ECE8813
Statistical Natural Language Processing

Lecture 8: String Matching

Chin-Hui Lee

School of Electrical and Computer Engineering

Georgia Institute of Technology

Atlanta, GA 30332, USA

chl@ece.gatech.edu

Comparing Strings: An Overview

- **Problem Statement**
  - Given a reference string *R* and a testing string *U* want to find *D(U, R),* a distance between *U* and *R*
  - Given a reference string model λ_R and a testing string *U* want to find $S(U \mid \lambda_R)$, a score of *U* against a model of *R*
  - But *U* and *R* are composed from smaller elements and of unequal length (dynamic time warping needed)
- **Issues**
  - Element distance (problem-dependent)
  - String modeling and scoring (problem-dependent)
  - Data structure & mapping of match (problem-dependent)
  - Matching algorithms and optimal properties (universal: dynamic programming, or DP, by Bellman, 1959)

Examples of String Matching

- Operations Research (OR) in IE or ISyE
  - many real-world dynamic programming problems
- Network Optimization
  - shortest or longest path, network resource planning
- Project Planning
  - critical path in project design
- Bioinformatics
  - study of DNA, RNA, Protein strings and structures
- Media Processing: Too Many
  - extensively used in all speech applications
  - extensively used in many problems
  - some in image recognition and verification
  - many more to come in the future

String Comparison: Example 1

- Speaker Distance
  - Given a reference speaker template string *R* and a testing template string *U*, want to find *D(U, R)*, a distance between *U* and *R*
  - But *U* and *R* are composed from smaller elements and of unequal length (time warping needed)
- Applications
  - Speaker Identification (SID) $\hat{i} = \arg\min_{1 \le i \le I} D(U, R_i)$
  - Speaker Verification (SV): Accept speaker *i* if

$$D(U, R_i) \le \tau_i$$

String Comparison: Example 2

- Word or phrase distance
  - Given a reference word template string *R* and a testing template string *U*, want to find *D(U, R)*, a distance between *U* and *R*
  - But *U* and *R* are composed from smaller elements and of unequal length (time warping needed)
- Applications
  - Isolated word recognition (IWR): $\hat{i} = \arg\min_{1 \le i \le I} D(U, R_i)$
  - Utterance verification (UV): accept phrase *i* if

$$D(U, R_i) \le \tau_i$$

Time Warping Function

- Distance between strings

$$D(U,R) = \min_{W\in\Gamma} D(W(U,R))$$

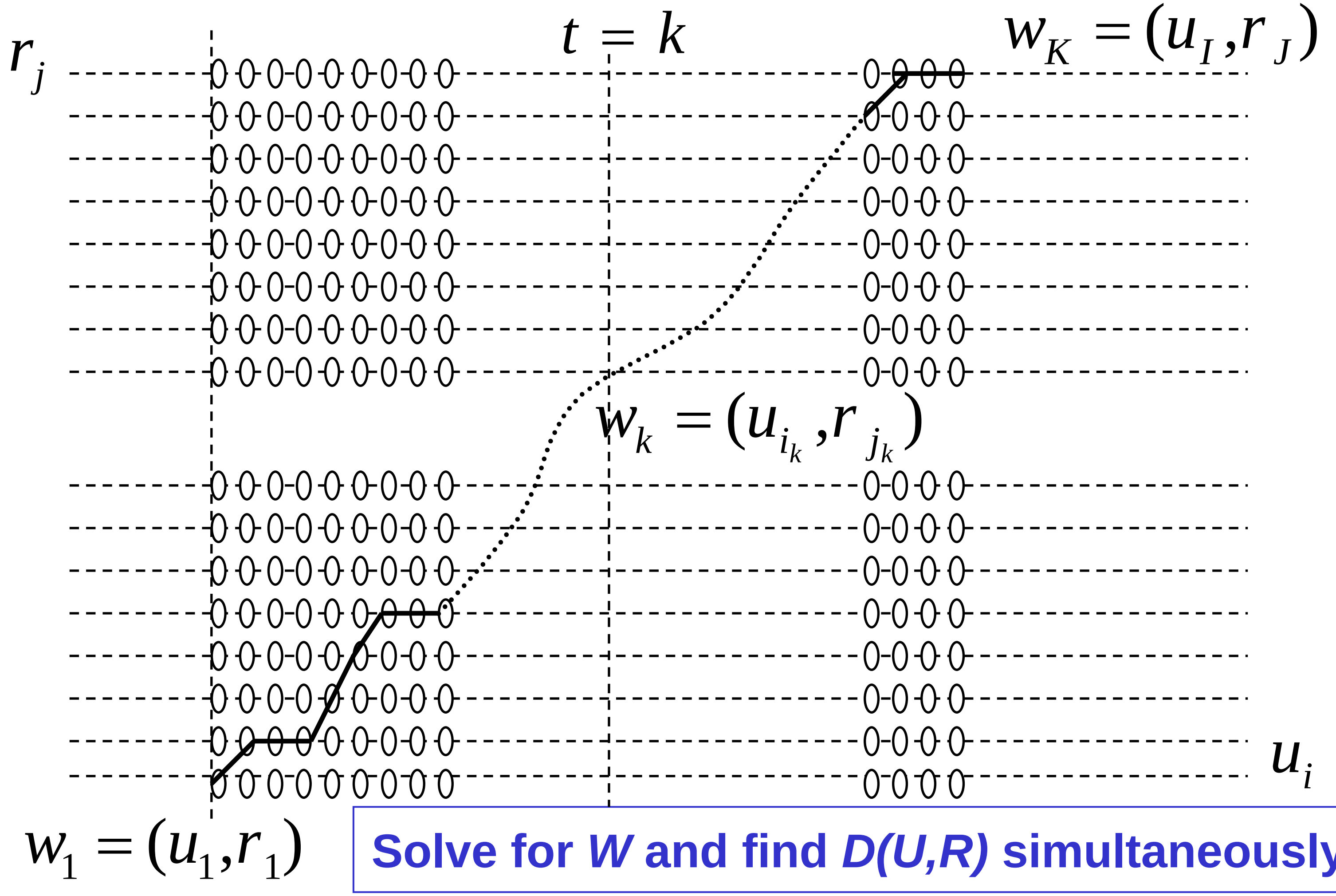
- *W* is a time warping function aligning element pairs from the testing and the reference templates, *U* and *R*

$$U=(u_1,u_2,\cdots,u_I),\quad R=(r_1,r_2,\cdots,r_J)$$

$$W=(w_1,w_2,\cdots,w_K)$$

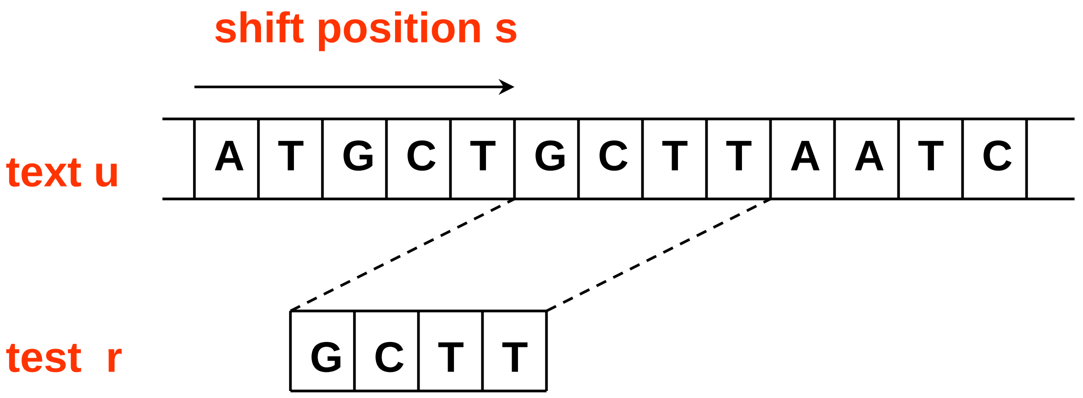
with $w_1=(u_1,r_1)$, $w_k=(u_{i_k},r_{j_k})$ and $w_K=(u_I,r_J)$

Warping Function Trajectory



Solve for *W* and find *D(U,R)* simultaneously?

Example 3: DNA String Matching (unix grep with no space in text)



neucleic acid
- adenine (A)
- guanine (G)
- cytocine (C)
- thymine (T)

- Issues:
  - From exact match to matching with errors
  - From naïve string matching algorithms to the use of optimal string matching with sub-string results
  - Matching with fuzzy distance (keyword spotting)
  - Matching with "don't care" symbols

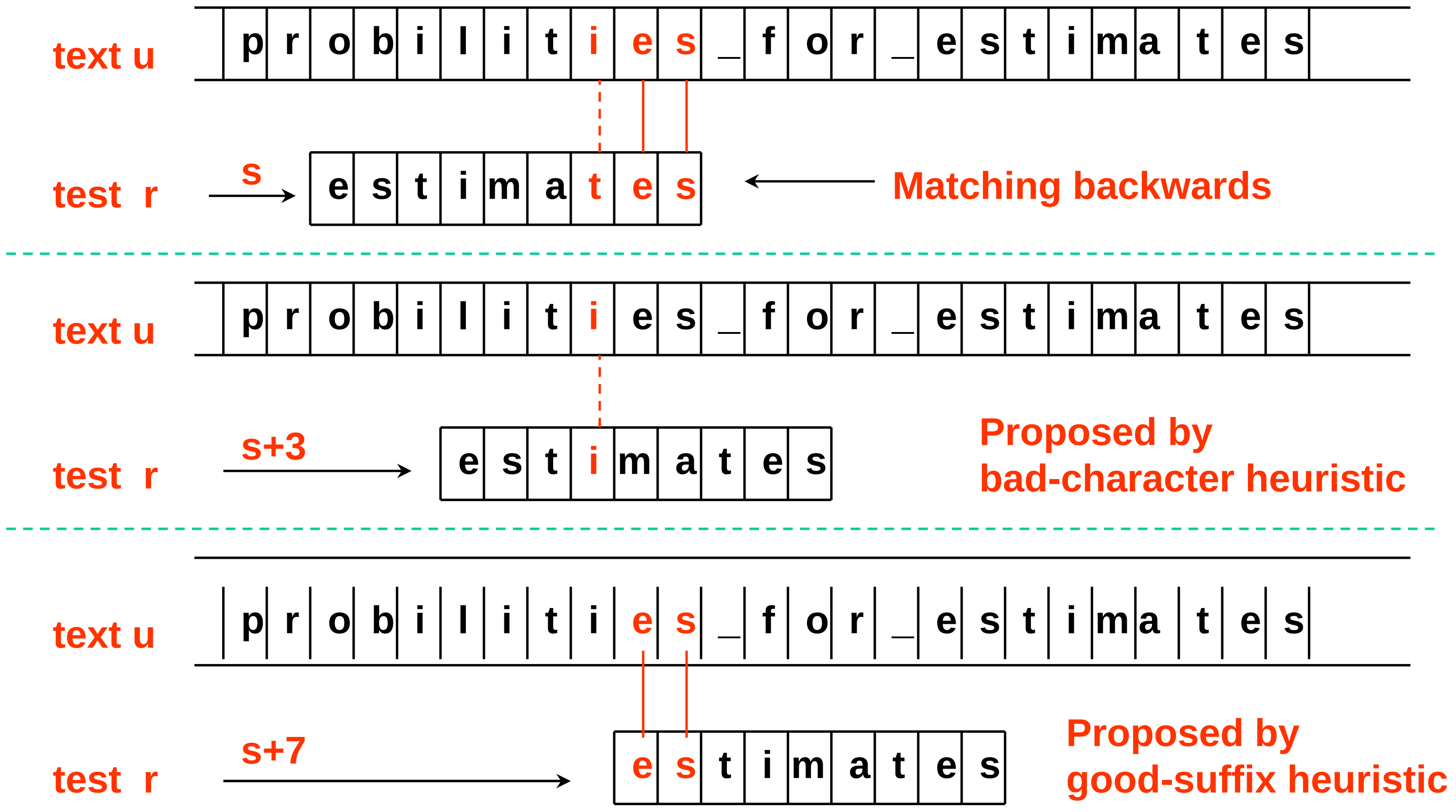
Naïve String Matching Algorithm

```
begin initialize u(), r(), I <- length[U], J <- length[R]
        s <- 0
        while s <= I-J
            if R[1,…,J] = U[s+1,…s+J]
            then print "pattern occurs at shift" s
                    s <- s+1
  return
end
```

} Loop through all shifts exhaustively

- Issues:
  - Not efficient for large *I* and *J*
  - Not easily extendable to other problems
  - Do not make use of sub-string information already computed in earlier iterations

Example 4: Sub-String Information


text u
p r o b i l i t i e s _ f o r _ e s t i m a t e s
test r
s
e s t i m a t e s
Matching backwards
text u
p r o b i l i t i e s _ f o r _ e s t i m a t e s
test r
s+3
e s t i m a t e s
Proposed by bad-character heuristic
text u
p r o b i l i t i e s _ f o r _ e s t i m a t e s
test r
s+7
e s t i m a t e s
Proposed by good-suffix heuristic

Not all shifts needed (7 is better than 3 here)

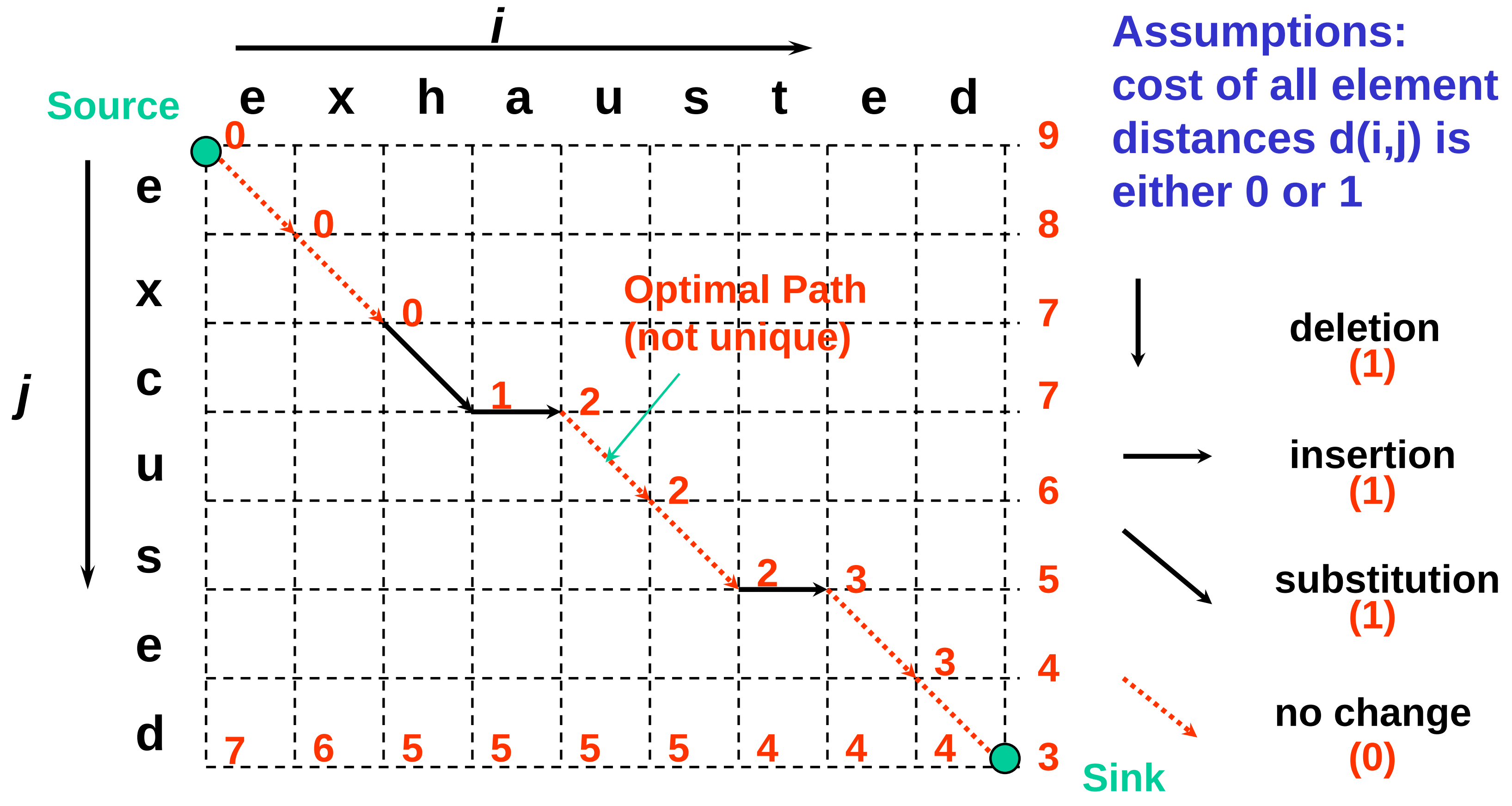
Boyer-Moore String Matching

```
begin initialize  u(), r(), I <- length[U], J <- length[R]
        F(r) <- last-occurrence function
        G(r) <- good-suffix function
        s <- 0
        while s <= I-J
            do j <- J
            while j > 0 and r[j] = u[s+j]
                        do j <- j-1
                        if j=0
                                    then print “pattern occurs at shift” s
                                    s <- s+J
                                    else s <- s+ j - min{G(j), F(u(s+j))}
  return
end
```

Comparing good-suffix & bad-character heuristics

- F(r) – a table containing every letter and the position of its rightmost occurrence in r(.), e.g. F(“a”)=6, F(“e”)=8, F(“i”)=4, F(“m”)=5, F(“s”)=9, F(“t”)=7, the rest F(.)=0
- G(r) – a table that for each suffix in r(.) gives the location of its other occurrences in r(.), it needs to be done only once. G(9)=G(“s”)=2, G(8)=G(“es”)=1, the rest G(.)=0

Example 5: String Edit Distance



- Exercise: Can you find all sub-paths and fill in all the numbers as accumulative distances for all the sub-paths ?

Algorithm for Edit Distance

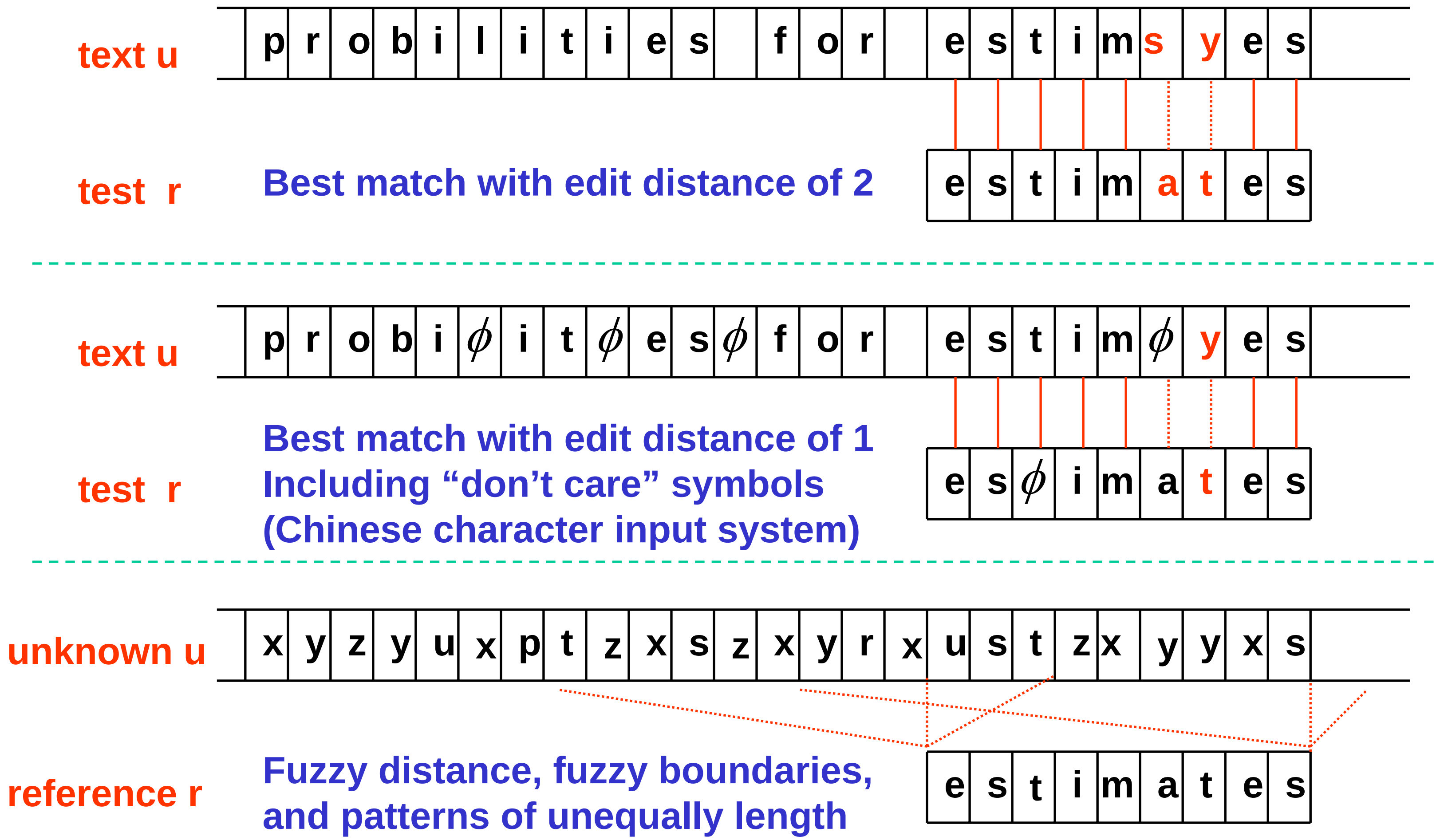
```
begin initialize u(.), r(.), I <- length[U], J <- length[R], D[0,0]=0
        i <- 0
        do i <- i+1
            D[i,0] <- i
        until i = I
        j <- 0
        do j <- j+1
            D[0,j] <- j
        until j = J
        i <- 0; j <- 0
        do i <- i+1
            do j <- j+1   (insertion) (deletion) (substitution or no change)
               D[i,j]=min{D[i-1,j]+1, D[i,j-1]+1, D[i-1,j-1]+q(u(i),r(j))}
            until j = J
        until i = I
    return D[I,J]
end
```

} **Initialize with large distances** (steps from i <- 0 through until j = J)

} **Minimum Edit Distance** (return D[I,J])

q(u(i),r(j)) **is 1 for substitution and 0 for no change**

Example 6: Uncertainty Match

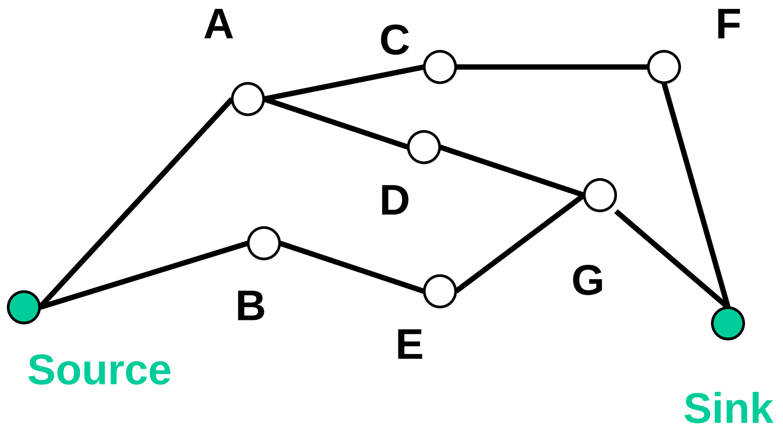

text u
p r o b i l i t i e s f o r e s t i m s y e s
test r
e s t i m a t e s
text u
p r o b i ϕ i t ϕ e s ϕ f o r e s t i m ϕ y e s
test r
e s ϕ i m a t e s
unknown u
x y z y u x p t z x s z x y r x u s t z x y y x s
reference r
e s t i m a t e s

Best match with edit distance of 2

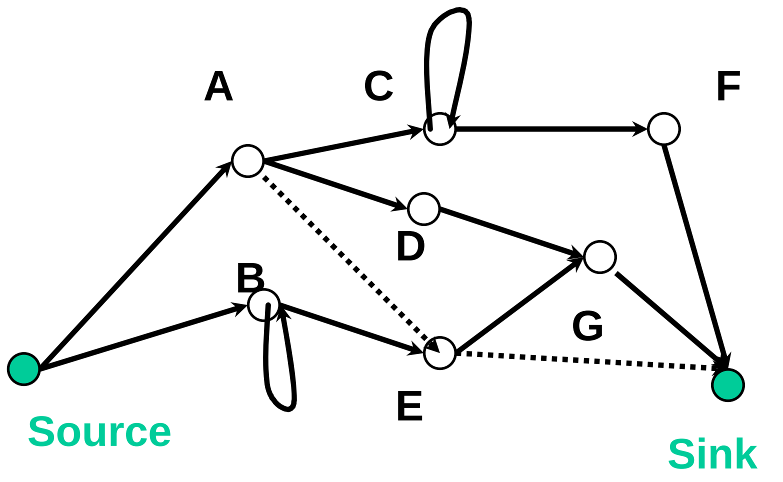
Best match with edit distance of 1
Including "don't care" symbols
(Chinese character input system)

Fuzzy distance, fuzzy boundaries, and patterns of unequally length

Graph Representation

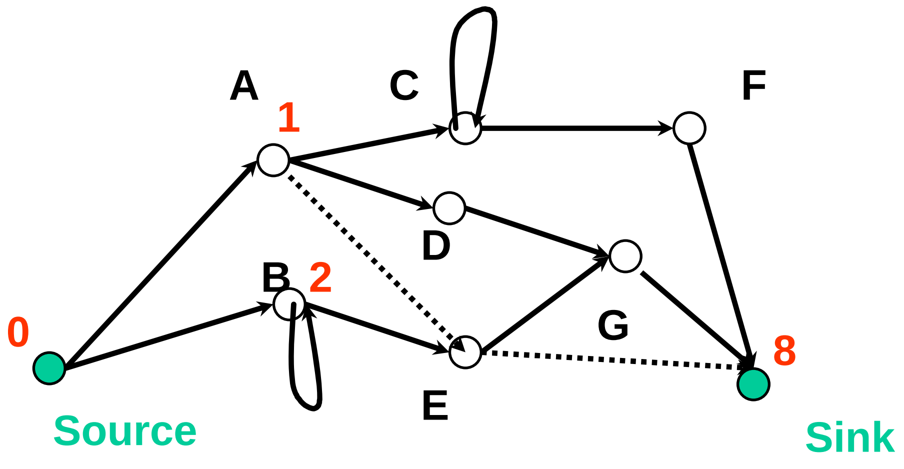


- Graph - *G=G(V, E)*
- Vertex Set - *V*
- Edge Set - *E*



- Digraph - *G=G(V, A)*
- Vertex Set - *V*
- Arc Set - *A* (*E* with directions)
- Self-loop allowed
- Null (lambda) arc included
- Cost can be assigned to arc traversal and node occupancy

Mapping of String Match into Graph Search Problem



For node B (labeled 2)

$$I_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad n_2 \qquad c_2 = \begin{bmatrix} c_{02} \\ \infty \\ c_{22} \\ \vdots \\ \infty \end{bmatrix}$$

- Labeling of nodes and arcs
- Prepare connection matrix

$$I_{ij} = \begin{cases} 1 & \text{if nodes i and j are connected} \\ 0 & \text{otherwise} \end{cases}$$

- Assign cost to for staying each node ***j***: n_j
- Assign cost for going through each arc ***(I, j)*** reaching each node (preparing a cost matrix): c_{ij}

The cost for traveling an arc from node i to j then staying in node j (sometimes called state):

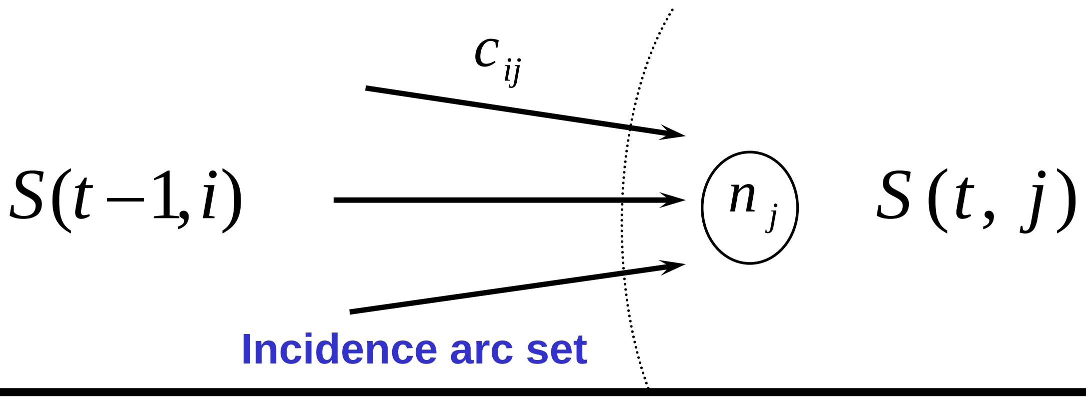
$$s_{ij} = c_{ij} + n_j$$

- Repeat the computation for each time
- Initialize total cost at each node at a high value

Dynamic Programming (DP) Theory

- Bellman's *Principle of Optimality*
  - An optimal path is consisted of only optimal sub-paths reaching to any node at any time
- Dynamic Programming Algorithm
  - At any time *t*, construct only optimal sub-paths to all nodes (or states) from all *active* states at time *t-1* (DP recursion)
  - Ideally for decoding in finite state networks

$$S(t, j) = \min_i [S(t-1, i) + c_{ij} + n_j]$$



Viterbi Decoding Algorithm

- Define optimal partial path score

$$\delta_i(t) = \max_{s_1^{t-1}} P(s_1^{t-1}, s_t = i, O_1^t \mid \Lambda)$$

  - Initialization $\delta_i(0) = \pi_i$
  - DP-recursion and history bookkeeping

$$\delta_j(t) = \max_{1 \le i \le N} [\delta_i(t-1) a_{ij}] b_j(o_t) \quad 1 \le t \le T \quad 1 \le j \le N$$

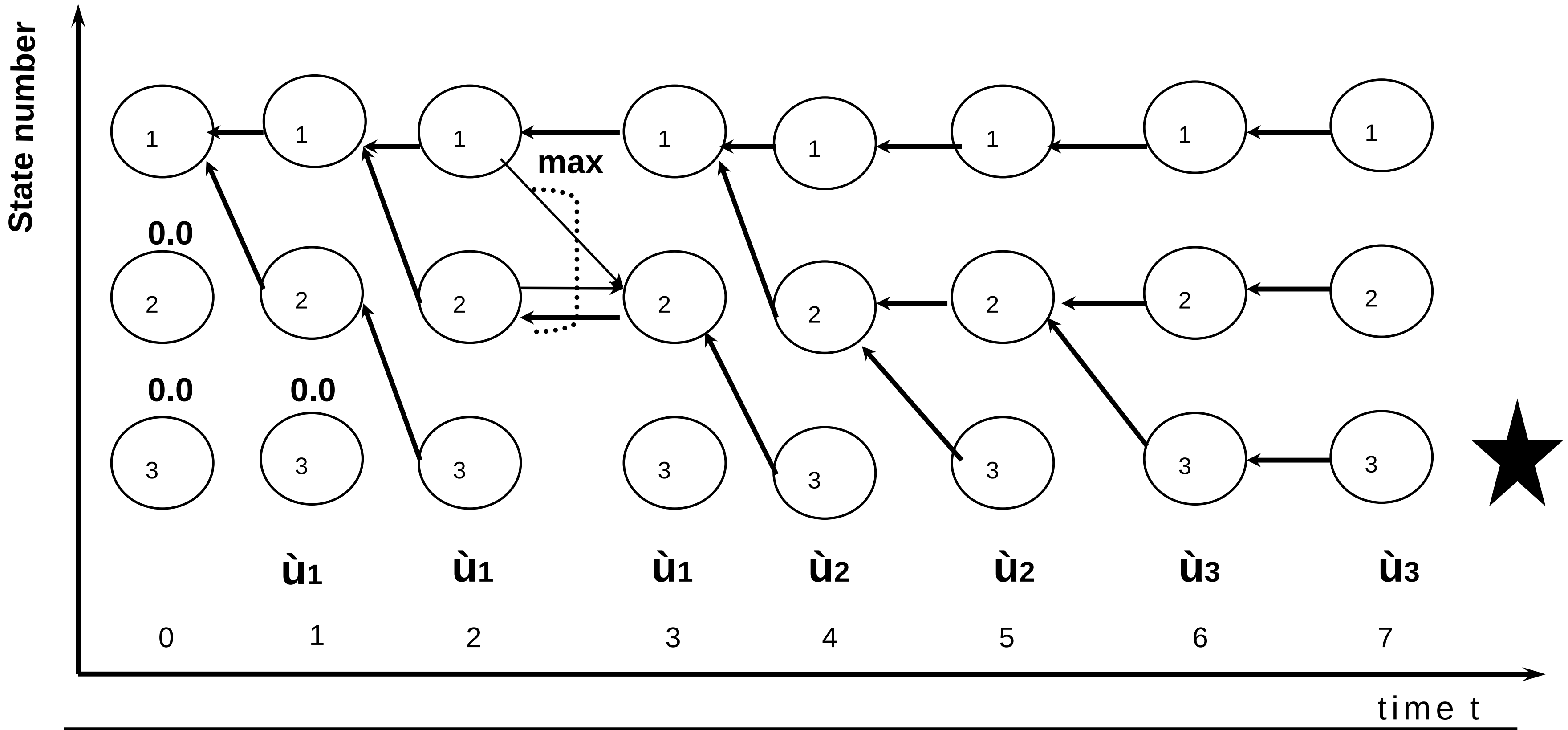
$$\psi_j(t) = \arg\max_{1 \le i \le N} [\delta_i(t-1) a_{ij}] \quad 1 \le t \le T \quad 1 \le j \le N$$

  - Termination $P_{\max} = \max_S p(S, O \mid \Lambda) = \max_{1 \le i \le N} \delta_i(T)$ and $\hat{s}_T = \arg\max_{1 \le j \le N} \psi_j(T)$
  - Path backtracking $\hat{s}_{t-1} = \psi_{\hat{s}_t}(t) \quad \mathrm{t = T, T\text{-}1, \ldots, 2}$
  - "Optimal" State Sequence: $\hat{\mathrm{s}} = (\hat{s}_1, \ldots, \hat{s}_T)$

Viterbi Decoding Algorithm: Trellis

Example: 3-state left-right HMM (more later)

For an observation *O={o1,o2,o3,o4,o5,o6,o7}*



Dynamic Programming: An Example

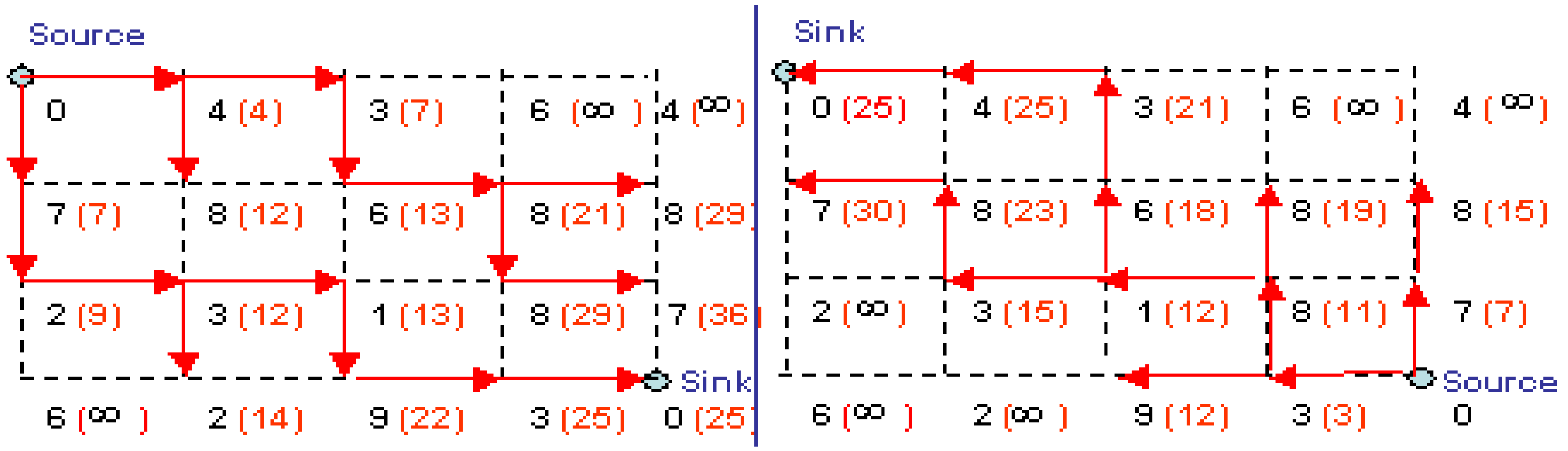
- The following 4×5 matrix of numbers shows a 20-state network, with each *(i, j)* network element indicating the cost of visiting that state.

$$\begin{matrix} 0 & 4 & 3 & 6 & 4 \\ 7 & 8 & 6 & 8 & 8 \\ 2 & 3 & 1 & 8 & 7 \\ 6 & 2 & 9 & 3 & 0 \end{matrix}$$

- Suppose it is desired to move from the upper left corner state, i.e. the source at state (1, 1), to the lower right corner state, the sink at state (4, 5), and an individual movement is allowed only one step rightward or downward, but not both (i.e. moving diagonally in the southeast direction). There is also an additional constraint that you cannot make more than three rightward or downward transitions on the same row or column in the same path (this is similar to a durational or slope constraint in DTW). Find the minimum cost path, the states never visited by any sub-path due to the durational constraint, and the corresponding minimum path cost. Explain in steps how you arrive at the answers. You can break ties arbitrarily when choosing same cost sub-paths. Do you get the same answer if you work backwards?

Dynamic Programming: Solution

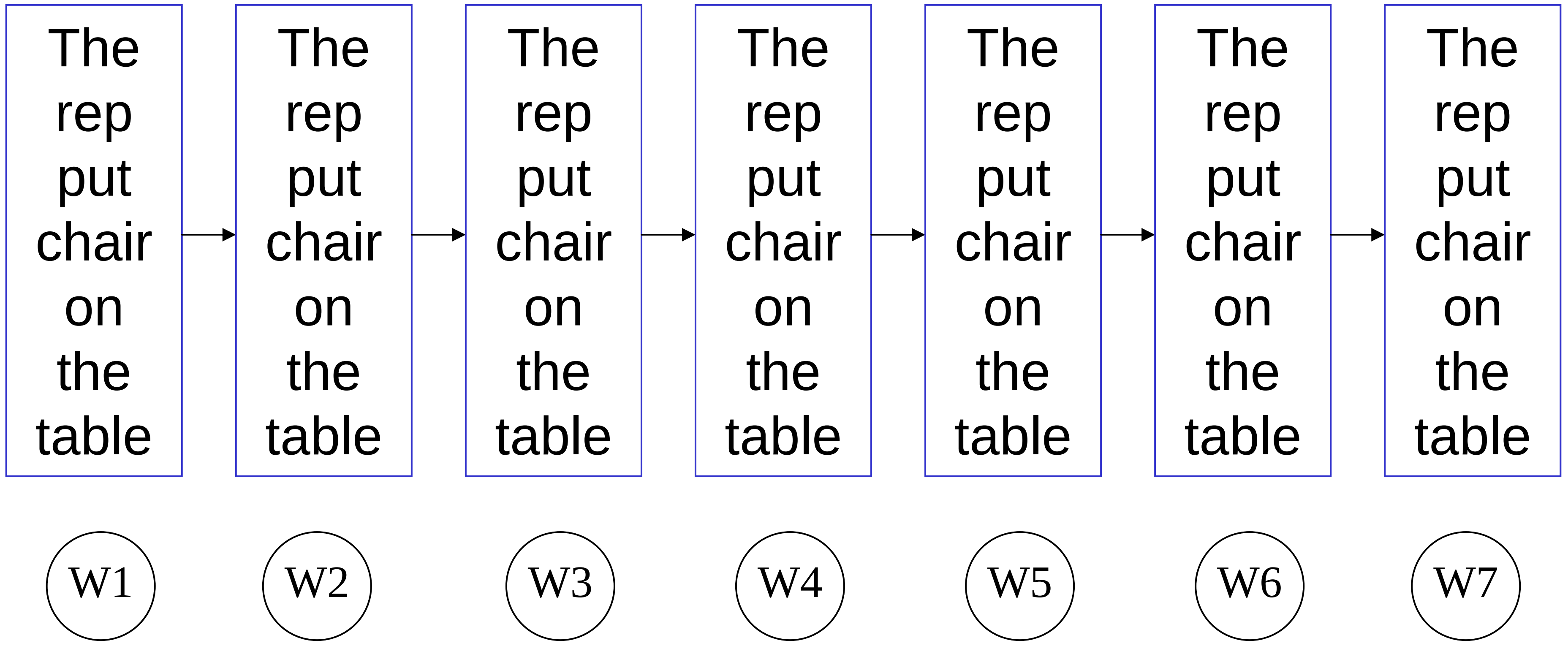
Find the minimum cost path in a network using dynamic programming by considering all sub-paths reachable in the process, and eliminating all sub-paths that are no longer competitive (non-optimal) in the eventual optimal path.



Optimal Forward Path: (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (4, 3), (4, 4), (4, 5)
Minimum Cost: 25 (shown in the parentheses are costs of optimal sub-paths
States never visited due to the duration constraint: (4, 1), (1, 4), (1, 5)

Optimal Reverse Path: (4, 5), (4, 4), (3, 4), (3, 3), (2, 3), (1, 3), (1, 2), (1, 1), with same cost 25
States never visited due to the duration constraint: (4, 1), (4, 2), (3, 1), (1,), (1,5)

Word Sorting from a Bag of Words


The rep put chair on the table
The rep put chair on the table
The rep put chair on the table
The rep put chair on the table
The rep put chair on the table
The rep put chair on the table
The rep put chair on the table
W1
W2
W3
W4
W5
W6
W7

$$\arg\max_W \log P(W) \approx \arg\max_{W_1^7} \sum_{k=1}^{7} \log P(w_k \mid w_{k-1})$$ bigram approximation

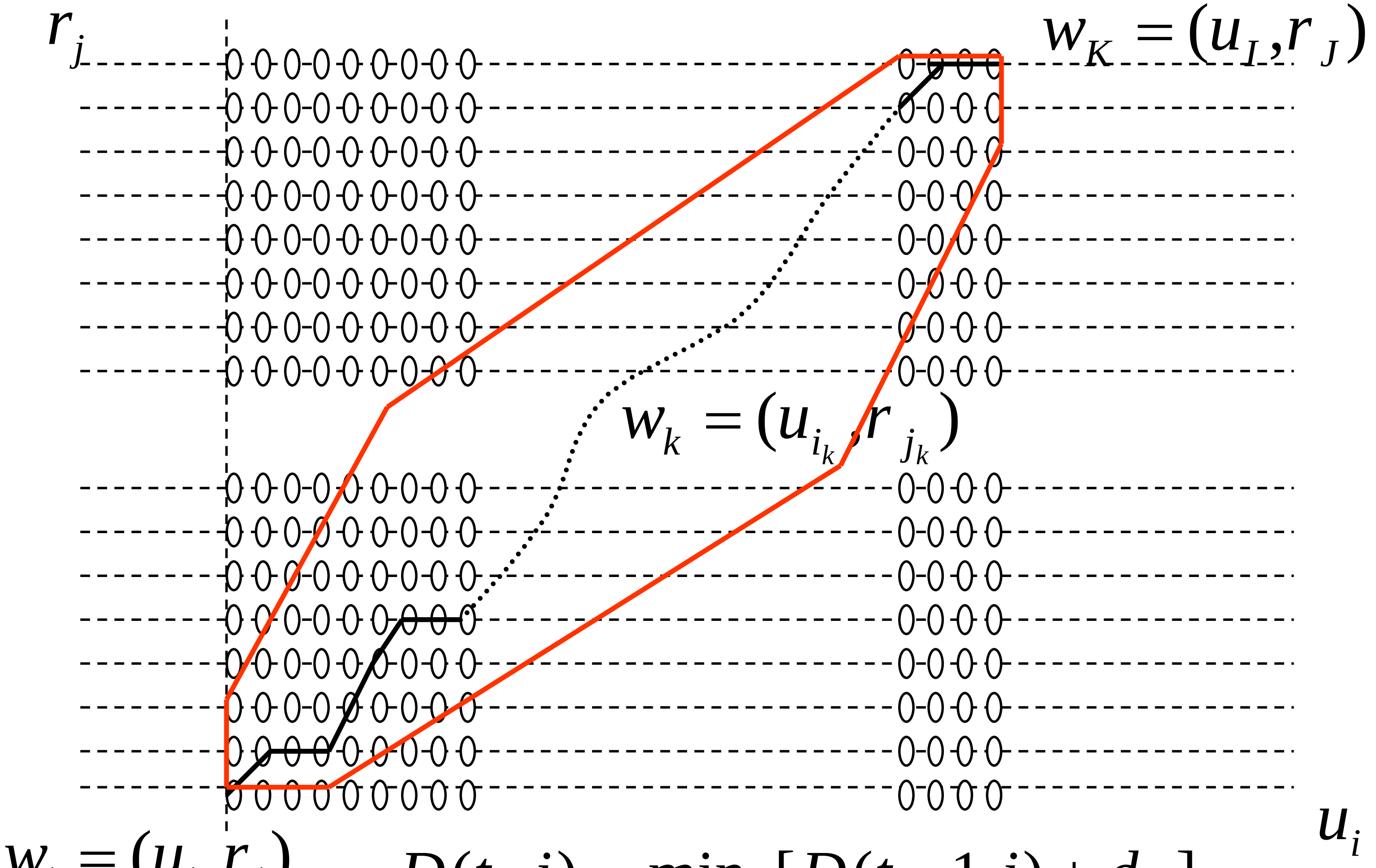
Typical DTW Constraints

- Monotony & Continuity Condition
  - *W*(.) is monotone and continuous
- Boundary Condition

$$w_1 = (u_1, r_1) \quad \textbf{and} \quad w_K = (u_I, r_J)$$

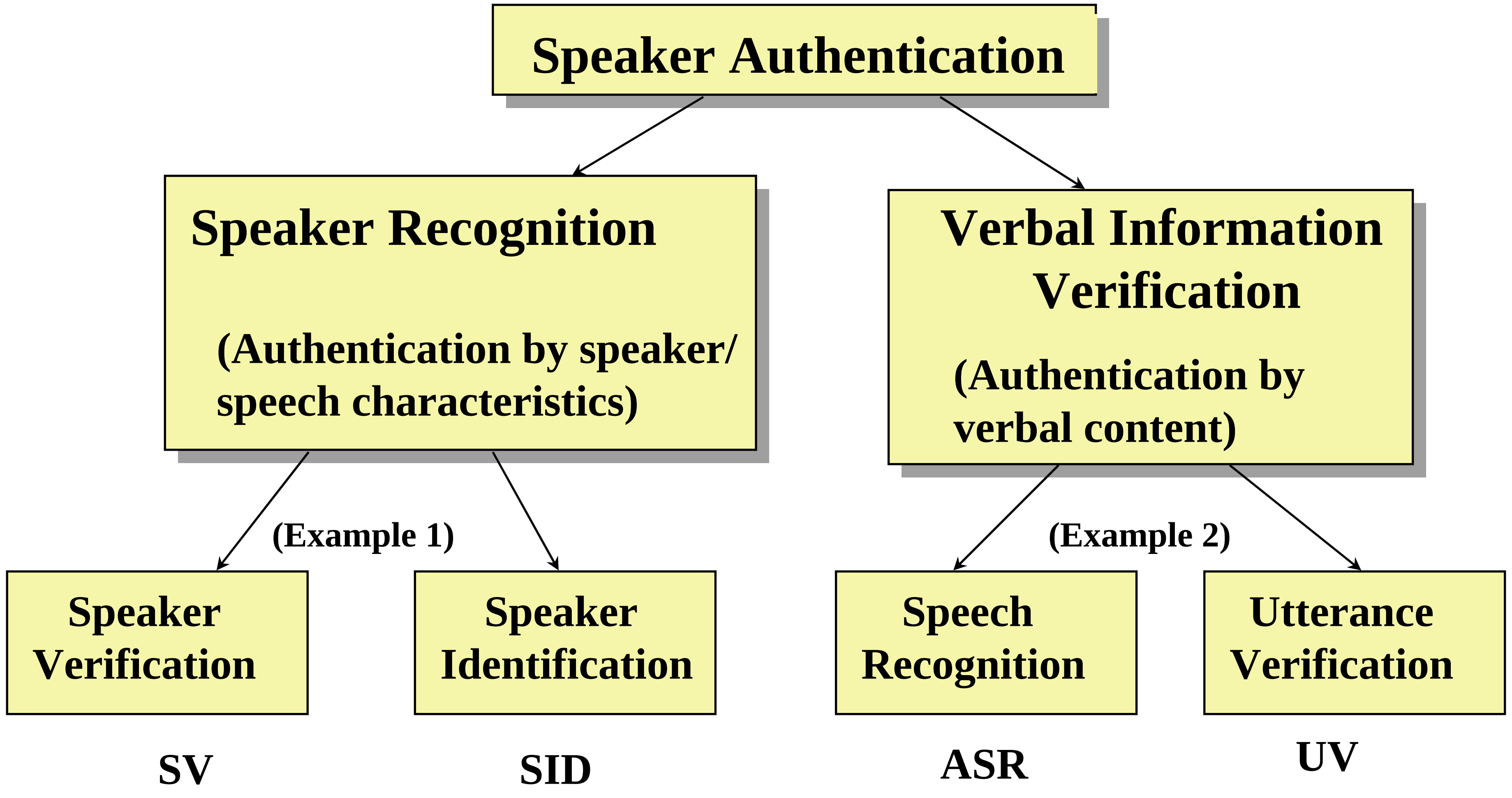
- Adjustment Window Condition
  - Prevent arbitrary expansion or contraction
- Contract to a small number of states first
  - HMM and Viterbi algorithm
- Others
  - Slope & weighting conditions, e.g. ½<=slope<=2

Warping Trajectory Constraints



$$D(t, j) = \min_i [D(t-1, i) + d_{ij}]$$

Speech for Biometric Authentication



Back to Example 1: SID and SV

- Applications (same for fingerprint, etc.)
  - Speaker Identification (SID)

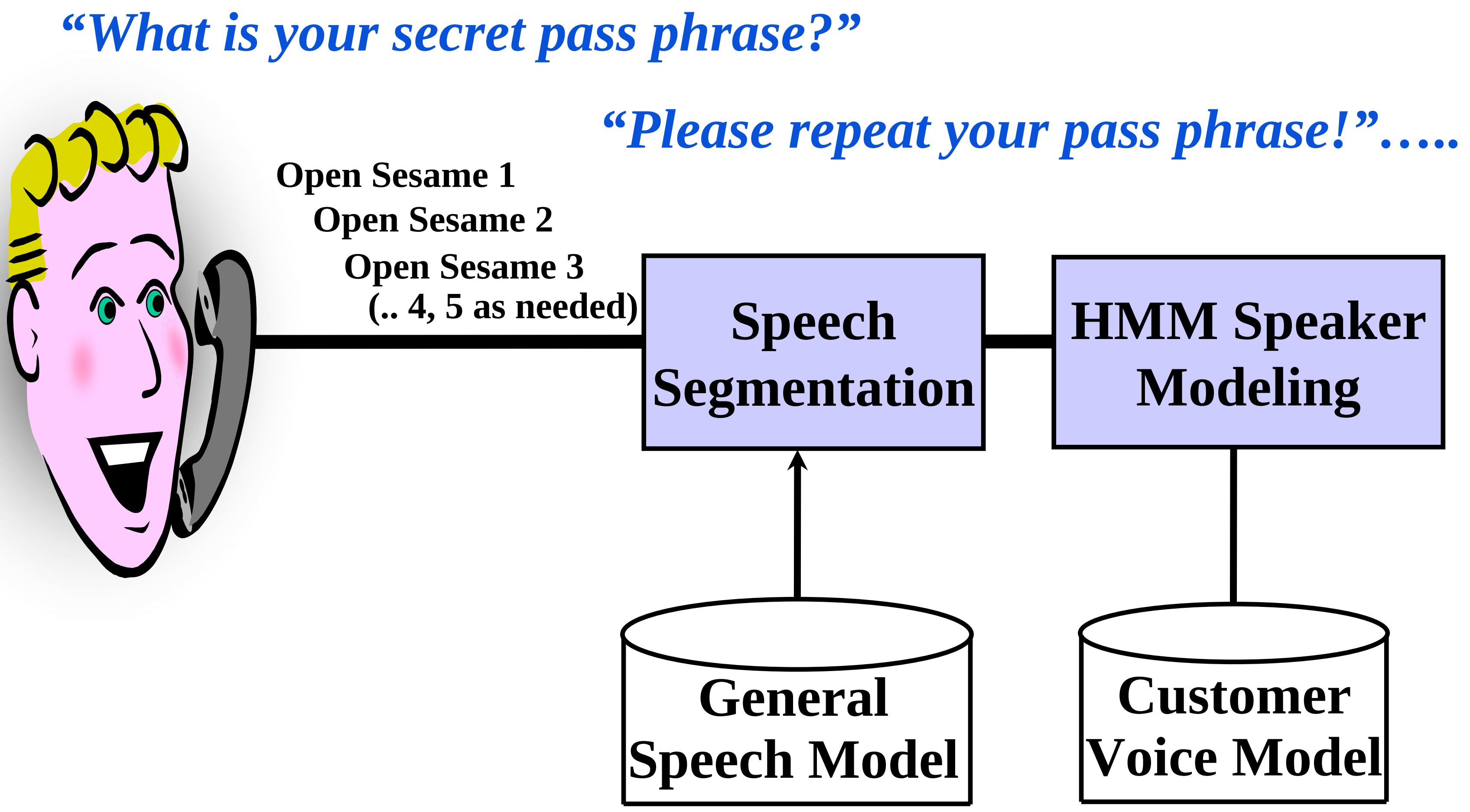
$$\widehat{i} = \arg\min_{1 \le i \le I} D(U, R_i)$$

  - Speaker Verification (SV): cccept speaker *i* if

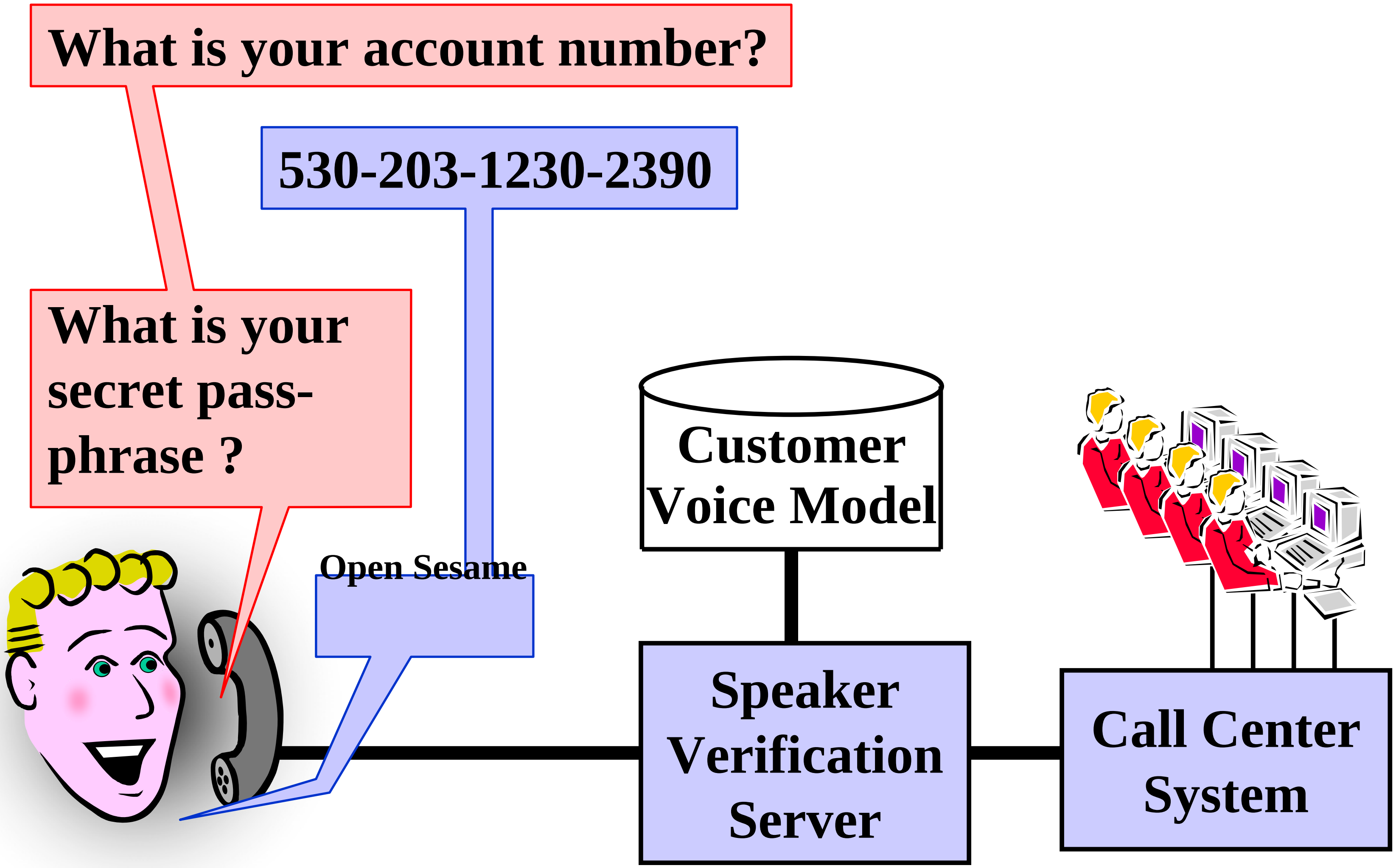
$$D(U, R_i) \le \tau_i$$

- Enrollment stage: collect samples to prepare reference templates (training)
- Testing stage: prepare testing template, compare distances, and make decisions

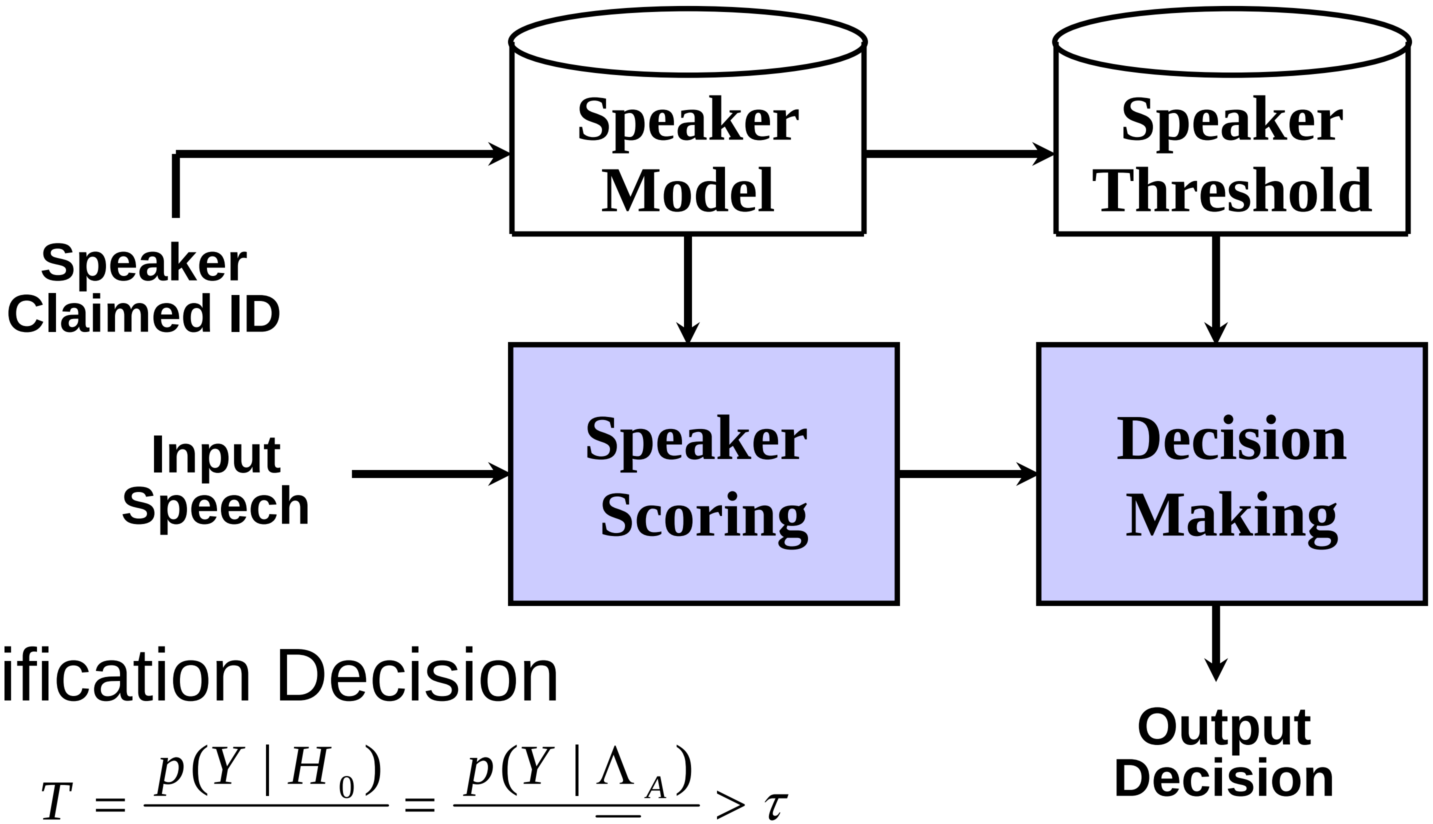
SV Enrollment & Speaker Modeling



Speaker Verification (SV)


What is your account number?
530-203-1230-2390
What is your secret pass-phrase ?
Open Sesame
Customer Voice Model
Speaker Verification Server
Call Center System

Speaker Verification & Biometric Authentication (Voice Print)



- Verification Decision

If $T = \frac{p(Y \mid H_0)}{p(Y \mid H_1)} = \frac{p(Y \mid \Lambda_A)}{p(Y \mid \overline{\Lambda}_A)} > \tau$

accept the user as A; otherwise, reject the user.

Summary

- **Today's Class**
  - String matching
- **Next Class**
  - N-gram on Feb. 5 and 10 (be prepared)
- **Homework**
  - **Lab2 assigned (due on Feb 10)**
- **Reading Assignments**
  - Manning and Schutze, Chapters 5, 6 & 7