Lecture Note for Mechanical Engineering Seminar, Tokyo Metropolitan University:

The Dynamic Time Warping Algorithms

Eiji Mizutani
Department of Computer Science
Tsing Hua University, Hsinchu 300 Taiwan

June 30, 2006

Abstract

Dynamic time warping (DTW) has proved to be of great value in diverse contexts of pattern matching (e.g., automated speech recognition). This tutorial note describes a fundamental concept of DTW and its formulation based on the dynamic programming (DP) principle. In particular, DTW is described in the standard DP-framework as an algorithm that solves a multi-stage optimization problem, in which one state (vector) is transformed by a decision (vector) into another at each step.

1 Pattern matching between two sequence patterns

Given two sequence patterns S of length N and T of length M, a straightforward distance measure between S and T, $Dist(S,T)$, can be given with $L \stackrel{\text{def}}{=} \min\{N, M\}$ as the sum of the *absolute distance* below

$$Dist(S,T) \;=\; \sum_{i=1}^{L} \|S_i - T_i\| \;+\; \begin{cases} \sum_{i=L+1}^{N} \|S_i\|, & \text{if } N > M \\ 0, & \text{if } N = M \\ \sum_{i=L+1}^{M} \|T_i\|, & \text{otherwise.} \end{cases} \tag{1}$$

This is a "rigid" measure like the *Hamming distance*, the number of symbols that disagree. In contrast, DTW offers greater flexibility in measuring *similarity* (or *distance*) between a given pair of patterns S and T. (This is similar in spirit to the *edit distance* and the *Levenshtein distance*.) Such a flexible distance measure is often meaningful in real-world applications.

Problem statement:

For an (unknown) N-length test sequence pattern S that consists of N slices (with slice i denoted by S_i), we want to find out the *shortest distance* from S to a (known) template sequence pattern T of length M; here, the shortest distance implies the minimum matching cost between S and T under the following assumptions:

Assumption 1 : The first and last elements/slices of S must match to those of T; that is,

$$\begin{cases} (1)\ S_1 \sim T_1 & \text{with a distance (or matching cost) } d(1,1), \\ (2)\ S_N \sim T_M & \text{with a distance (or matching cost) } d(N,M), \end{cases} \tag{2}$$

where $d(i,j)$ can be defined as the absolute distance [see Equation (1)]

$$d(i,j) = \|S_i - T_j\|. \tag{3}$$

Assumption 2 : Three **actions** are considered so that the sum of $d(i,j)$ in Equation (3) can differ, even when $N = M$, from such a Hamming-distance-like computation that involves *no decision* as $Dist(S,T)$ defined in Equation (1). That is, the current **state** is defined as (i,j), and it is transformed into another state by a **transition rule** that consists of the following three **actions**:

(1) "expansion" action "$\rightarrow$" (go right) from $(i-1,j)$ to (i,j);

(2) "match" action "$\nearrow$ " (go diagonally up) from $(i-1,j-1)$ to (i,j);

(3) "contraction" action " $\uparrow$ " (go straight up) from $(i,j-1)$ to (i,j);

Those three transitions are illustrated in Figure 1, where S is interpreted as a spoken *sentence* while T as some template to be matched.

- Give efficient forward & backward dynamic programming algorithms, and then work on the small example below to verify the DP-formulations.

  Example:

  - Sentence S: "2 4 6 3" (with its length $N = \text{leng}(S) = 4$).
  - Template T: "1 5 4 3" (with its length $M = \text{leng}(T) = 4$).

1.1 Forward DP-Formulation for DTW

We first show the forward DP formulation:

Optimal value function:

$$V(i,j) \;=\; \text{minimum } \textbf{cost-so-far} \text{ (sum of absolute distances so far), arriving at state } (i,j). \tag{4}$$

Recurrence relation ($i,j > 1$; see Figure 1):

$$V(i,j) = \; d(i,j) \; + \; \min \begin{bmatrix} \text{"Expansion" } \rightarrow \text{ (go right):} & V(i-1,j) \\ \text{"Match" } \nearrow \text{ (go diagonally up):} & V(i-1,j-1) \\ \text{"Contraction" } \uparrow \text{ (go straight up):} & V(i,j-1) \end{bmatrix}. \tag{5}$$

Boundary condition:

$$\begin{cases} V(1,1) = d(1,1) \\ V(i,j) = \infty \;\text{ if }\; j \leq 0 \\ V(1,j) = \infty \;\text{ if }\; j > 1. \end{cases} \tag{6}$$

The DTW-approach begins with the boundary condition in Equation (6), and uses the recurrence relation (5) to fill the cost-so-far values in the M-by-N DTW-table, where M and N are the number of slices of T and S, respectively. The answer is given by $V(N,M)$, the cost-so-far value in the upper-right corner of the DTW-table; e.g., see Figure 2(a).

1.2 Backward DP-Formulation for DTW

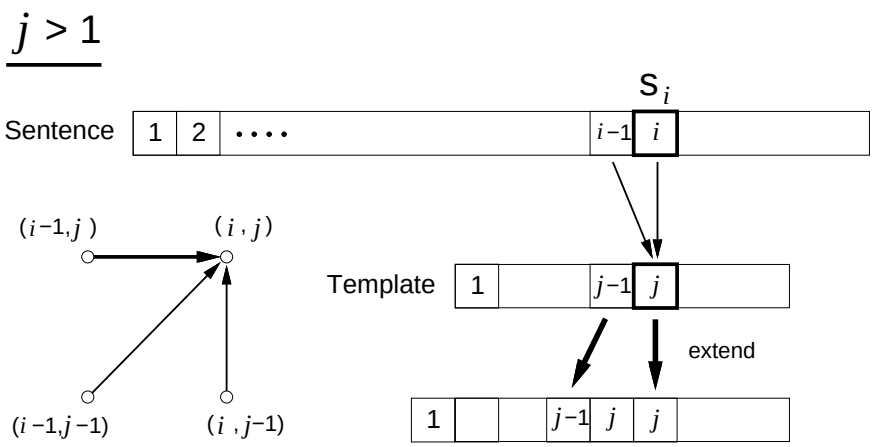
We next show the backward DP formulation:

Optimal value function:

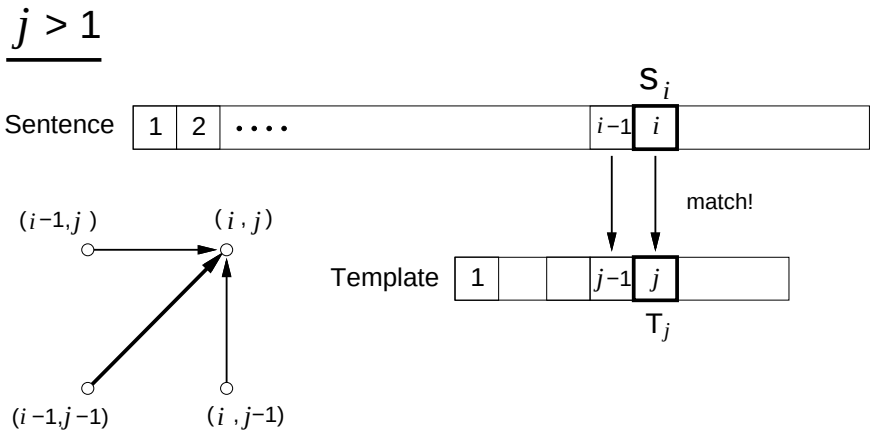
$$U(i,j) \;=\; \text{minimum } \textbf{cost-to-go} \text{ (sum of absolute distances to go), starting at state } (i,j) \text{ to } (N,M). \tag{7}$$

Recurrence relation ($i,j > 1$; see Figure 1):

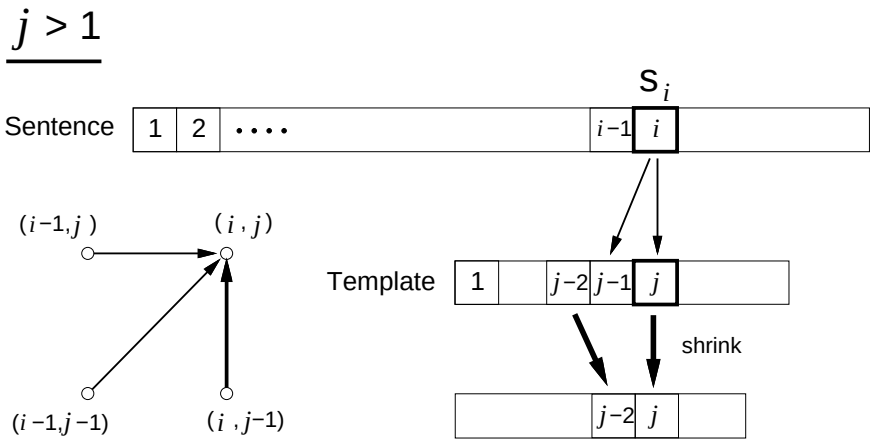
$$U(i,j) = \; d(i,j) \; + \; \min \begin{bmatrix} \rightarrow \text{ (go right):} & U(i+1,j) \\ \nearrow \text{ (go diagonally up):} & U(i+1,j+1) \\ \uparrow \text{ (go straight up):} & U(i,j+1) \end{bmatrix}. \tag{8}$$



(1) "Expansion" action "→ (go right)" from $(i-1, j)$ to (i, j); T is **extending** at slice j.

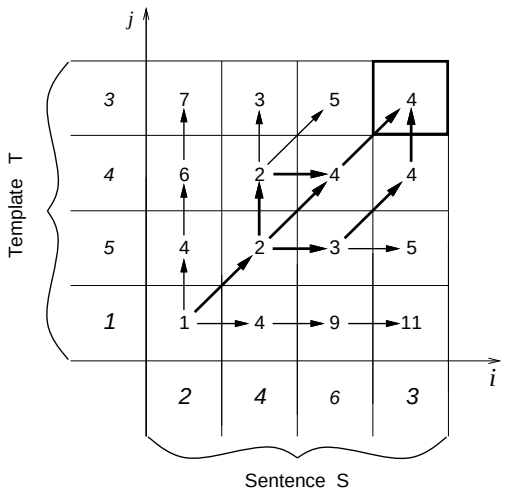


(2) "Match" action "↗ (go diagonally up)" from $(i-1, j-1)$ to (i, j); slice i of S matches slice j of T.

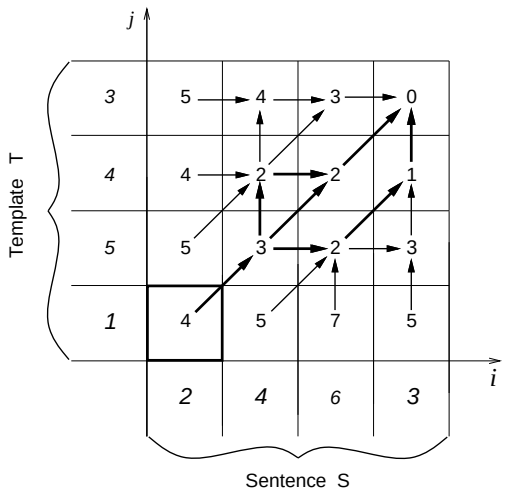


(3) "Contraction" action "↑ (go straight up)" from $(i, j-1)$ to (i, j); T is **shrinking** at slice j.

Figure 1: Dynamic Time Warping (DTW) when $j > 1$ (and $i > 1$): S_i (slice i of S) and T_j (the jth slice of template T) are matched up. Here, three decisions are considered (1) go right; (2) go diagonally up; and (3) go up.



(a) A DTW-table of minimum **cost-so-far** values.



(b) A DTW-table of minimum **cost-to-go** values.

Figure 2: Dynamic Time Warping (DTW) tables obtained by (a) forward DP that yields the answer at (4,4), and (b) backward DP that gives the answer at (1,1).

Boundary condition:

$$\begin{cases} U(N,M) = d(N,M) \\ U(i,j) = \infty \ \text{ if } \ j \leq 0 \\ U(1,j) = \infty \ \text{ if } \ j > 1. \end{cases} \tag{9}$$

The answer is given by $U(1,1)$, the cost-to-go value in the lower-left corner of the DTW-table in Figure 2(b).

Worked example: S = "2 4 6 3" ($N = 4$), and T = "1 5 4 3" ($M = 4$).

The DTW-tables obtained by forward DP in Section 1.1 and backward DP in Section 1.2 are presented in Figure 2. By forward-DP algorithm in Section 1.1, first, the boundary condition in Equation (6) gives

$$V(1,1) = d(1,1) = S_1 - T_1 = 2 - 1 = 1.$$

Then, the recurrence relation in Equation (5) yields the cost-so-far values, leading to the answer, $V(4,4) = 4$, as shown in Figure 2(a).

Similarly, by backward-DP algorithm in Section 1.2, the boundary condition in Equation (9) gives

$$U(4,4) = d(4,4) = S_4 - T_4 = 3 - 3 = 0.$$

The recurrence relation in Equation (8) yields the cost-to-go values, leading to the answer, $U(1,1) = 4$, as shown in Figure 2(b).

In any event, the given sentence S is matched with T at the end with the minimum cost 4. By tracing the arrows, the following optimal three paths, denoted by a sequence of (i,j), connecting (1,1) to (4,4) are obtained:

(1) $(1,1) \nearrow (2,2) \rightarrow (3,2) \nearrow (4,3) \uparrow (4,4)$;

(2) $(1,1) \nearrow (2,2) \nearrow (3,3) \nearrow (4,4)$;

(3) $(1,1) \nearrow (2,2) \uparrow (2,3) \rightarrow (3,3) \nearrow (4,4)$.

The optimal paths must link states (1,1) and (N,M) due to the assumption described in Equation (2).

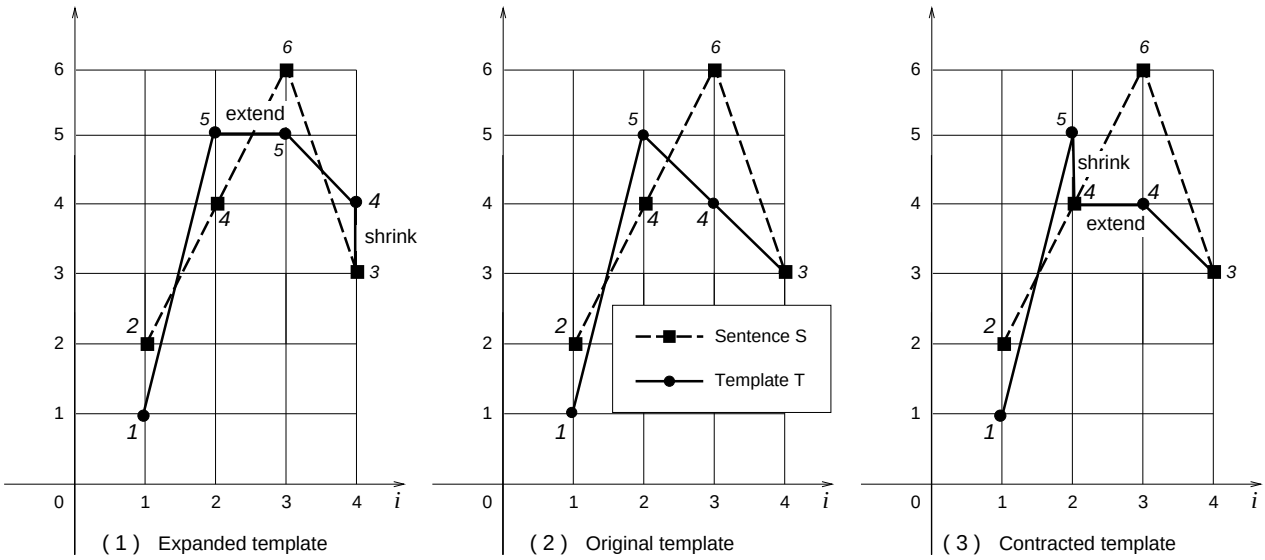


Figure 3: Dynamic Time Warping (DTW) effects: The three figures (1) to (3) above illustrate how Template T extends and shrinks to match Sequence S, associated with the three obtained optimal paths: (1) $(1,1) \nearrow (2,2) \rightarrow (3,2) \nearrow (4,3) \uparrow (4,4)$; (2) $(1,1) \nearrow (2,2) \nearrow (3,3) \nearrow (4,4)$; and (3) $(1,1) \nearrow (2,2) \uparrow (2,3) \rightarrow (3,3) \nearrow (4,4)$. Along path (2), Template T neither expands/extends ($\rightarrow$) nor contracts/shrinks ($\uparrow$). Yet, on path (1), $T_2(=5)$ appears as if it splits into two, extending over slices 2 and 3 ($i = 2, 3$). On path (3), $T_3(=4)$ looks as if it shrinks in toward slice 2 ($i = 2$) and then extends back at slice 3 ($i = 3$). By inspection, we can readily confirm that the matching cost is minimum 4 for all those three paths.

1.3 Dynamic Time Warping Effects — a constrained "measuring worm"-like locomotion

A measuring worm (or inchworm) moves by drawing up the hind end of the body forward and then advancing the front end. Imagine such a caterpillar's locomotion subject to a constraint expressed in Equation (2) with its front and rear legs held fixed. Figures 3(1),(2), and (3) look like three snapshots of such a "poor" inchworm's alternate expansions and contractions with no freedom at both front and hind ends. Here, by analogy with those expanding and contracting behaviors of the "constrained" worm, we attempt to explain why DTW yields a more "flexible" distance measure than straightforward "rigid" Euclidean and Hamming distances. DTW accomplishes template (or pattern) matching in such an *elastic* way that original template T [in Figure 3(2)] attempt to fit a given sequence S by expansion [as in Figure 3(1)] or by contraction [as in Figure 3(3)].

It should be noted that the *contraction* effect is realized by action $\uparrow$ [see Figure 1(3)], but the effect can be drawn by other actions also. For example, instead of action $\uparrow$ from $(i, j-1)$ to (i, j), one may introduce a transitional action from $(i-1, j-2)$ to (i, j) similar to action $\nearrow$. In consequence, we end up having two optimal paths (1) and (3) with the minimum cost 3, to be shown later in Figure 6.

Remarks:

1. For attacking DTW-problems in practice, forward-DP is used (rather than backward-DP) because of the following reasons:
   - real-time (or on-line) processing;
   - **deterministic** nature of decision-makings.
2. For the **stochastic** decision-making optimization problems (e.g., see [3]), backward-DP is the method of choice. A variety of DP examples can be found in [1].
3. In the context of multi-stage neural-network learning, use of the state-dependent *stage cost* alone in the DTW recurrence relation (5) corresponds to **hidden-node teaching** [4]. (Of course, the *min*-operation is infeasible in neural-network learning.)

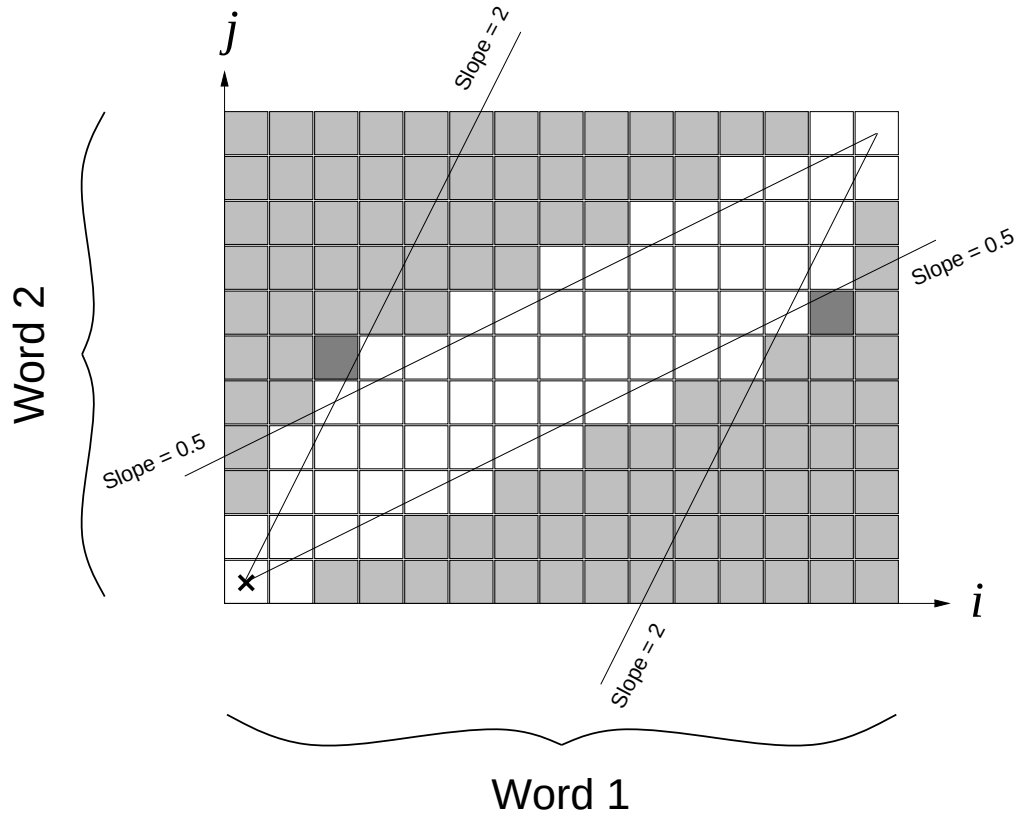


Figure 4: A general form of the Itakura parallelogram for limiting the DTW search space. There is a subtle difference between the above diagram and the original parallelogram (posed in Fig.1, p. 69 in [2]); for instance, Cell(2,1) is allowed above, but it was disallowed in the original diagram because of a modified set of three actions with no directly up "↑" move, as shown in Figure 6(left).

2 Constrained Shortest Path Problems

We now consider two constraints on consecutive moves below into the aforementioned DTW example.

- Constraint (1): No (two) consecutive horizontal moves ("→"-"→") allowed;
- Constraint (2): No (two) consecutive vertical moves ("↑"-"↑") allowed.

Due to Constrain (1), the minimum slope of $\frac{1}{2}$ is obtainable by two consecutive moves: "→"-"↗" (alternatively, "↗"-"→"). Likewise, by Constrain (2), the sequential moves: "↑"-"↗" (alternatively, "↗"-"↑") yield the maximum slope of 2 (see Fig. 2 (a) and (b), page 45 in Sakoe & Chiba 1978 [6]). In this way, the constraints imposed on a certain set of consecutive vertical or horizontal moves can be interpreted as **slope constraints**. Due to the two slope constraints, the *DTW search space is reduced*, and it delineates a **parallelogram**, as illustrated in Figure 4.

2.1 The Itakura 1975 Hybrid Algorithm

For solving the problem with the two slope constraints, Itakura [2] devised a "hybrid" DTW algorithm below:

$$V(i,j) = d(i,j) + \min \begin{bmatrix} \text{action} \rightarrow & : V(i-1,j) \cdot g(i-1,j) \\ \text{action} \nearrow & : V(i-1,j-1) \\ \text{action} \nearrow\!\!\!\nearrow & : V(i-1,j-2) \end{bmatrix} \tag{10}$$

where $g(i-1,j)$ is defined as

$$g(i-1,j) = \begin{cases} \infty, & \text{if the best path to } (i-1,j) \text{ is only from } (i-2,j) \\ 1, & \text{otherwise.} \end{cases} \tag{11}$$

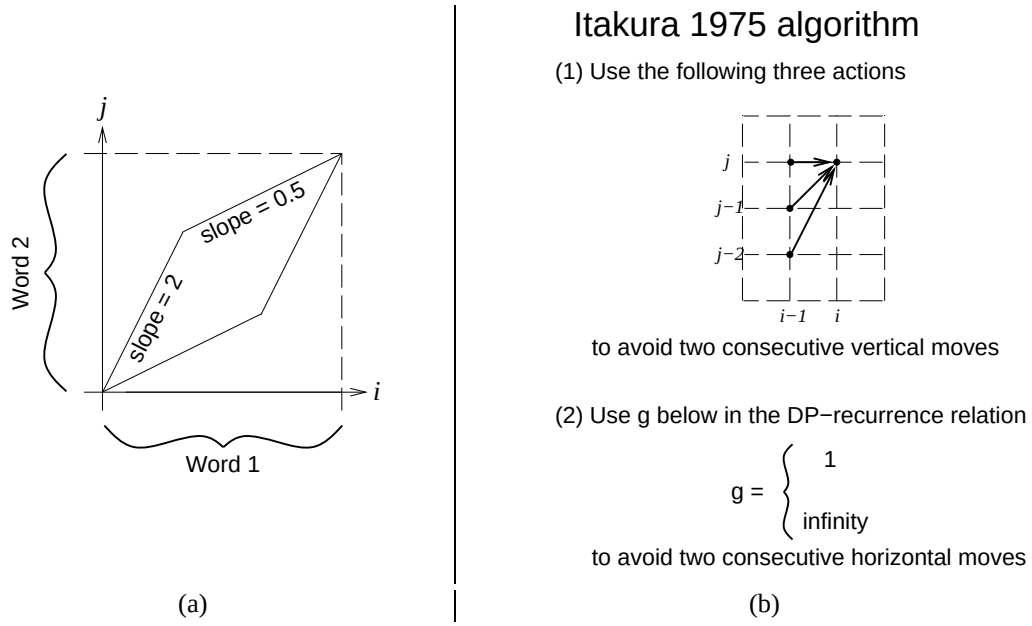


Figure 5: (a) The Itakura parallelogram (see Fig.1, p.69, in [2]) for admissible DTW search space; and (b) two elements of the Itakura 1975 DTW algorithm (see Eqs.(20) & (21), p.69, in [2]) for dealing with two slope constraints.

Two key ideas of this hybrid DTW algorithm are summarized in Figure 5(b); that is, Constraint (1) is treated by introducing an indicator function $g(.,.)$, whereas Constraint (2) is handled by a particular set of three actions [see Figure 5(b)(1) and Figure 6(left)]. In consequence, the algorithm is controlled to search through only a parallelogram space, as depicted in Figure 5(a); the posed parallelogram is slightly different from the one illustrated in Figure 4; read the figure caption therein. In the DTW literature, the resulting search space is often called the "global" Itakura parallelogram, and a role of the indicator function g in Equation (11) is called the Itakura "local" constraint (e.g., see Figs. 8 & 9 on page 12 in [7]). Here, the parallelogram may be *global* in the sense that the DTW table can be initialized as a pre-process to conform to a given admissible region; specifically, value ∞ can be put into all the shaded cells outside the parallelogram in the DTW table of Figure 4 so as to speed up the DTW procedure. To further reduce the search space, one might also introduce a so-called "Sakoe-Chiba band" (see Fig.11, p.13 in [7]), which was introduced by Sakoe & Chiba as an adjustment window (see Fig.1, p.44 in [6]).

To understand algorithmic behaviors of the hybrid method with Equation (10), we first consider Constraints (1) and (2) individually. The following Exercises 2.1 and 2.2 investigate an effect of the specially-designed set of three actions [see Figure 6(left)] that automatically satisfies Constraint (2). Later, Section 2.2 examines how to deal with Constraint (1) alone. After that, Section 2.3 presents two DTW approaches to handle both Constraints (1) and (2).

Exercise 2.1: *A DTW problem in Figure 6.*

> Omit the indicator function $g(.,.)$ from Equation (10) and then employ it to confirm the DTW result in Figure 6(right). In comparison to Figure 2(a), observe the different optimal paths resulting from a new set of three transitional actions in Figure 6(left). □

Exercise 2.2: *A DTW problem in Figure 6.*

> For the two optimal paths in Figure 6(right), illustrate their corresponding "matching" situations just by modifying Figures 3(1) and 3(3). □

2.2 Imposing constraints on successive horizontal moves

In this section, we describe three approaches to deal with Constraint (1), no two consecutive horizontal moves allowed. The first approach is the Itakura algorithm in Equation (10) that uses an indicator function g. The second approach is a popular DTW method based on what we call *stage unfolding*. The third approach is a standard dynamic programming algorithm by state augmentation.

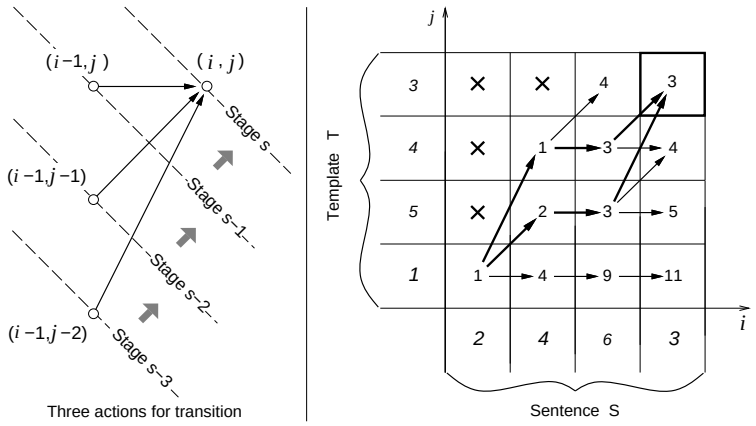
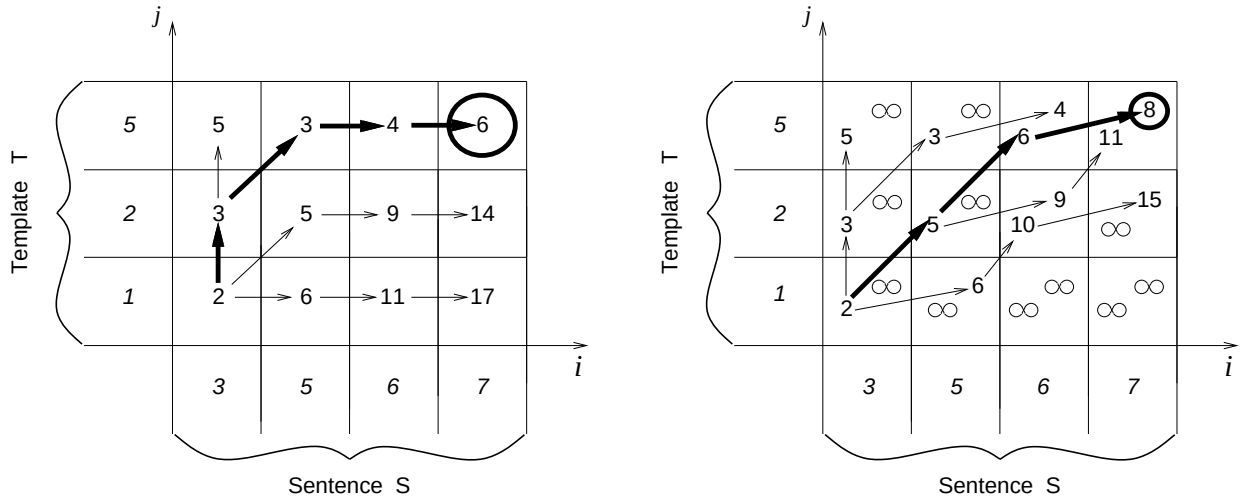


Figure 6: A different set of three transitional actions (left), and its associated DTW-table (of minimum cost-so-far values), where there are two optimal DTW paths with the minimum cost 3. Compared with the results in Figure 2, a newly-introduced action for transition from state $(i-1, j-2)$ at stage $s-3$ to state (i, j) at stage s, where $s \equiv i+j+1$, rules out the simple diagonal route (2) "$(1,1) \nearrow (2,2) \nearrow (3,3) \nearrow (4,4)$" with associated cost 4, yielding a lower cost 3.



(a) No constraint.

(b) Constraint on consecutive horizontal moves.

Figure 7: Dynamic Time Warping (DTW) tables of cost-so-far values.

2.2.1 Use of the indicator function

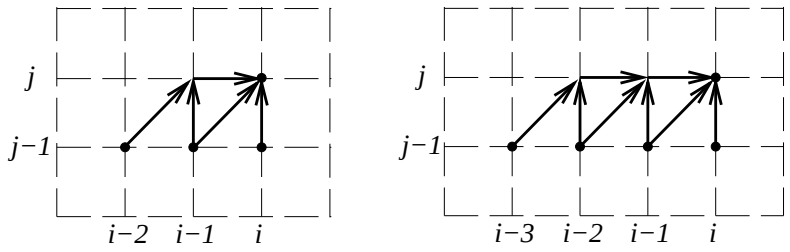
Here, we consider only Constraint (1); for comparison purposes, we shall modify the recurrence relation (10) of the Itakura hybrid algorithm: That is, "action $\nearrow\!\!\!\!/$" that handles Constraint (2) is replaced with an original "action $\uparrow$" leading to the following recurrence relation:

$$V(i,j) = d(i,j) + \min \begin{cases} \text{action } \text{“}\rightarrow\text{”} & : V(i-1,j) \cdot g(i-1,j) \\ \text{action } \text{“}\nearrow\text{”} & : V(i-1,j-1) \\ \text{action } \text{“}\uparrow\text{”} & : V(i,j-1) \end{cases} \tag{12}$$

where $g(i-1,j)$ is defined in Equation (11).

Exercise 2.3: *A DTW problem in Figure 7.*

Apply Equation (12) to two sequences [3,5,6,7] and [1,2,5] with Constraint (1) alone. Note in the unconstrained case that the optimal path consists of two horizontal consecutive moves, as shown in Figure 7(a). □



(a) four admissible routes (b) six admissible routes

Figure 8: Admissible routes under a single constraint posed on (a) two consecutive horizontal moves, and (b) three such moves. In (a), four allowable routes end in (i, j), leading to the DP recurrence equation (13). Obvious generalization applies to (b), where three consecutive horizontal moves are prohibited, yielding six possible paths ending in (i, j).

2.2.2 Stage unfolding

To deal with Constraint (1) alone, one might consider four possible routes depicted in Figure 8(a), leading to the following recurrence relation:

$$V(i,j) = d(i,j) \; + \; \min \left[\begin{array}{l} \text{Action ``}\uparrow\text{''} \\ \quad \text{from} \quad : \; V(i,j-1); \\ \quad (i,j-1) \\ \\ \text{Action ``}\nearrow\text{''} \\ \quad \text{from} \quad : \; V(i-1,j-1); \\ (i-1,j-1) \\ \\ \text{Action ``}\rightarrow\text{''} \\ \quad \text{from} \quad : \\ \quad (i-1,j) \end{array} \right. \left. \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right] , \qquad (13)$$

$$\left\{ \begin{array}{l} \infty, \quad \text{if } V(i-1,j) = \infty; \text{ otherwise, compute below} \\ \\ d(i-1,j) + \min \left[\begin{array}{l} \text{``}\uparrow\text{'': } V(i-1,j-1) \\ \\ \text{``}\nearrow\text{'': } V(i-2,j-1) \end{array} \right]; \end{array} \right.$$

where the four cost-so-far values $V(.,.)$ on the right-hand side are related to $V(i,j)$. This relation corresponds to four paths, Path (1) to (4), in table below:

	Admissible routes depicted in Figure 8	Associated cost-so-far value functions
Path (1)	$(i,j-1) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i,j-1)$
Path (2)	$(i-1,j-1) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i-1,j-1)$
Path (3)	$(i-1,j-1) \rightarrow (i-1,j) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i-1,j-1)$
Path (4)	$(i-2,j-1) \rightarrow (i-1,j) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i-2,j-1)$
Path (5)	$(i-2,j-1) \rightarrow (i-2,j) \rightarrow (i-1,j) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i-2,j-1)$
Path (6)	$(i-3,j-1) \rightarrow (i-2,j) \rightarrow (i-1,j) \rightarrow (i,j)$	$V(i,j) \leftarrow V(i-3,j-1)$

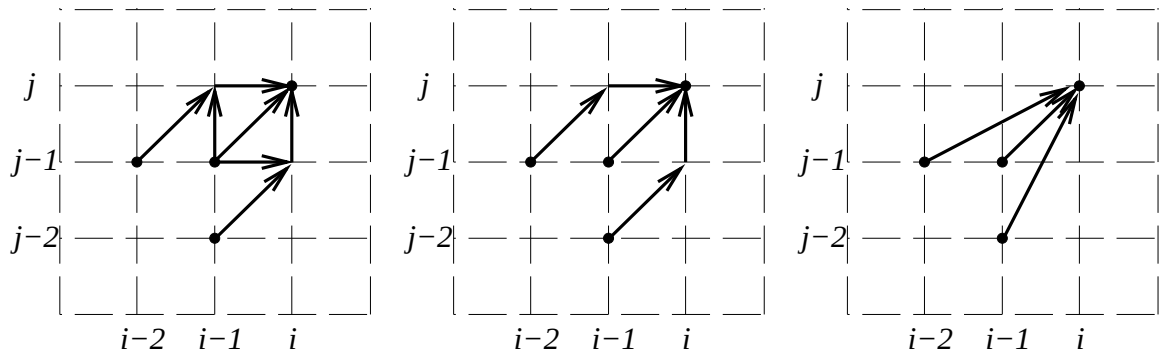
In Equation (13), to manage Constraint (1), one *state-action function* $Q(i,j,\text{“}\rightarrow\text{”})$, called *Q-value* in *reinforcement learning* (see [3] and references therein), is expanded one more stage, relating “$\rightarrow$” to either two concatenated actions: “$\rightarrow$”–“$\uparrow$” or “$\rightarrow$”–“$\nearrow$” so as to avoid “$\rightarrow$”–“$\rightarrow$.” This is what we call **stage unfolding** or **stage fold-back**. (It was referred to as “look-ahead” in [5].)

Exercise 2.4: *A stage fold-back for dealing with Constraint (1), “no two consecutive horizontal moves.”*

Apply Equation (13) to two sequences [3,5,6,7] and [1,2,5] with Constraint (1) alone; see Figure 7(a) for the unconstrained case. □

Exercise 2.5: *No three consecutive horizontal moves allowed.*

When no “*three consecutive horizontal moves*” are allowed [see Figure 8(b)], there are six admissible paths [see Path (1) to (6) in the above table]. Write down the recurrence relation. □



(a) Original paths (b) Simplified paths (c) Short-cut version

Figure 9: Local routes to handle slope constraints (cf. Fig. 2, page 45 in Sakoe & Chiba, 1978 [6]) (a) Original paths; (b) Simplified paths; and (c) Short-cut version. See also [5, 7].

2.2.3 A standard dynamic programming approach by state augmentation

Concatenating actions in the *stage fold-back* (or *unfolding*) concept corresponds to *enlarging the state space appropriately so that the Markov property can hold*. In the standard dynamic programming, the *state augmentation* is often accomplished by introducing extra arguments in the optimal value function (e.g., see [3]).

Exercise 2.6: *A Standard Dynamic Programming by State Augmentation.*

> Give an efficient *standard* dynamic programming algorithm by state augmentation for solving a shortest path problem with Constraint (1) alone. Then apply it to two sequences [3,5,6,7] and [1,2,5] to confirm the result in Figure 7(b). □

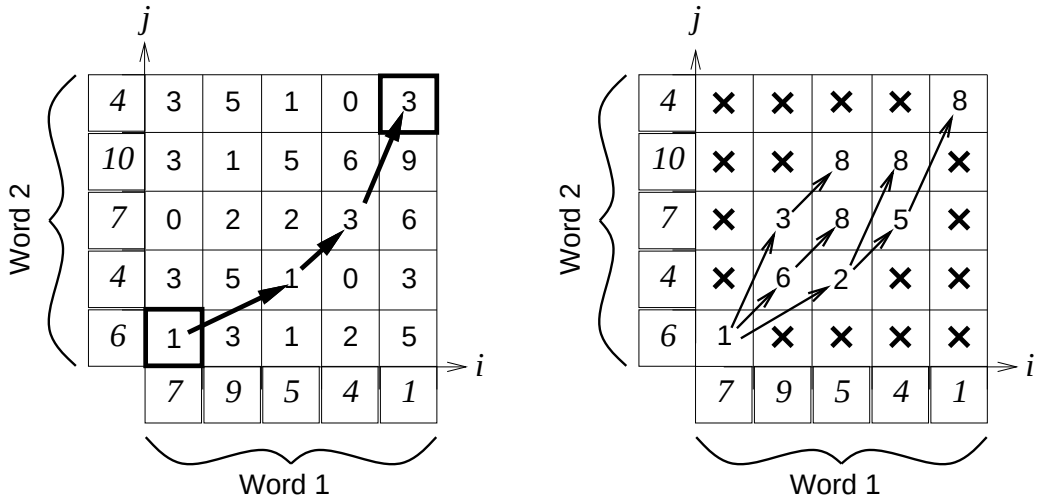
2.3 Two DTW approaches to deal with Constraints (1) and (2)

To handle both Constraints (1) and (2), one could extend the *stage-unfolding* concept introduced in Figure 8(a) to obtain the local admissible paths illustrated in Figure 9(a). Since the up-right (or right-up) move cannot be better than the diagonal move, Figure 9(a) reduces to (b). It should be noted here that Sakoe & Chiba [6] considered **weighting functions** on transitional moves; for instance, a so-called "city-block weighting" assigns a certain weight value to each action in such a way that the diagonal move "↗" may not always be better than the up-right "↑"-"→" and right-up "→"-"↑" moves. In such a case, Figure 9(b) corresponds to a situation where an additional *orthogonal constraint* is imposed on Figure 9(a) so that both up-right and right-up moves are prohibited as well [on top of Constraints (1) & (2)]. In this note, we do not consider any weighting function; so, the additional orthogonal constraint takes no effect.

Figure 9(c), just a short-cut version of (b), designates a new set of three actions that automatically satisfies both Constraints (1) and (2): The idea is essentially the same as using the three actions depicted in Figure 6(left) that automatically satisfy Constraint (2); see Exercises 2.1 and 2.2. We shall investigate how the short-cut version "approximates" the solution: The DP recurrence relation for Figure 9(c) can be written out as

$$V(i,j) = d(i,j) \; + \; \min \left[\begin{array}{l} \text{Action "}\uparrow\text{"} \\ \quad \text{skipping} \quad : \; V(i-1,j-2) \\ \quad (i,j-1) \\ \\ \text{Action "}\nearrow\text{"} \\ \quad \text{from} \quad : \; V(i-1,j-1); \\ (i-1,j-1) \\ \\ \text{Action "}\rightarrow\text{"} \\ \quad \text{skipping} \quad : \; V(i-2,j-1) \\ \quad (i-1,j) \end{array} \right]. \tag{14}$$

This equation yields the result shown in Figure 10. In Figure 10(b), the cost-so-far values are evaluated only in the so-called Itakura parallelogram [see Figures 4 & 5(left)].



(a) (b)

Figure 10: (a) The arrows show the optimal path from State (1,1) to (5,5) under the posed two constraints, Constraints (1) & (2). (b) The obtained table of the optimal cost-so-far values only within the Itakura parallelogram.

Exercise 2.7: *Compare the results obtained by two different sets of three actions in Figures 9(b) and (c).*

(1) Write down the recurrence relation using the local routes (i.e., actions) in Figure 9(b), and then apply it to the same pair of sequences in Figure 10 to find out the optimal path. After that, compare that path with the optimal path shown in Figure 10(a) obtained by Equation (14) [that employs the local actions in Figure 9(c)].

(2) Can we really say that the posed two DTW approaches [Fig.9(b) vs. 9(c)] solve the same problem? □

Exercise 2.8: *A standard dynamic programming by state augmentation (as the third approach).*

Follow Sec. 2.2.3 to give an efficient algorithm that deals with both Constraints (1) and (2) by state augmentation. □

Acknowledgments

I would like to thank Stuart Dreyfus for his "optimal" guidance on dynamic programming over the last decade. Naoyuki Kubota deserves special thanks for giving me opportunities to talk about the subject.

References

[1] Richard E. Bellman and Stuart E. Dreyfus. *Applied dynamic programming*. Princeton University Press, 1962.

[2] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing (ASSP)*, ASSP-23(1):67–72, February 1975.

[3] Eiji Mizutani and Stuart E. Dreyfus. Two stochastic dynamic programming problems by model-free actor-critic recurrent network learning in non-Markovian settings. In *Proc. of the IEEE-INNS International Joint Conf. on Neural Networks (IJCNN'04)*, Budapest, Hungary, July 2004. Available at http://www.ieor.berkeley.edu/People/Faculty/dreyfus-pubs/ijcnn04_1.pdf.

[4] Eiji Mizutani and Stuart E. Dreyfus. On derivation of stagewise second-order backpropagation by invariant imbedding for multi-stage neural-network learning. In *Proceedings of the IEEE-INNS International Joint Conference on Neural Networks (IJCNN'06), part of the World Congress on Computational Intelligence (Wcci'06)*, Vancouver, Canada, July 2006. Available at http://www.ieor.berkeley.edu/People/Faculty/dreyfus-pubs/ESwcci06.pdf.

[5] Cory Myers, Lawrence R. Rabiner, and Aaron E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing (ASSP)*, ASSP-28(6):623–635, December 1980.

[6] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing (ASSP)*, ASSP-26(1):43–49, February 1978.

[7] Harvey F. Silverman and David P. Morgan. The application of dynamic programming to connected speech recognition. *IEEE ASSP Magazine*, pages 7–25, July 1990.