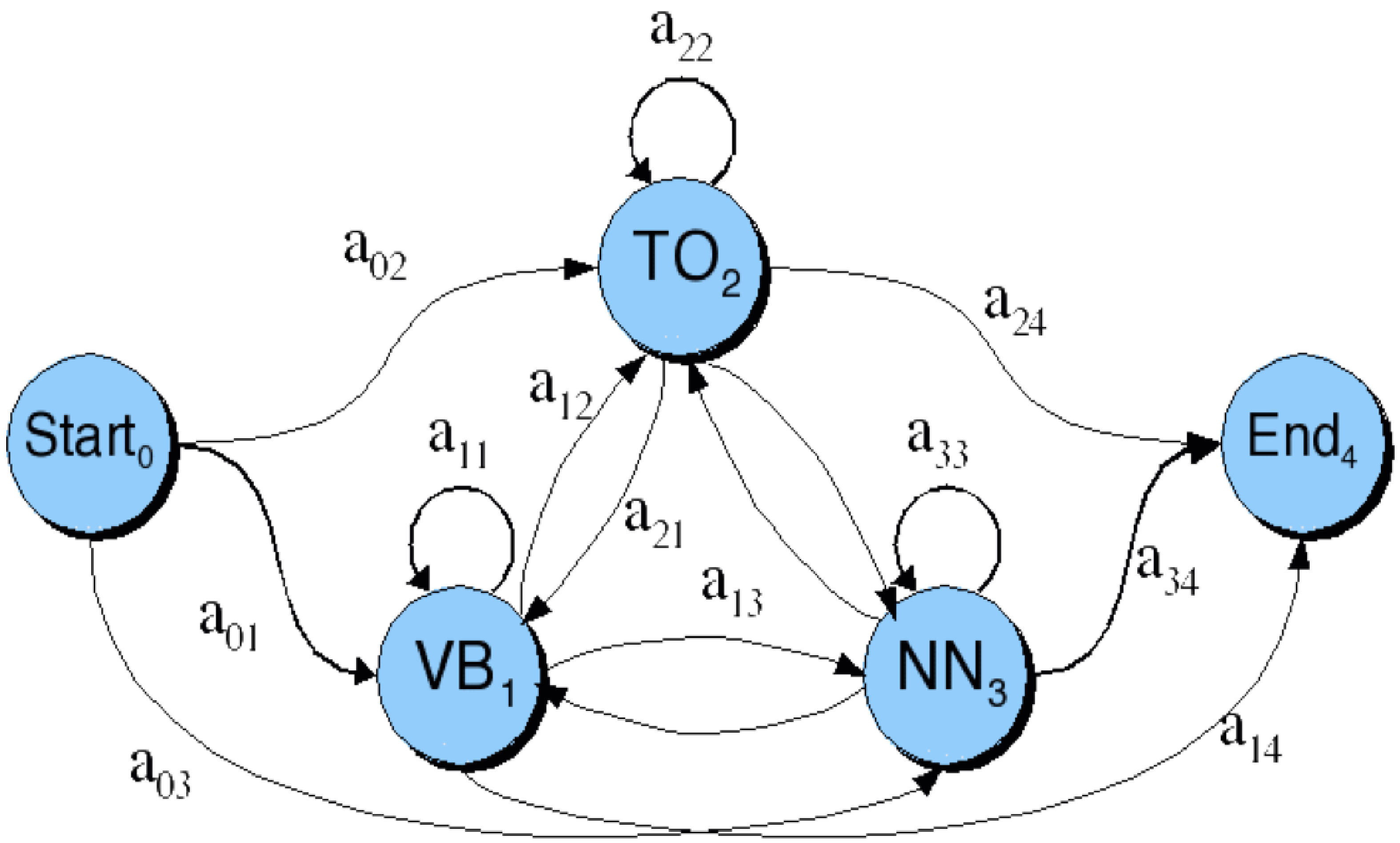
1. Tag Transition probabilities - $p(t_i/t_{i-1})$
2. Word Likelihood probabilities (emissions) - $p(w_i/t_i)$

Hidden Markov Models (HMMs)

Elements of an HMM model:

- A set of states (here: the tags)
- An output alphabet (here: words)
- Initial state (here: beginning of sentence)
- State transition probabilities (here $p(t_i/t_{i-1})$)
- Symbol emission probabilities (here $p(w_i/t_i)$)

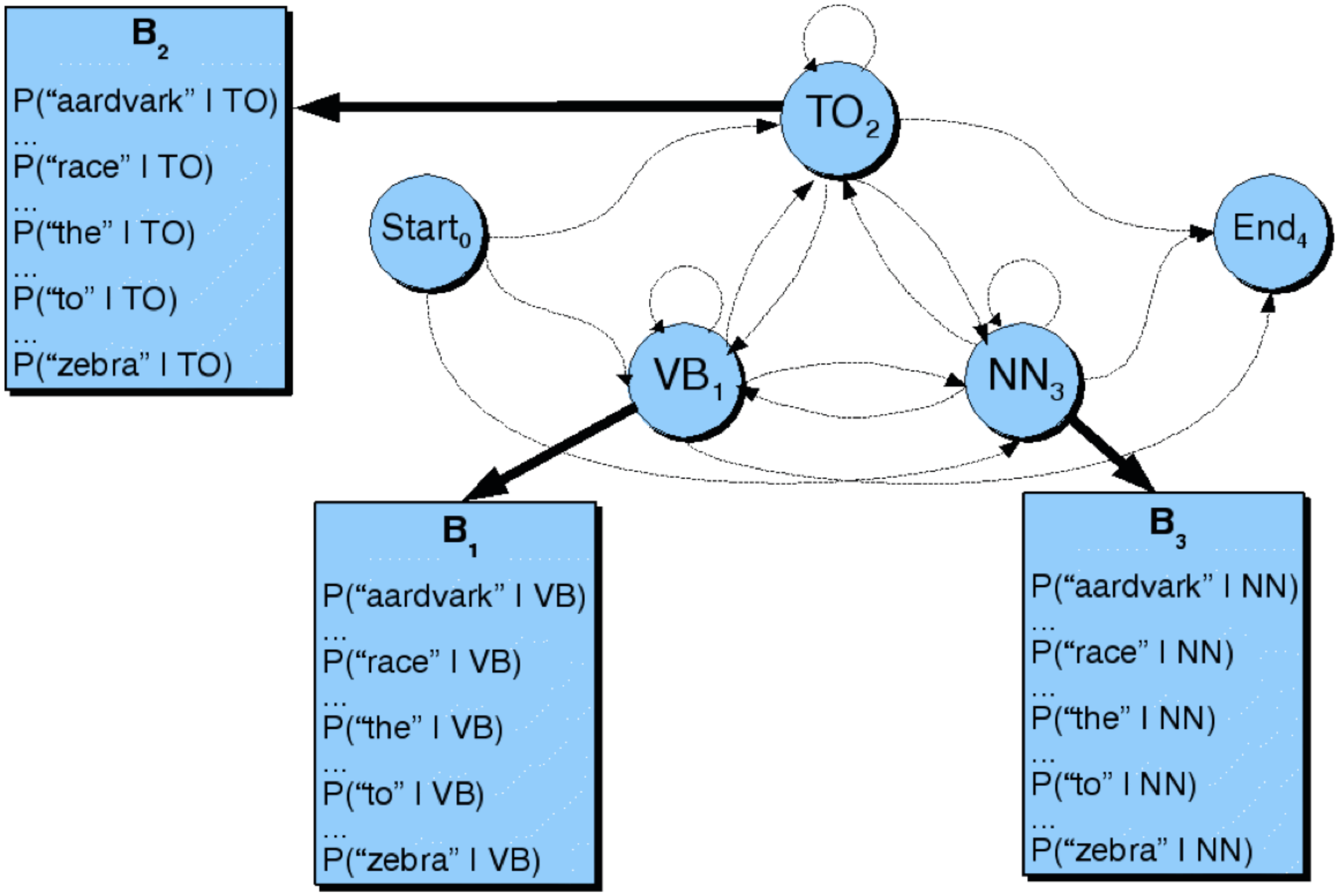
Graphical Representation



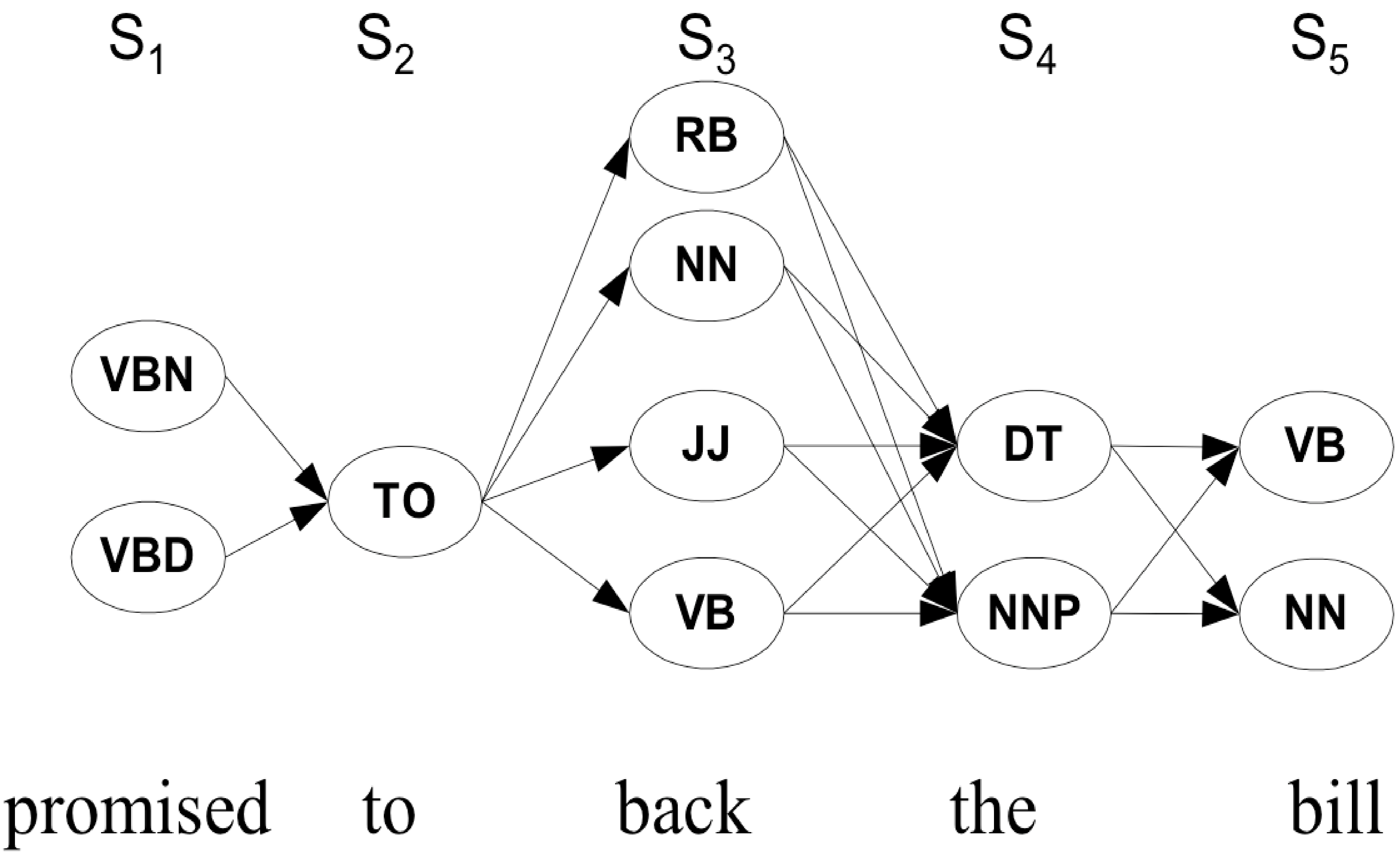
- Edges are labeled with the state transition probabilities: $P(t_i/t_{i-1})$

Graphical Representation

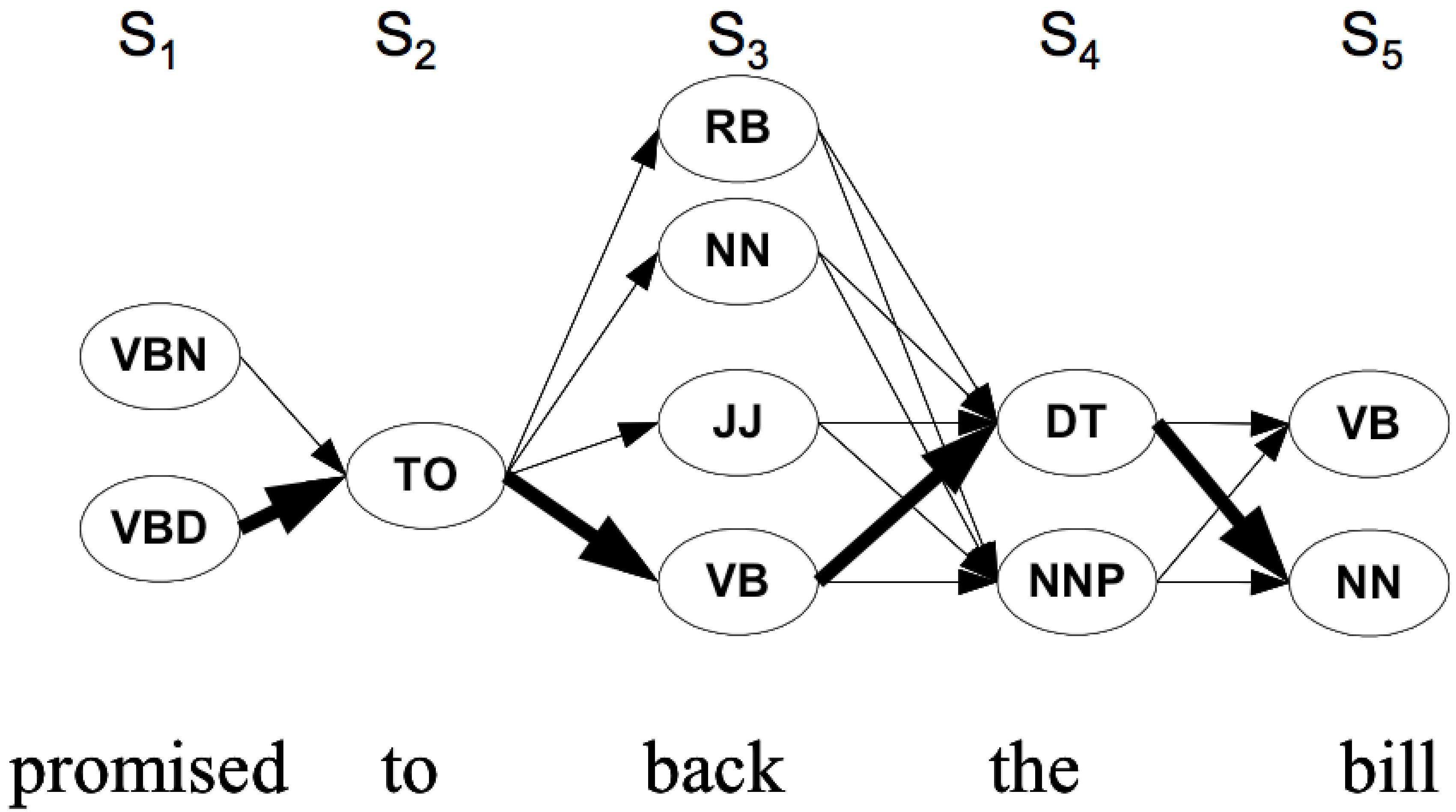
- At each state we emit a word: $P(w_i/t_i)$



Walking through the states: best path



Walking through the states: best path



Problem

- Consider tag set containing only four tags (ART, N, V, P) then with our independence assumption (the tag t_i depends only on the tag of the (i-1)th word) we can use a markov chain to capture this bigram probabilities as shown in the figure below. Assume that any bigram probability not in Figure 1 has a value of .0001.

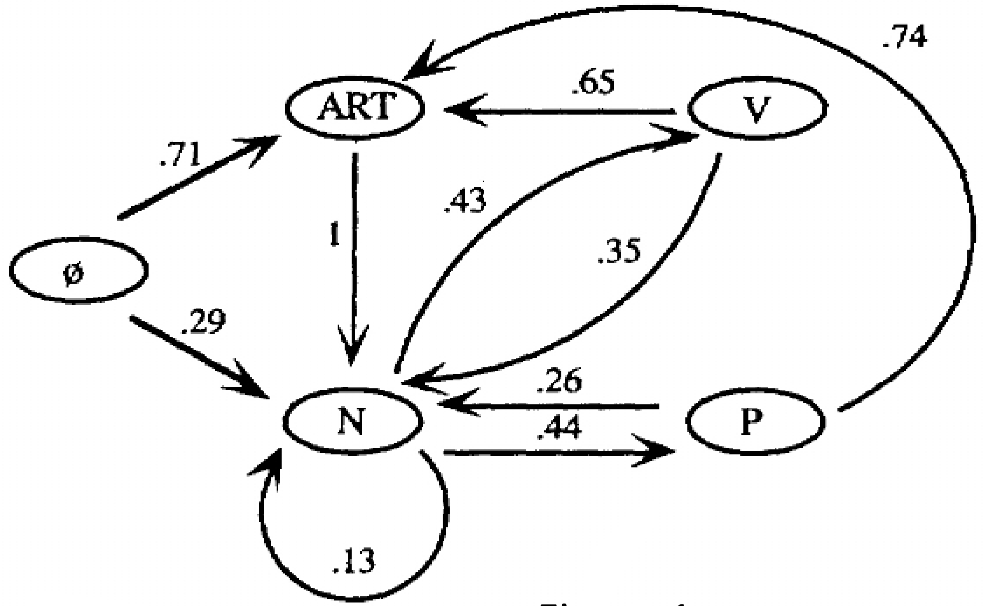


Figure 1

With the help of the above diagram the probability of a sequence can be calculated by multiplying the probabilities along the path.
e.g.

$$P(N, V, ART, N) = 0.29 * 0.43 * 0.65 * 1$$

Such a network representation is called a **Markov model.**

Lexical Probability

- We can further extend this network representation to include the word generation probabilities
- With each of the tags(states) we can associate a probability table that indicates, for each word (observable output) how likely that word is to be selected if that particular tag(state) is selected.
- These probabilities are nothing but $P(W_i|T_i)$.

Viterbi Algorithm

- Uses a dynamic programming approach where we sweep forward through the words one at a time to find the most likely sequence ending in each tag.
- For example, for the word sequence : “**Flies like a flower**”,
  - First, find the four best sequences for the first word each ending in different tag from start.
  - Second, find the four best sequences for the two words "*flies like*"*:* the best ending with "*like*" as a V, the best ending with "*like*" as an N, the best ending with "*like*" as a P, and the best ending with "*like*" as an ART.
  - Using this information we will then find the four best sequences for "*flies like a*", each one ending in a different tag.
  - This process is repeated until all the words in the sentence are considered.
- This algorithm is called the **Viterbi** algorithm. For a problem involving **T words** and **N lexical categories**, requires: **$k*T*N^2$** steps

Viterbi Algorithm

```
Initialization Step
For i = 1 to N do                    //N is no of lexical categories and T is no of words
SEQSCORE (i,1) = $P(W_1|C_i) * P(C_i|ø)$
BACKPTR(i,1) = 0
Iteration Step
For t = 2 to T do
For i = 1 to N
SEQSCORE (i, t) = $MAX_{j=1,N}$(SEQSCORE (j,t-1) * $P(C_i|C_j)$) * $P(w_t| C_i)$
BACKPTR(i, 1) = index of j that gave max above
 Sequence Identification Step
      C(T) = i that maximizes SEQSCORE (i,T)
For i = t-1 to 1 do
  C(i) = BACKPTR(C(i+1) , i+1) //        Back trace to find the sequence
```

- The array SEQSCORE (i, j) stores the probability for the optimal sequence up to word j ending in the category i.

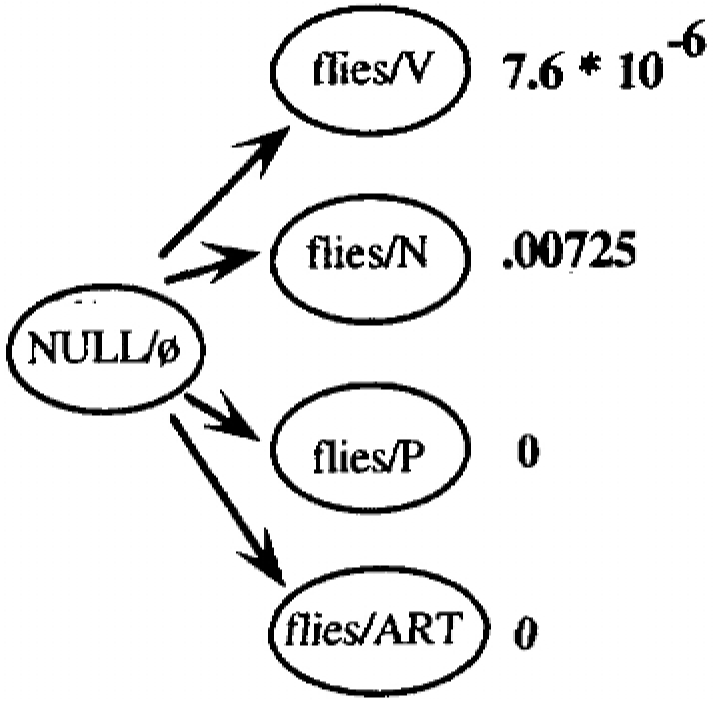
- Consider that after analyzing the corpus we have estimated the bigram probabilities as shown in Figure 1 and lexical-generation probabilities as shown in the table below:-

PROB (*the* | ART)	.54	PROB (*a* | ART)	.360
PROB (*flies* | N)	.025	PROB (*a* | N)	.001
PROB (*flies* | V)	.076	PROB (*flower* | N)	. 063
PROB (*like* | V)	0.1	PROB (*flower* | V)	.05
PROB (*like* | P)	0.068	PROB (*birds* | N)	.076
PROB (*like* | N)	0.012		

Now, using Viterbi Algorithm, the first row is set in the initialization phase using the formula:-

$$\text{SEQSCORE}(i,1) = P(\text{Flies}|C_i) * P(C_i|\text{ø})$$

- The result of the first step of the algorithm, the probability of "flies" in each category has been computed, and is shown below:



SEQSCORE (i,1) = P (Flies|C$_i$) * P (C$_i$|ø)

P (Flies|V) * P (V|ø) = 0.076 * 0.0001 = **$7.6 * 10^{-6}$**

P (Flies|N) * P (N|ø) = 0.025 * 0.29 = **0.00725**

P (Flies|P) * P (P|ø) = 0 * 0.0001 = **0**

P (Flies|ART) * P (ART|ø) = 0 * 0.71 = **0**

The second phase of the algorithm extends the sequences one word at a time, keeping track of the best sequence found so far to each category.

Probability of the state **like/V** is computed as:

PROB (like/V) = MAX (PROB (flies/N) * PROB (V | N),

PROB(flies/V) * PROB (V | V)) * PROB (like/V)

= MAX (.00725 * .43, 7.6 * 10^{-6} * .0001) * 0.1

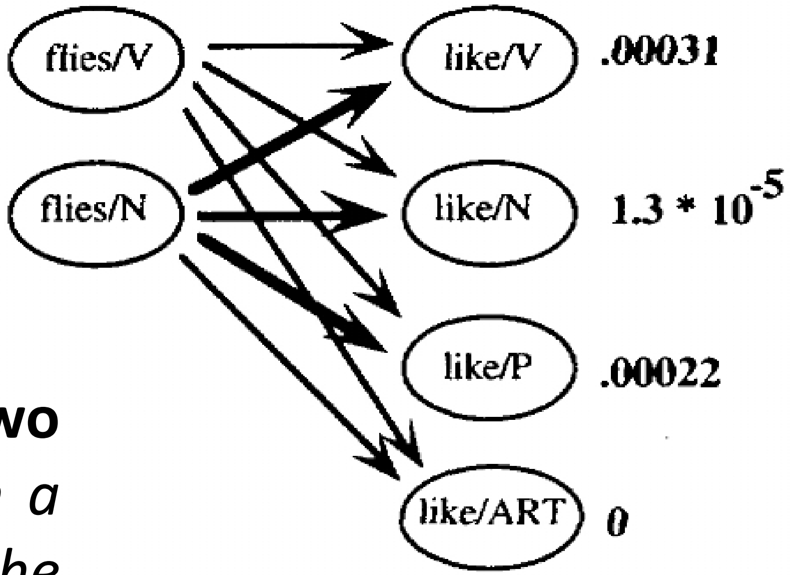
= **3.12 * 10^{-4}**

PROB (like/N) =1.3 * 10^{-5}

PROB (like/P) = 2.2 * 10^{-5}

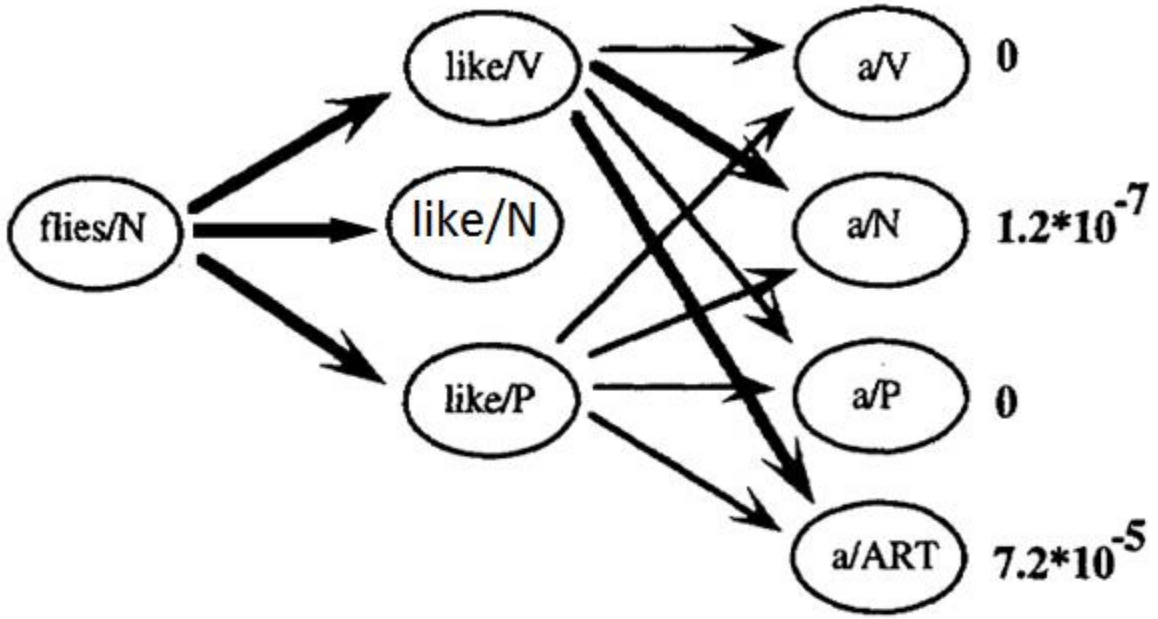
PROB (like/ART) = 0

The most likely **sequence of length two** generating "***Flies like***" *and ending in a* ***V*** *has a score of 3.12 * 10^{-4} (and is the sequence* ***N V****)*



Result after first iteration

- The computation continues in the same manner until each word has been processed.



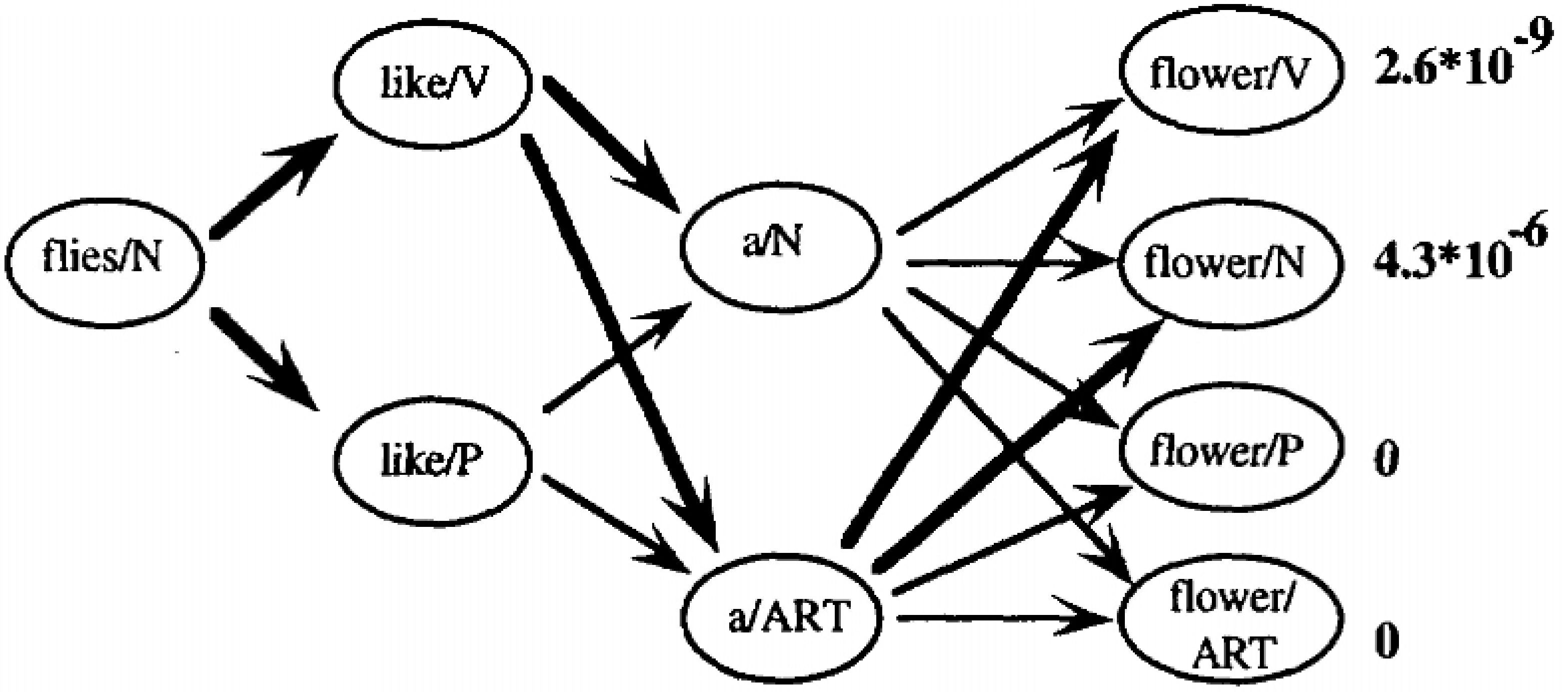
Results after second iteration

PROB (a/ART) = MAX (PROB (like/V) * PROB (ART |V), PROB(like/P) * PROB (ART|P), PROB(like/N) * PROB (ART|N)) * PROB (a/ART)

= MAX(0.00031*0.65, 0.00022*0.74, 1.3x10^{-5} * 0.0001) x 0.360

= **7.254*10^{-5}**

The computation continues in the same manner until each word has been processed.



Results after third iteration [Allen, 1995]

The highest probability sequence ends in state flower/N.
It is easy to back trace from here to get the complete sequence N, V, ART, N.

Results and Conclusion

- Results- HMM tagger gives an accuracy of about 96% when trained and tested on the Wall Street Journal corpus.
- Conclusion-

  i) HMM tagger is a generative probabilistic model for Part of Speech Tagging which does not assign tags to individual words but selects the best tag sequence for the entire sentence.

  ii) The use of Viterbi algorithm speeds up the tagging process by reducing the number of computations in each iteration.