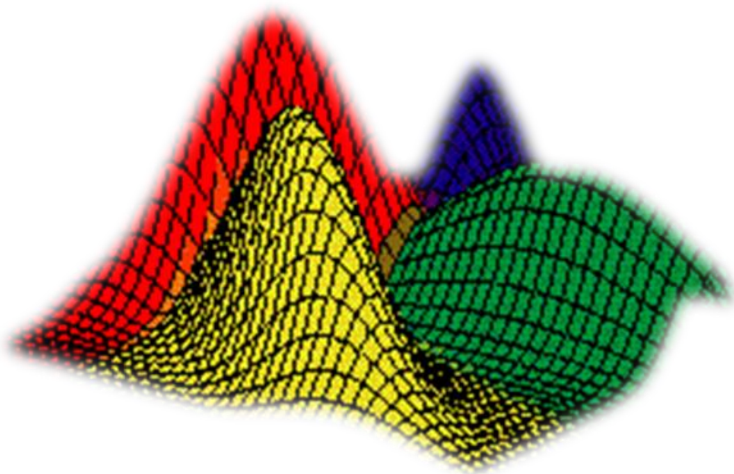
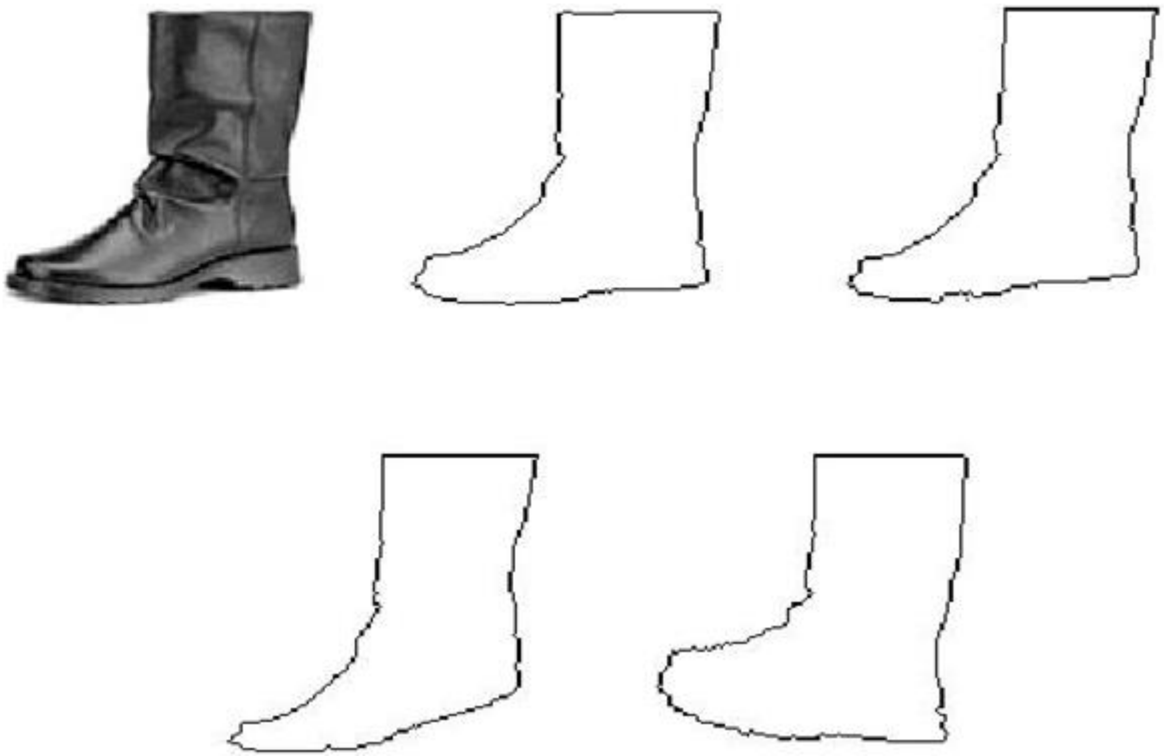


Pattern Recognition

Template Matching



Template Matching

- Typical Applications
  - Speech Recognition
  - Motion Estimation in Video Coding
  - Data Base Image Retrieval
  - Written Word Recognition
  - Bioinformatis

Template Matching

- The Goal:
  - Given a set of reference patterns known as TEMPLATES,
  - find the best match for unknown pattern
  - each class represented by a single typical pattern.

- requires an appropriate "measure" to quantify similarity or matching.

Template Matching

- The cost "measure" :
  - deviations between the template and the test pattern.
  - For example:
  - The word beauty may have been read a beeauty or beuty, etc., due to errors.
  - The same person may speak the same word differently.

Template Matching Method

- Optimal path searching techniques
- Correlation
- Deformable models

TM using Optimal Path Searching

- Representation: Represent the template by a sequence of measurement vectors or string patterns

Template: $\underline{r}(1), \underline{r}(2), \ldots, \underline{r}(I)$

Test pattern: $\underline{t}(1), \underline{t}(2), \ldots, \underline{t}(J)$

TM using Optimal Path Searching

Template: $\underline{r}(1), \underline{r}(2), \ldots, \underline{r}(I)$

Test pattern: $\underline{t}(1), \underline{t}(2), \ldots, \underline{t}(J)$

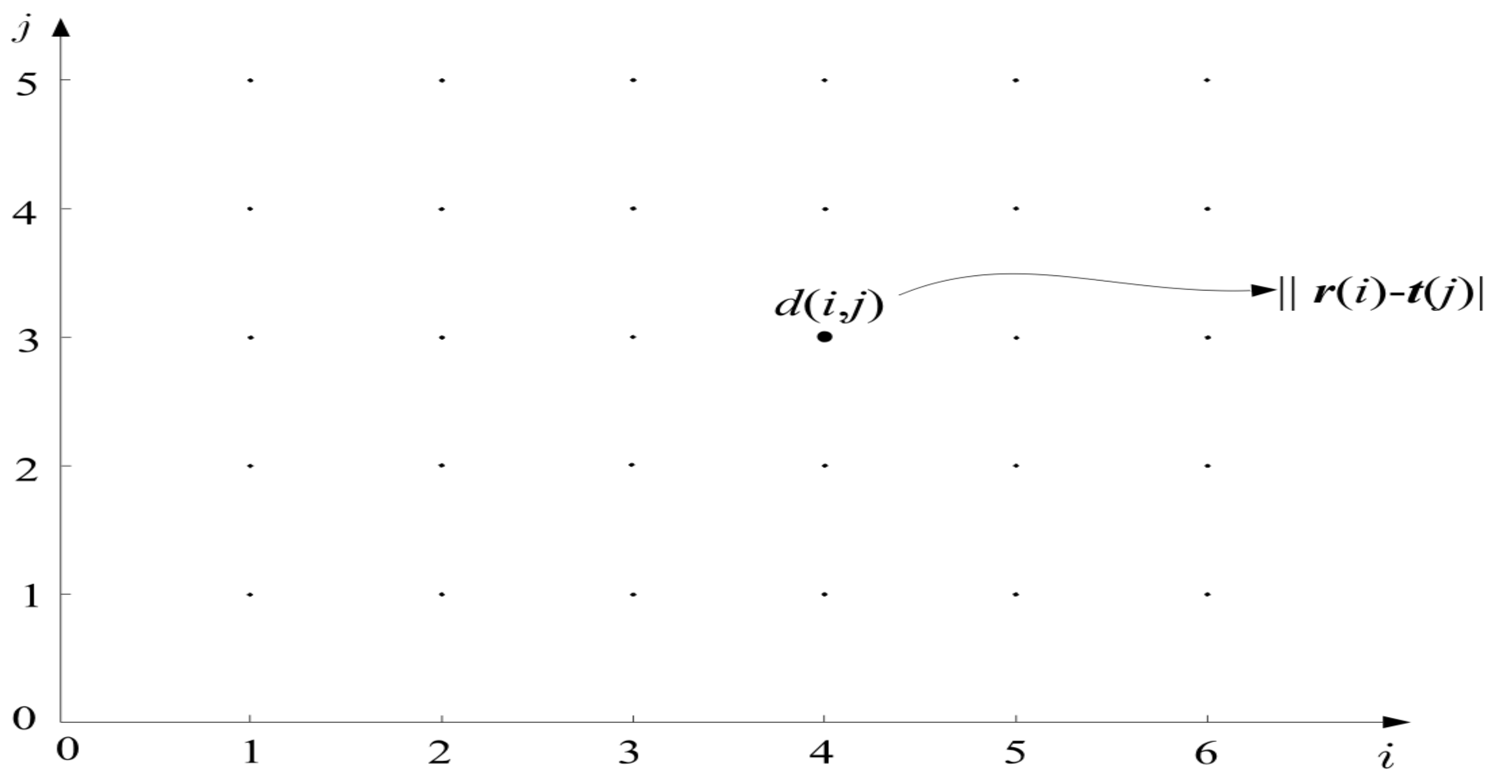
– In general $I \neq J$

– We need to find an appropriate distance measure between test and reference patterns.

TM using Optimal Path Searching

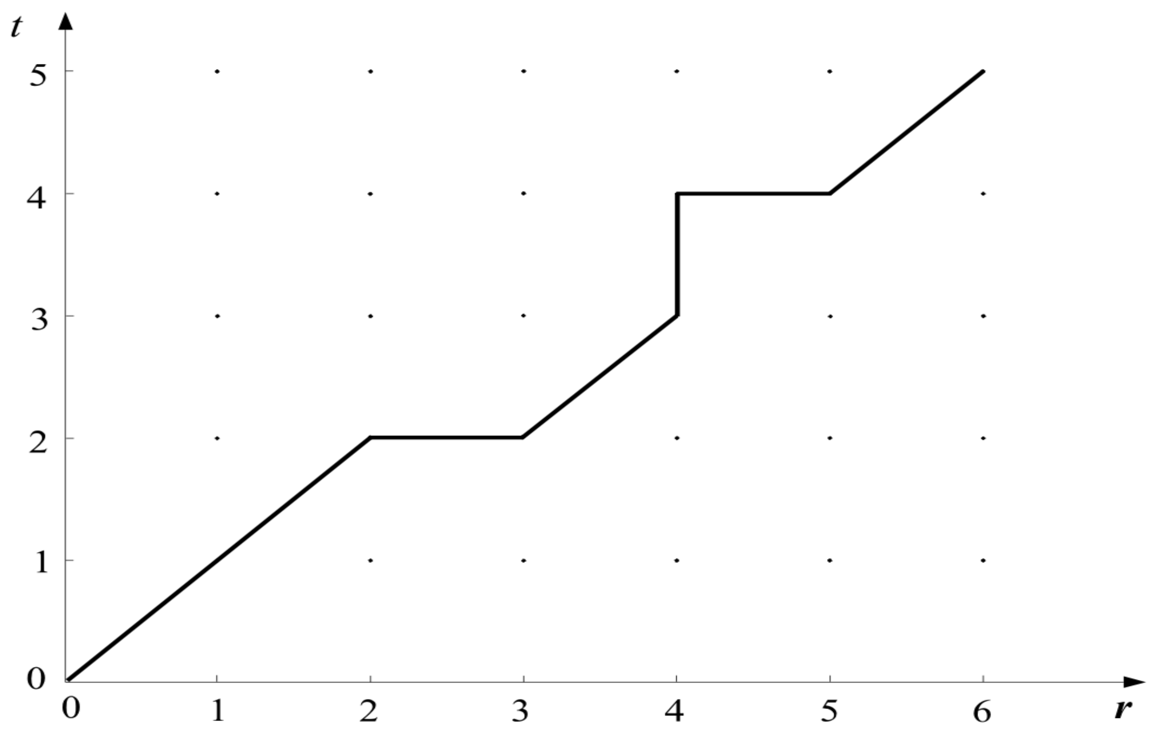
– Form a grid with I points (template) in horizontal and J points (test) in vertical

– Each point (i,j) of the grid measures the distance between $\underline{r}(i)$ and $\underline{t}(j)$



TM using Optimal Path Searching

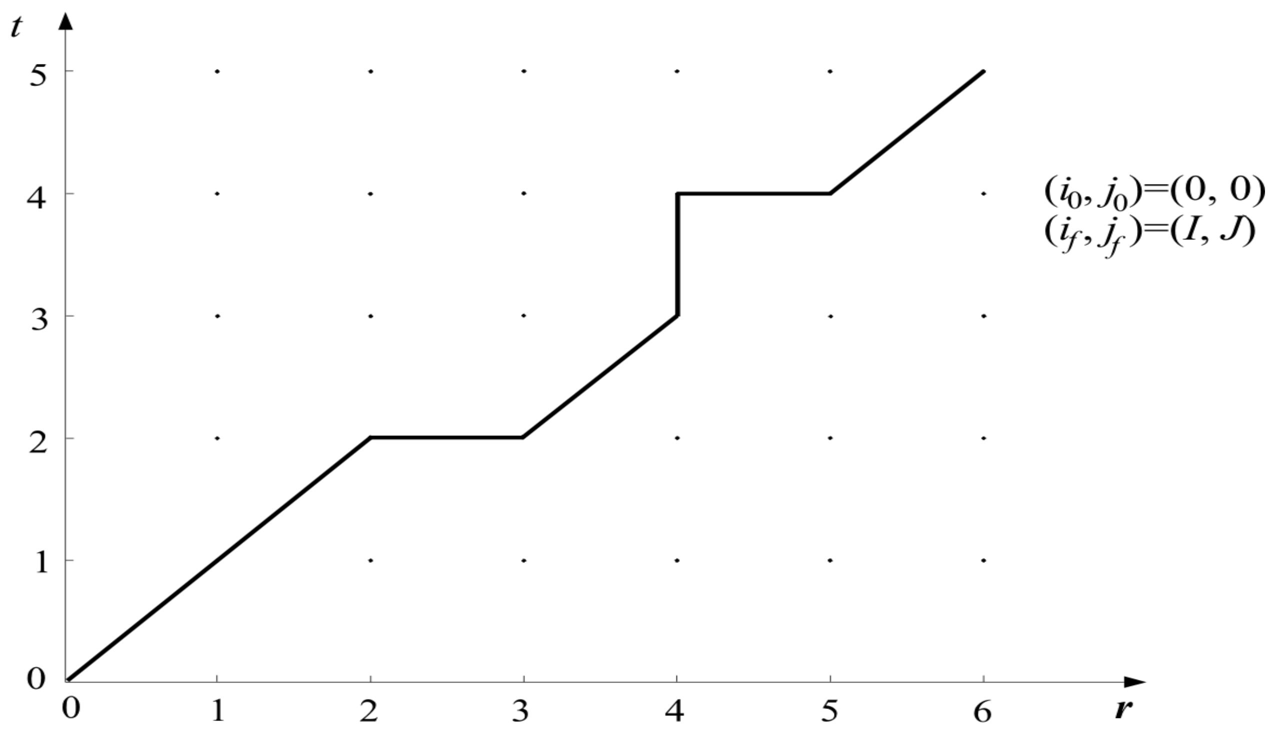
– Path: A path through the grid, from an initial node (i_0, j_0) to a final one (i_f, j_f), is an ordered set of nodes $(i_0, j_0), (i_1, j_1), (i_2, j_2) \ldots (i_k, j_k) \ldots (i_f, j_f)$



TM using Optimal Path Searching

– Path: A path is complete path if:

$$(i_0, j_0) = (0, 0), (i_1, j_1), (i_2, j_2), \ldots, (i_f, j_f) = (I, J)$$

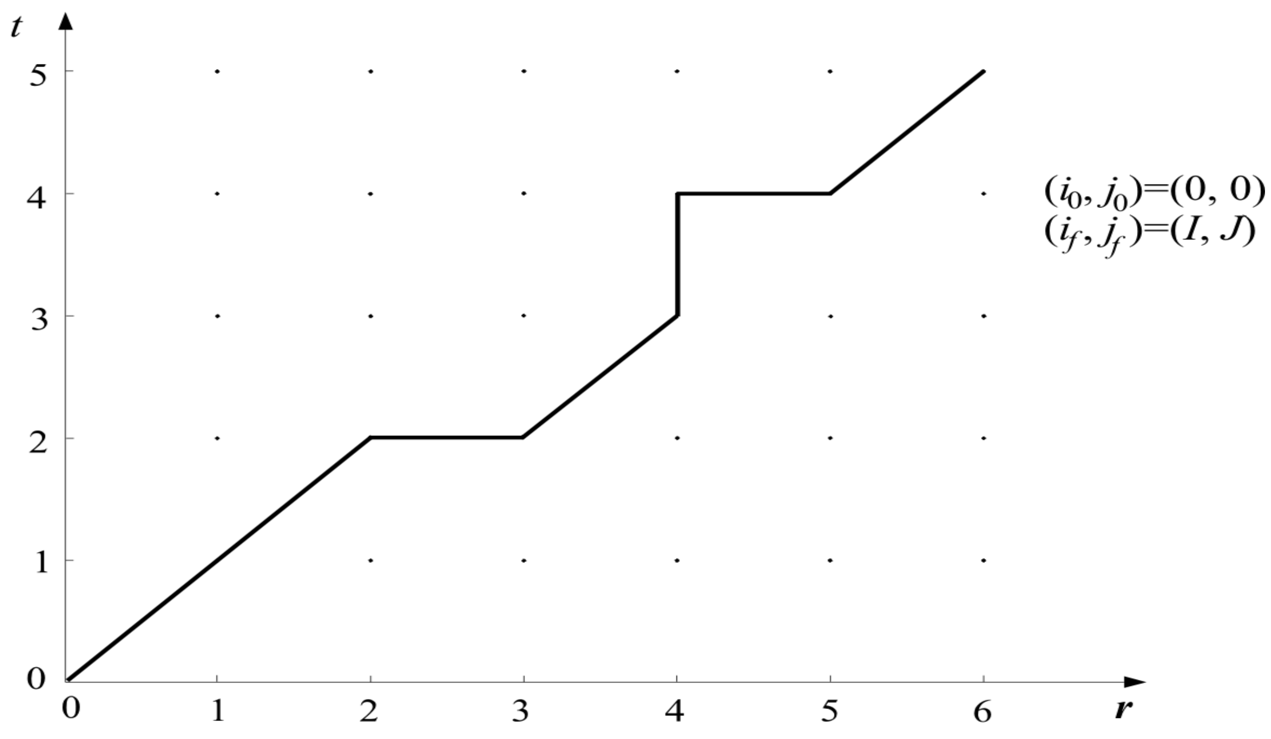


TM using Optimal Path Searching

– Each path is associated with a cost

$$D=\sum_{k=0}^{K-1} d(i_k , j_k)$$

where K is the number of nodes across the path



TM using Optimal Path Searching

- The cost up to node (i_k, j_k) is: $D(i_k, j_k)$
- By convention
  - $D(0, 0)=0$
  - $d(0,0)=0$

TM using Optimal Path Searching

- The equation

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

  assumes that each node has been associated with some cost

TM using Optimal Path Searching

- The equation

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

assumes that each node has been associated with some cost

- However, each transition (i_{k-1}, j_{k-1}) to (i_k, j_k) may also associate with a cost
- The new equation is:

$$D = \sum_{k} d(i_k, j_k | i_{k-1}, j_{k-1})$$

TM using Optimal Path Searching

$$D = \sum_{k} d(i_k, j_k | i_{k-1}, j_{k-1})$$

- Search for the path with the optimal cost $D_{opt.}$
- The matching cost between template $\underline{r}$ and test pattern $\underline{t}$ is $D_{opt.}$
- Costly operation
- Needs efficient computation

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

- Let (i,j) be an intermediate node, i.e.

$$(i_0, j_0) \to \ldots \to (i, j) \to \ldots \to (i_f, j_f)$$

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

- Let (i,j) be an intermediate node, i.e.

$$(i_0, j_0) \rightarrow \ldots \rightarrow (i, j) \rightarrow \ldots \rightarrow (i_f, j_f)$$

Then write the optimal path through (i, j)

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f)$$

Bellman's Optimality Principle

- Bellman's Principle:

$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$ can be obtained as

$$(i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

- meaning: The overall optimal path from (i_0,j_0) to (i_f,j_f) through (i,j) is the concatenation of the optimal paths from (i_0,j_0) to (i,j) and from (i,j) to (i_f,j_f)

Bellman’s Optimality Principle

- Bellman’s Principle:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f) \Leftrightarrow (i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

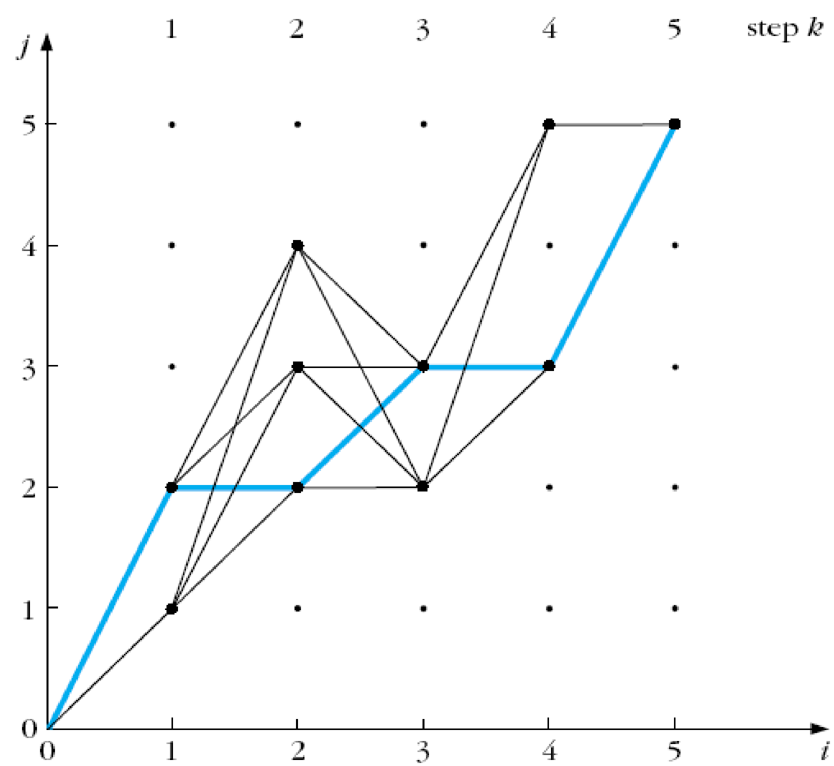
- Let $D_{opt.}(i_{k\text{-}1}, j_{k\text{-}1})$ is the optimal path to reach $(i_{k\text{-}1}, j_{k\text{-}1})$ from (i_0, j_0), then Bellman’s principle is stated as:

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

Bellman's Optimality Principle

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

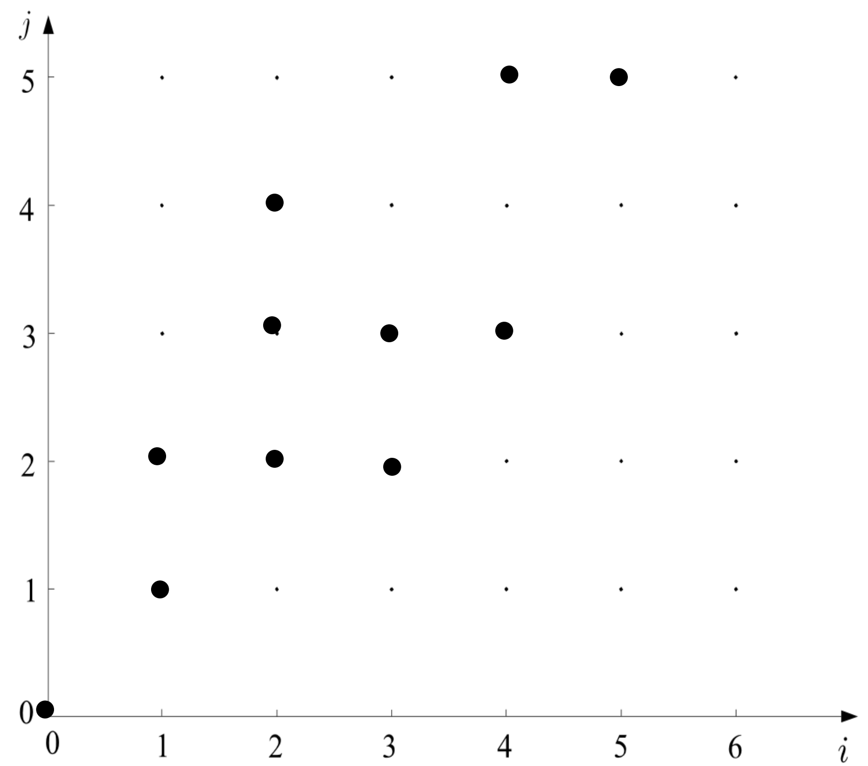
- We don't need to search the whole space to find the optimal path
- Global and local constraints may be imposed to reduce the search space



Bellman's Optimality Principle

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

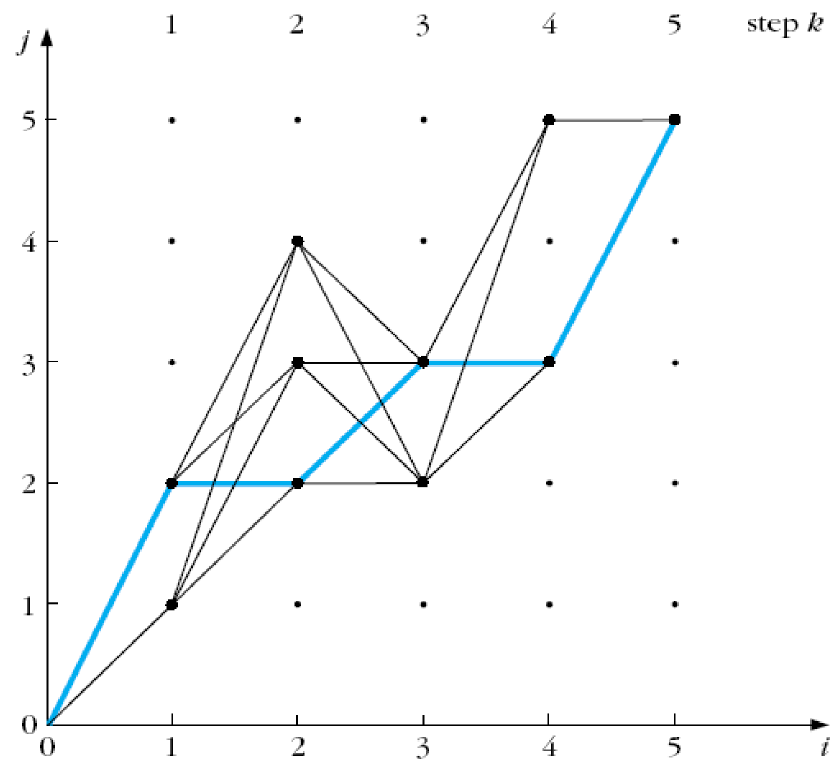
- We don't need to search the whole space to find the optimal path
- Global and local constraints may be imposed to reduce the search space



Bellman's Optimality Principle

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

- We don't need to search the whole space to find the optimal path
- Global and local constraints may be imposed to reduce the search space



Application of TM in Text Matching: The Edit Distance

- The Edit distance
  - It is used for matching written words. Applications:
    - Automatic Editing
    - Text Retrieval

Application of TM in Text Matching: The Edit Distance

- The Edit distance
  - It is used for matching written words. Applications:
    - Automatic Editing
    - Text Retrieval
  - The measure to be adopted for matching, must take into account:
    - Wrongly identified symbols
      e.g. "befuty" instead of "beauty"
    - Insertion errors, e.g. "bearuty"
    - Deletion errors, e.g. "beuty"

Examples:

- **Input: str1 = “geek”, str2 = “gesek”**
- Output: 1
- Explanation: We can convert str1 into str2 by inserting a ‘s’.

- **Input: str1 = “cat”, str2 = “cut”**
- Output: 1
- Explanation: We can convert str1 into str2 by replacing ‘a’ with ‘u’.

- **Input: str1 = “sunday”, str2 = “saturday”**
- Output: 3
- Explanation: Last three and first characters are same. We basically need to convert “un” to “atur”. This can be done using below three operations. Replace ‘n’ with ‘r’, insert t, insert a

The Edit Distance

- Edit distance: **Minimal** total number of changes, C, insertions I and deletions R, required to change pattern A into pattern B,

$$D(A,B) = \min_j [C(j) + I(j) + R(j)]$$

where j runs over All possible variations of symbols, in order to convert $A \longrightarrow B$

The Edit Distance

- Edit distance: **Minimal** total number of changes, *C*, insertions I and deletions R, required to change pattern A into pattern B,

$$D(A,B) = \min_{j}[C(j) + I(j) + R(j)]$$

where j runs over All possible variations of symbols, in order to convert $A \longrightarrow B$

- *Example*: many ways to change **beuty** to **beauty**

The Edit Distance

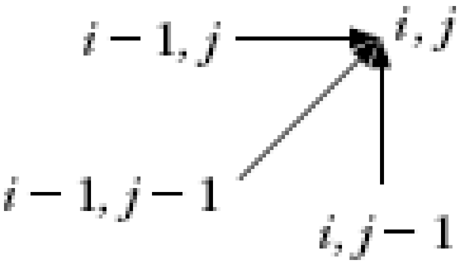
- The optimal path search algorithm can be used, provided we know
  - Initial conditions
  - Search space
  - Allowable transitions
  - Distance measure

The Edit Distance

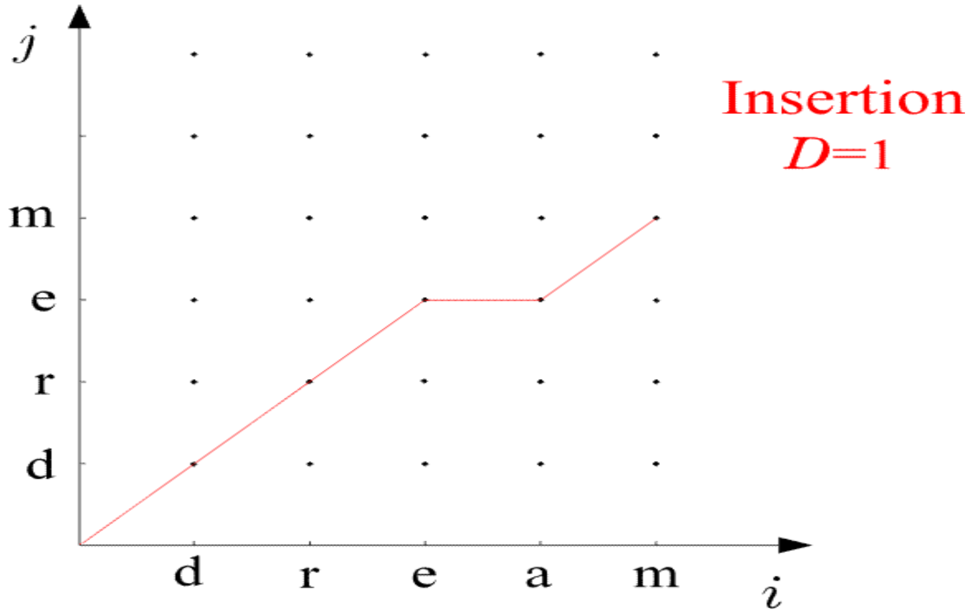
- Cost $D(0,0) = 0$,
- Complete path is searched
- Allowable predecessors and costs
  - $(i-1, j-1) \rightarrow (i, j)$

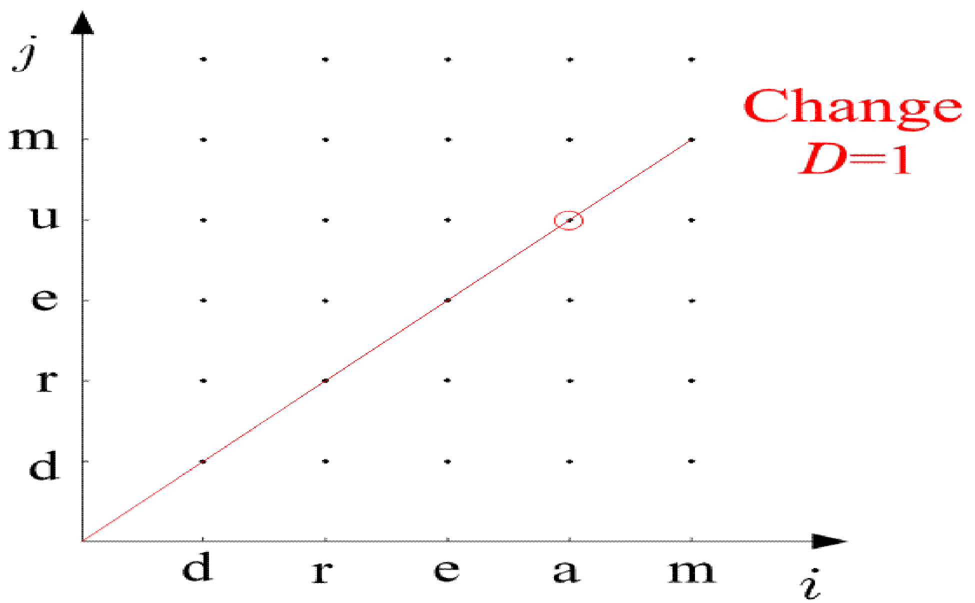
$$d(i, j \mid i-1, j-1) = \begin{cases} 0, & \text{if } t(i) = r(j) \\ 1, & t(i) \neq r(j) \end{cases}$$

  - Horizontal $d(i, j \mid i-1, j) = 1$
  - Vertical $d(i, j \mid i, j-1) = 1$

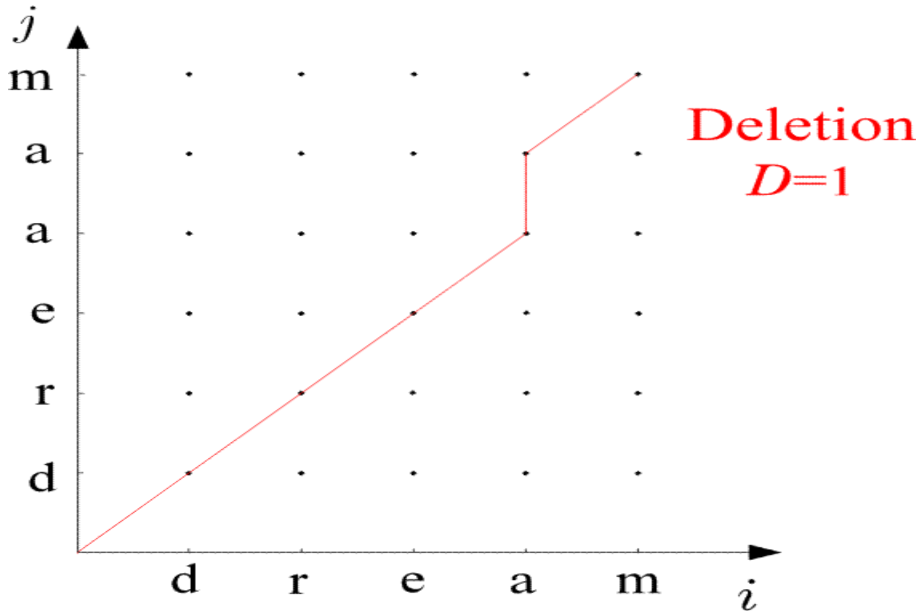


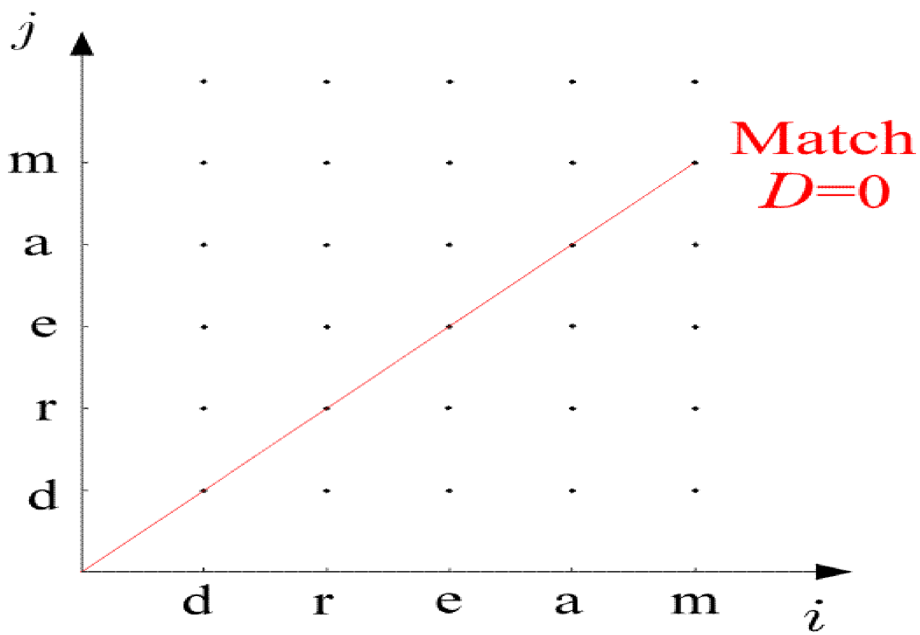
- Examples:





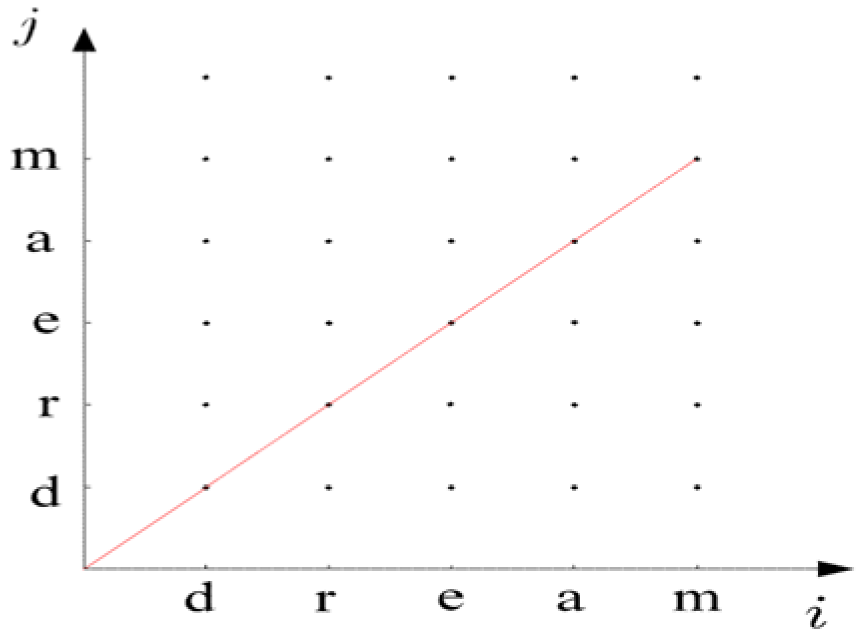
- Examples:





The Edit Distance

- The Algorithm
  - $D(0,0)=0$
  - *For* $i=1$, *to* I
    - $D(i,0)=D(i-1,0)+1$
  - *END {FOR}*
  - *For* $j=1$ *to* J
    - $D(0,j)=D(0,j-1)+1$
  - *END{FOR}*
  - *For* $i=1$ *to* I
    - *For* $j=1$, *to* J
      - $C_1=D(i-1,j-1)+d(i,j \mid i-1,j-1)$
      - $C_2=D(i-1,j)+1$
      - $C_3=D(i,j-1)+1$
      - $D(i,j)=\min(C_1,C_2,C_3)$
    - *END {FOR}*
  - *END {FOR}*
  - $D(A,B)=D(I,J)$



Dynamic Program Table for String Edit

Measure distance between strings `PARK`

`SPAKE`

Edit operations for turning SPAKE into PARK

		P	A	R	K
	delete				
S					
P					
A			insert		
K					substitute
E					

Dynamic Program Table for String Edit

Measure distance between strings PARK

SPAKE

		P	A	R	K
	c_{00}	c_{02}	c_{03}	c_{04}	c_{05}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}
A	c_{30}	c_{31}	???		
K					
E					

Dynamic Program Table for String Edit

		P	A	R	K
	c_{00}	c_{02}	c_{03}	c_{04}	c_{05}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	subst c_{21}	delete c_{22}	c_{23}	c_{24}
A	c_{30}	insert c_{31}	???		
K					
E					

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$
= \min \begin{cases} D(i-1,j-1), \text{ if } si=tj & \textit{//copy} \\ D(i-1,j-1)+1, \text{ if } si!=tj & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}
$$

Dynamic Program Table Initialized

		P	A	R	K
	0	1	2	3	4
S	1				
P	2				
A	3				
K	4				
E	5				

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1			
P	2				
A	3				
K	4				
E	5				

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$
= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}
$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2				
A	3				
K	4				
E	5				

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \text{//substitute} \\ D(i-1,j)+1 & \text{//insert} \\ D(i,j-1)+1 & \text{//delete} \end{cases}$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	1			
A	3				
K	4				
E	5				

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$
= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}
$$

Dynamic Program Table ... filling in

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	1	2	3	4
A	3	2	1	2	3
K	4	3	2	2	2
E	5	4	3	3	3

Final cost of aligning all of both strings.

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$