# Synchronization of Batch Trajectories Using Dynamic Time Warping

**Athanassios Kassidas, John F. MacGregor, and Paul A. Taylor**
Dept. of Chemical Engineering, McMaster University, Hamilton, Ontario, L8S 4L7, Canada

*The application of dynamic time warping (DTW) to the analysis and monitoring of batch processes is presented. This dynamic-programming-based technique has been used in the area of speech recognition for the recognition of isolated and connected words. DTW has the ability to synchronize two trajectories by appropriately translating, expanding, and contracting localized segments within both trajectories to achieve a minimum distance between the trajectories. Batch processes often are characterized by unsynchronized trajectories, due to the presence of batch-to-batch disturbances and the existence of physical constraints. To compare these batch histories and apply statistical analysis one needs to reconcile the timing differences among these trajectories. This can be achieved using DTW with only a minimal amount of process knowledge. The combination of DTW and a monitoring method based on Multiway PCA/PLS is used for both off-line and on-line implementation. Data from an industrial polymerization reactor are used to illustrate the implementation and the performance of this method.*

## Introduction

Batch processes play an important role in the production of high added value products, such as specialty polymers, pharmaceuticals, and biochemical materials. Analysis and monitoring of the operation of these processes is crucial to the production of consistent, good quality products. Moreover, products from batch processes are often manufactured in a series of steps; early detection of a bad product at any of these steps will save energy, raw material, and plant capacity. Early detection will also make it easier to assign a cause to the fault and modify the process to eliminate the cause. Furthermore, there may be a chance of compensating for the fault with an appropriate control strategy if the monitoring scheme is implemented on-line.

Product quality measurements in batch processes are obtained infrequently; they are often obtained after the product has been shipped to the customer, or after it has been forwarded to the next processing step. Fortunately, a multitude of process measurements, such as temperatures, pressures, flow rates, are readily available during the process of a batch. In view of this fact, MacGregor and Nomikos (1992) and Nomikos and MacGregor (1994, 1995a,b) proposed a method for monitoring batch processes using these readily measured process variables. Their method is based on multiway principal component analysis (MPCA) and multiway projection to latent structures (MPLS), which are extensions of PCA and PLS to handle three-dimensional matrices. The method essentially builds a statistical model for the deviations of the process variables about their average trajectories using data only from good quality batches. Then, it compares the variation of a new batch about the average trajectory with the MPCA model; any deviation that cannot be statistically attributed to the common process variation indicates that the new batch is different from the good quality batches. When quality measurements are available, one can use MPLS to monitor the progress of the batch and predict its final quality (Nomikos and MacGregor, 1995b).

One strong assumption of the methods proposed by Nomikos and MacGregor is that all batches have equal duration and are synchronized. However, there are many situations in which the total time duration of the batches and/or the duration of various stages within the batches are not the same. Examples include polymerization reactors where there can be batch-to-batch variations in impurities and in the initial charges of the recipe components. Different heat removal capabilities arising from seasonal changes in cooling water temperatures will also influence the rate at which the

Correspondence concerning this article should be addressed to J. F. MacGregor.
Current address of A. Kassidas: Dofasco Inc., Process Automation Technology, DQC-1, Hamilton, ON, L8N 3J5, Canada.

reactions can proceed. Furthermore, in those batch processes, which are not fully automated, some stages are left to the discretion of an operator and quite large variations in the variable trajectories can occur. In all these cases, one has to synchronize the trajectories before the batch histories can be compared and any analysis is performed.

To handle the problem of synchronization, Nomikos and MacGregor (1994) proposed the use of an indicator variable (the approach has been applied by Kourti et al. (1996) to an industrial batch polymerization process). According to their proposal, the trajectories are plotted not with respect to time, but with respect to another variable that must be strictly monotonic, has the same starting and ending values for all batches, and is not noisy. Then, a constant increment is selected and one progresses along the indicator variable. Synchronization is performed by retaining the points in the trajectories that have the same values of the indicator variable. The indicator variable approach assumes that such a variable exists and that process knowledge can be used to determine it. However, there may not exist a single indicator variable that satisfies the above requirements, or if there are several possibilities, it may not be obvious which is the best. The problem of batch trajectories of unequal duration has also been encountered by Lakshminarayanan et al. (1996); their solution is to extend all the trajectories to match the duration of the longest trajectory by simply padding the shorter trajectories with artificial measurements, which are all equal to their last measurement. By doing that, they implicitly assume that all the timing differences between trajectories appear at the last stage of the batch process. This is clearly a strong assumption, which is not true in most cases (including the case study presented in this article).

The presence of unsynchronized trajectories is a common problem in the area of speech recognition and particularly in isolated word recognition (Myers et al., 1980; O'Shaughnessy, 1986; Silverman and Morgan, 1990). The same word can be uttered with different duration and intensity, in different environments, and by different speakers; yet the speech recognition system should be able to classify it correctly. A major part of speech recognition research has concentrated on the type of features to be extracted from speech signals. However, this research is not directly applicable to applications in chemical processes because speech signals are nonstationary high frequency signals and, as such, are quite different from the outputs produced by a chemical process.

Even when the correct features are extracted, the problem of a flexible pattern-matching scheme still remains. Dynamic time warping (DTW) is such a flexible, deterministic, pattern matching scheme which works with pairs of patterns and is able to locally translate, compress, and expand the patterns so that similar features within the patterns are matched. Gollmer and Postens (1995) used this localized nonlinear synchronization capability of DTW to detect the onset of different growth phases or failures in a batch fermentation process. Similarly, DTW could provide an elegant solution to the problem of synchronization of batch trajectories.

In this article, the basic theory of the various DTW algorithms is presented, along with details for their implementation. An iterative method based on DTW is presented for the synchronization of batch trajectories. The method is multivariate in the sense that it does not rely on a single variable to perform the synchronization of the trajectories. The proposed method is then applied to synchronize 31 batch trajectories from an industrial emulsion polymerization process. Finally, it is shown how DTW can be combined with the monitoring scheme of Nomikos and MacGregor (1994, 1995b) for both off-line and on-line batch monitoring.

## Theory of Dynamic Time Warping

The material of this section is a compilation of the theory of DTW that appears in numerous articles in the area of speech recognition. A compact, yet detailed, description is presented in this section with the intention that DTW is to be understood and applied by the unfamiliar reader. Moreover, the application of DTW presented in this article requires several modifications of the standard algorithms, which would be difficult to explain had the DTW theory not been previously described.

### *Introduction*

Let $\boldsymbol{T}$ and $\boldsymbol{R}$ denote the multivariate trajectories of two batches; both are matrices of dimension $t \times N$ and $r \times N$, respectively, where $t$ and $r$ are the number of observations and $N$ is the number of measured variables. As discussed in the introduction, most likely $t$ and $r$ will not be equal. One way to make them equal would be to create artificial points by linear interpolation or extrapolation in one of the two trajectories so that the modified trajectories will contain exactly the same number of points. However, this may not be a good approach since the timing differences between the two batches will probably be local and not global. In such a case, linear global compression or expansion of the time scales of either trajectory will not reconcile their timing differences. Furthermore, with more than two trajectories, it is not obvious how to extrapolate or interpolate periods in a meaningful way. Finally, even if the number of observations is the same for both batches (that is $t = r$), their trajectories may not be synchronized. In either case, if one applies the batch analysis and monitoring scheme of Nomikos and MacGregor (1994) to a set of unsynchronized batch trajectories, unnecessary variation will be included in the statistical model and the resulting statistical tests will not be as sharp in detecting faulty batches (that is, a larger probability for Type II Error).

Thus, a method is required which will synchronize similar characteristics in the two trajectories. DTW is such a method (Itakura, 1975; Sakoe and Chiba, 1978). DTW uses the principle of dynamic programming to minimize a dissimilarity measure (a distance) between the two trajectories. DTW nonlinearly warps the two trajectories in such a way that similar events are aligned and a minimum distance between them is obtained. It will shift some feature vectors in time, compress some and/or expand others so that a minimum distance is achieved (Nadler and Smith, 1993). Readers who are interested only in seeing the application of DTW to synchronize and monitor batch processes and not in the details of the theory and its implementation can proceed directly in the case studies section.

Let $i$ and $j$ denote the time index of the $\boldsymbol{T}$ and $\boldsymbol{R}$ trajectories, respectively. DTW will find a sequence $\boldsymbol{F}^*$ of $K$ points on a $t \times r$ grid

$$F^* = \{c(1), c(2), \ldots, c(k), \ldots, c(K)\}, \quad \max(t, r) \le K \le t + r \tag{1}$$

where

$$c(k) = [i(k), j(k)] \tag{2}$$

and each point $c(k)$ is an ordered pair indicating a position in the grid. For a symmetric DTW algorithm (to be explained in the following paragraphs), this sequence can be viewed as defining a path on the grid that optimally matches each vector in both trajectories so that a normalized total distance between them is minimized. Figure 1 (following O'Shaughnessy, 1986) illustrates the main idea behind DTW for two univariate trajectories $\boldsymbol{T}$ and $\boldsymbol{R}$. By proceeding vector by vector, DTW finds the best vector in $\boldsymbol{R}$ against which to compare each vector in $\boldsymbol{T}$, and vice-versa (O'Shaughnessy, 1986).

As will be explained below, there are many variants of the DTW algorithm. However, all of them can be classified either as symmetric or as asymmetric. In the symmetric versions, the time index $i$ of $\boldsymbol{T}$ and the time index $j$ of $\boldsymbol{R}$ are both mapped onto a common time index $k$, as Eqs. 1 and 2 depict. $\boldsymbol{T}$ and $\boldsymbol{R}$ are considered to be equally important and the optimal path will pass through all the points in both trajectories. If the roles are reversed and their placement in the grid is interchanged (that is, $\boldsymbol{R}$ is placed on the horizontal and $\boldsymbol{T}$ on the vertical axis), a symmetric DTW algorithm will give the same optimal path and the same minimum distance.

On the other hand, an asymmetric DTW algorithm will perform one of the two tasks:

(1) It will map the time index of $\boldsymbol{R}$ on the time index of $\boldsymbol{T}$ or vice-versa, or

(2) It will map both time indices in a common time index, but it will tend to expand or compress more one trajectory relative to the other.

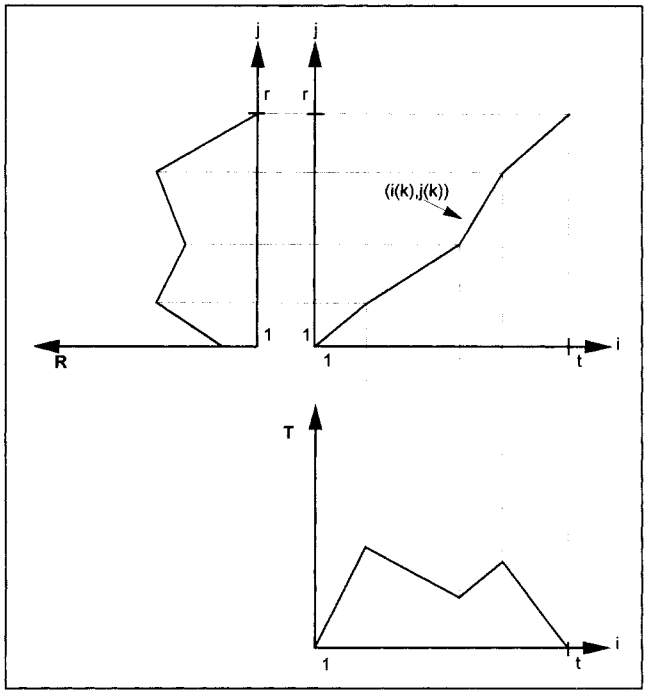


**Figure 1. Example of nonlinear time alignment for two univariate trajectories $\boldsymbol{R}$ and $\boldsymbol{T}$ using DTW.**

For both tasks, the two trajectories are not considered equivalent in an asymmetric DTW algorithm. Hence, if their roles are interchanged, a different optimal path and a different minimum distance will be obtained. The most common asymmetric DTW algorithms map the time index of the trajectory placed on the horizontal axis (that is, the $\boldsymbol{T}$ trajectory in our discussion) onto the time index of the one placed on the vertical axis (that is, the $\boldsymbol{R}$ trajectory). In such a case, the common time index $k$ is in fact the time index $i$ of the trajectory placed on the horizontal axis $\boldsymbol{T}$ and the optimal path contains exactly $t$ points, that is,

$$F^* = \{c(1), c(2), \ldots, c(i), \ldots, c(t)\} \tag{3}$$

and

$$c(i) = (i, j(i)) \tag{4}$$

This implies that the path will go through each vector of $\boldsymbol{T}$, but it may skip vectors of $\boldsymbol{R}$. Nonetheless, both the symmetric and the asymmetric DTW algorithms can be cast in the same framework and a unique solution can be found using the method of dynamic programming.

### *Local and global constraints*

In order to find the best path through the grid of $t \times r$ points, several factors of the DTW algorithm have to be specified. These include: constraints on the endpoints of the path, local continuity constraints that define localized features of the path (that is, slope) and global constraints that define the allowable space for the path.

The most common, and simplest, endpoint constraints require that the two extreme points of both trajectories be matched. That implies that the first $c(1)$ and the last $c(K)$ path points are as follows

$$c(1) = (1, 1) \tag{5a}$$

and

$$c(K) = (t, r) \tag{5b}$$

These constraints are useful when the initial and final points in both trajectories are located with certainty. However, when there is uncertainty about the location of the two extreme points, various endpoint constraints are imposed (Rabiner et al., 1978) which specify an allowable region where the first and last path point may be placed. In this article only the fixed endpoint constraints will be considered; for a detailed description on the implementation of the various relaxed endpoint constraints, see Kassidas (1997).

The local continuity constraints reflect physical considerations (for instance, events should be compared in their natural order in time) and they also guarantee that excessive compression or expansion of the two time scales is avoided (Myers et al., 1980). The first requirement is satisfied by forcing the path to be monotonous of non-negative slope. This can be expressed as

$$i(k+1) \geq i(k) \tag{6a}$$

and

$$j(k+1) \geq j(k) \tag{6b}$$

The second requirement (that is, to avoid excessive compression or expansion of the two time scales) is achieved by not allowing the local slope of the path to exceed a specified range. This is accomplished by specifying a set of allowable predecessors for each point in the grid: if $(i,j)$ is the $k$th path point, then the previous $(k-1)$th path point can only be chosen from a set of specified grid points. Figure 2 illustrates common local continuity constraints and the corresponding slope range that they define.

In Figure 2a the Itakura local constraint is shown (Itakura, 1975). For each $(i, j)$ point in the grid, only three predecessors are allowed: $(i-1,j)$, $(i-1,j-1)$ and $(i-1,j-2)$. Or, in other words, the only way to reach the $(i,j)$ point is either through the $(i-1,j)$ or the $(i-1,j-1)$ or the $(i-1,j-2)$ point. The last local transition (that is, going to the $(i,j)$ point through the $(i-1,j-2)$ point) is characterized by a slope of 2: one horizontal and two vertical steps. Thus, a slope of 2 is the maximum slope allowed. On the other hand, two consecutive horizontal transitions are not allowed, as Figure 2a shows. The local transition from point $(i-1,j)$ to point $(i,j)$ will not be considered at all, if the optimal way to go to point $(i-1,j)$ is through the $(i-2,j)$ point. This means that whenever a horizontal local optimal transition exists (that is, with 0 slope), it has to be followed by a transition that has slope of either one or two. This results in a minimum allowable local slope of 1/2 for the path. Hence, the Itakura local continuity constraint results in a slope range of $[1/2,2]$. Moreover, it is an asymmetric constraint since horizontal local transitions are treated differently from vertical transitions; in fact, vertical transitions are not even considered.

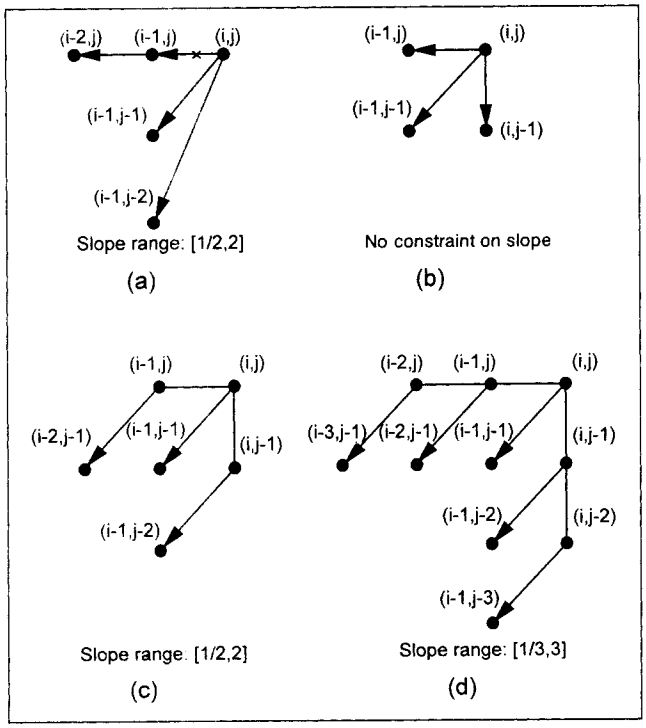


**Figure 2. Typical local continuity constraints.**

(a) Itakura local constraint allowing slopes in [1/2,2]; (b) Sakoe-Chiba local constraint with no constraint on slope; (c) Sakoe-Chiba local constraint allowing slopes in [1/2,2], (d) Sakoe-Chiba local constraint allowing slopes in [1/3,3].

Figures 2b, 2c and 2d illustrate other types of local constraints, the so-called Sakoe-Chiba constraints (Sakoe and Chiba, 1978). All of them restrain the slope of the optimal path by defining a set of allowable predecessors. The local constraint of Figure 2b is an exception to the above statement, because it does not impose any restriction on the slope of the path; the path can follow horizontal or vertical local transitions with no restriction on their length. On the other hand, the local constraints shown in Figures 2c and 2d restrict the slope of the path to $[1/2,2]$ and $[1/3,3]$, respectively. The way to read these local constraints can be illustrated with the following example. Consider the upper local transition of the Figure 2c. It indicates that the only way to reach the $(i,j)$ from the $(i-2,j-1)$ point is through the $(i-1,j)$ point. Moreover, all of them are symmetric since for each $(i,j)$ point the possible predecessors are located in symmetrical local transitions about the diagonal. For a detailed presentation of other local constraints see Myers et al., 1980. Note that in Figure 2, the arrows point towards the allowable predecessor for each $(i,j)$ point to indicate that each point defines its possible predecessors. Thus, the arrows' directions are the opposite of the actual directions of the allowable transitions.

One can extend these local constraints to obtain a particular range of slope. However, this will complicate the dynamic programming-based implementation. An easier way to impose constraints on the slope of the path is to use the local constraint shown in Figure 2b, combined with a check on consecutive horizontal and/or vertical optimal local transitions. This modification will result in a symmetric constraint with the desired slope range. Thus, if $m$ is the maximum number of allowable consecutive horizontal or vertical local transitions, the slope of path will be restricted between $1/m$ and $m$; that is, a slope range of $[1/m, m]$.

If the local constraints define a set of predecessors for each $(i,j)$ point, the global constraints define a subset of the $t \times r$ grid to be the actual search space for the optimal path. Most of the global constraints need not be explicitly imposed. This is due to the fact that the implementation of most of the local continuity constraints automatically implies the global constraints. For example, assume that the local constraints of Figures 2a or 2c are used, in conjunction with the fixed-endpoint constraints of Eqs. 5. Then, the actual search space will be the area included by the lines of slope 1/2 and 2, emanating from the first (1, 1) and the last $(t,r)$ path point. This is illustrated in the Figure 3a: the search area is the shaded parallelogram (Itakura, 1975). In the case that the number of observations in $\boldsymbol{T}$ is twice (or half) of those in $\boldsymbol{R}$, the allowed search space is reduced to the diagonal line (Silverman and Morgan, 1990).

Figure 3b shows the band global constraint. This constraint does not allow the path to deviate $\pm M$ grid points from the linear path starting at point (1,1) (Sakoe and Chiba, 1978). For a feasible search space to exist, $M$ has to be at least equal to or greater than the absolute value of the difference between the number of observations in $\boldsymbol{R}$ and $\boldsymbol{T}$, that is,

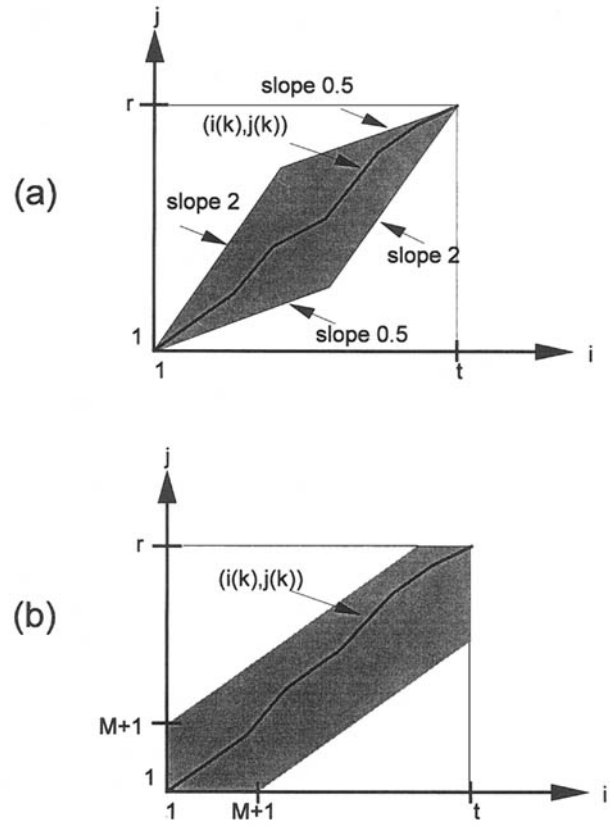


**Figure 3. Typical global constraints.**

(a) Itakura global constraint; (b) Sakoe-Chiba band constraint.

$$M \geq |t - r| \tag{7}$$

This global constraint is usually used in conjunction with the local constraint of Figure 2b, that is, when no restriction is imposed on the slope of the path. The combination of the two constraints will prevent large deviations from the linear path, although this may be an indication of the dissimilarity between the two trajectories. When the band constraint is present, this dissimilarity will appear as an inflated total distance. Moreover, it is possible to combine different local and global constraints. For example, one can use the local constraint of Figure 2c, together with the band global constraint. In such a case, the search space will be the intersection of the two shaded regions of Figures 3a and 3b.

### *Formulation of the minimum distance problem*

As mentioned above, the objective of DTW is to find the best path through a grid of vector-to-vector distances such that some total distance measured between the two trajectories is minimized. A general form of this distance measured for any path (Sakoe and Chiba, 1978; Myers et al., 1980) is

$$D(t,r) = \frac{\sum_{k=1}^{K} d[i(k),j(k)] \cdot w(k)}{N(w)} \tag{8}$$

where $D(t,r)$ is a normalized total distance between the two trajectories; $d[i(k),j(k)]$ is the weighted local distance between the $i(k)$ vector of the $\boldsymbol{T}$ trajectory and the $j(k)$ vector of the $\boldsymbol{R}$ trajectory. The most commonly used local distance is the weighted quadratic distance.

$$d(i(k),j(k)) = \{\boldsymbol{T}[i(k),:] - \boldsymbol{R}[j(k),:]\} \cdot \boldsymbol{W} \cdot \{\boldsymbol{T}[i(k),:] - \boldsymbol{R}[j(k),:]\}^T \tag{9}$$

where $\boldsymbol{W}$ is a positive definite weight matrix that reflects the relative importance of each measured variable; $w(k)$ is a nonnegative weighting function for the $d[i(k),j(k)]$ local distance; and $N(w)$ is a normalization factor which is a function of the weighting function $w(k)$.

Therefore, $\boldsymbol{D}(t,r)$ is the sum of all the local distances between pairs of observations in the two trajectories that lie along the path, weighted by $w(k)$, and divided by the normalization factor $N(w)$.

Thus, the optimal path is found as the solution to the following optimization problem

$$\boldsymbol{D}^*(t,r) = \min_{F} [\boldsymbol{D}(t,r)] \tag{10}$$

and

$$\boldsymbol{F}^* = \operatorname*{argmin}_{F} [\boldsymbol{D}(t,r)] \tag{11}$$

where $\boldsymbol{D}^*(t,r)$ is the minimum normalized total distance between $\boldsymbol{T}$ and $\boldsymbol{R}$, and $\boldsymbol{F}^*$ is the optimal path.

The $N(w)$ parameter is a scalar and serves as normalization factor for the distance estimation. Its value will depend on the type of the weighting function $w(k)$ that is used. Its purpose is to make the normalized total distance independent of the number of path points $K$ and the lengths of the two trajectories.

The weighting function $w(k)$ depends on the local continuity constraints and serves two purposes. The first is to provide more flexibility in the DTW algorithm by weighting the local distance $d[i(k),j(k)]$, depending on the local transition by which the $[i(k),j(k)]$ path point can be reached from the $[i(k-1),j(k-1)]$ previous path point. As Figure 2 shows, for any $(i,j)$ point in the grid, a set of allowable local transitions is defined by which the $(i,j)$ point can be reached; $w(k)$ allows some local transitions to be treated preferentially (by assigning small weights to them) over some others. The second purpose of $w(k)$ is to make the normalized total distance independent of the number of the path points by imposing an appropriate value for the normalization factor $N(w)$.

The importance of the last point can be seen in Eqs. 8 and 10. The optimization problem of Eq. 10 uses a rational function as a criterion. In principle, it is possible to solve such optimization problems. However, dynamic programming cannot be used for this problem, since the global solution in dynamic programming is obtained recursively by a series of local solutions that do not consider the best global path at all. Dynamic programming retrieves the optimal path at the end, assuming that the optimal total distance has been found. Thus, problems like the one of Eq. 10, where the minimization depends simultaneously on both the total distance and the path, cannot be solved by dynamic programming. On the

other hand, if the normalization factor $N(w)$ is independent of the optimal path, the optimization problem reduces to

$$D^*(t,r) = \frac{1}{N(w)} \min_F \left\{ \sum_{k=1}^{K} d[i(k), j(k)] \cdot w(k) \right\} \quad (12)$$

and this problem lends itself to a dynamic programming-based solution (Myers et al., 1980).

Many different weighting functions have been proposed in the literature of DTW (Itakura, 1975; Sakoe and Chiba, 1978; Myers et al., 1980; Ney, 1984). The two most common ones are

$$\text{symmetric: } w(k) = [i(k) - i(k-1)] + [j(k) - j(k-1)], \quad i(0) = j(0) = 0 \quad (13a)$$

$$\text{asymmetric: } w(k) = i(k) - i(k-1), \quad i(0) = 0 \quad (13b)$$

The weighting function of Eq. 13a weights a local transition from the $(k-1)$th path point to the $k$th path point, according to the number of horizontal and vertical steps that need to be taken for that particular local transition. Both horizontal and vertical steps are considered equivalent. Thus, it is a symmetric weighting function. On the other hand, Eq. 13b considers only the number of horizontal steps required for a local transition and, for that reason, it is an asymmetric weighting function. Since only the horizontal steps are considered, this weighting function favors transitions for which $i(k) = i(k-1)$, that is, vertical transitions. Figure 4 illustrates these weighting functions, when the local continuity constraint of Figure 2b is applied. The coefficients for each local transition are the result of the weighting functions of Eqs. 13a and 13b. Similar coefficients are obtained when different local continuity constraints are applied [see Sakoe and Chiba (1978), and Myers et al. (1980) for more on various weighting functions].

The normalization factor $N(w)$ can now be defined. The normalized total distance, as defined in Eq. 8, is an average distance between the two trajectories along any path. As such, it is reasonable to set $N(w)$ equal to the number of the local distances computed along the path

$$N(w) = \sum_{k=1}^{K} w(k) \quad (14)$$

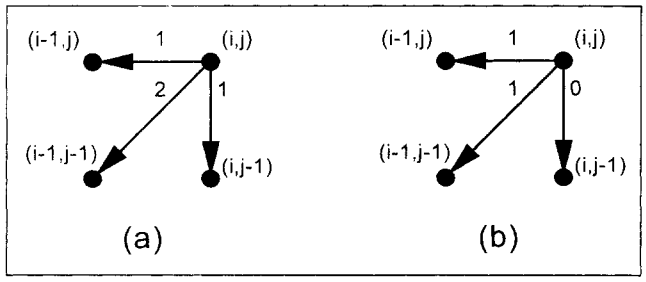


**Figure 4. Local continuity constraint with no constraint on slope.**

(a) Symmetric weighting function; (b) asymmetric weighting function.

Hence, if the weighting functions of Eqs. 13a and 13b are used, the corresponding normalization factors are

$$N(w) = \sum_{k=1}^{K} \{[i(k) - i(k-1)] + [j(k) - j(k-1)]\} \quad (15a)$$

$$= i(K) - i(0) + j(K) - j(0) = t + r$$

$$N(w) = \sum_{k=1}^{K} [i(k) - i(k-1)] = i(K) - i(0) = t \quad (15b)$$

and they are both independent of the optimal path.

For the application of DTW presented in this article, the $N(w)$ factor plays no role and it can be omitted. However, in speech recognition applications, a new word is compared with several reference words, which may contain a different number of observations, and the classification is done on the basis of minimum distance. If a non-normalized distance is used, the decision will be biased towards the referenced word with the minimum number of observations. The normalization factor is therefore used to prevent this undesirable characteristic of DTW.

It was mentioned that there are symmetric and asymmetric DTW algorithms. A symmetric DTW algorithm will result if a symmetric local continuity constraint is used together with a symmetric weighting function. Conversely, if either an asymmetric local constraint (such as the Itakura local constraint) and/or an asymmetric weighting function are used then the resulting DTW algorithm will be asymmetric.

### *Solution via dynamic programming*

The solution of the optimization problem shown in Eq. 10 is based on the Principle of Optimality, which states that "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman and Dreyfus, 1962; Bertsekas, 1987). For this problem, the Principle of Optimality is translated into the following two rules (Myers et al., 1980; Ney, 1984; Silverman and Morgan, 1990):

*Rule (I):* Let $F^*$ be the optimal global path on the $t \times r$ grid. If $F^*$ goes through an $(i,j)$ point, then the optimal path to the $(i,j)$ point is part of $F^*$.

*Rule (II):* The optimal path to the $(i,j)$ point depends only on previous grid points.

The above rules, used in any variant of DTW, define a recursive dynamic programming relationship. This recursive relationship depends on the type of local continuity constraint and on the weighting function. It will be described by means of an example.

Assume that the fixed-endpoint constraints of Eqs. 5 are used, together with the symmetric local continuity constraint and the weighting function of Figure 4a. Also, assume that the band constraint of Figure 3b is used. Let $D_A(i,j)$ be the minimum accumulated distance from point $(1,1)$ to point $(i,j)$, that is

$$D_A(i,j) = \min_{F'} \sum_{k=1}^{K'} d[i(k), j(k)] \cdot w(k) \quad (16)$$

where $F'$ is any path, $F'^*$ is the optimal path to the $(i,j)$ point and $K'$ is the number of path points. Thus, Eq. 12 becomes

$$D^*(t,r) = \frac{1}{t+r} D_A(t,r) \tag{17}$$

(as mentioned, $N(w) = t + r$ for this type of symmetric weighting function, Eq. 15a).

The assumed local continuity constraint implies that the $(i,j)$ point can only be reached by either the $(i-1,j)$, or the $(i-1,j-1)$ or the $(i,j-1)$ point. However, for any of these three possible predecessor points, there is a minimum accumulated distance. Due to Rule (I), if the $(i,j)$ point lies on the optimal path, then the transition from the three possible predecessors has to be optimal. Also, due to Rule (II), this optimal transition will not be affected by any subsequent decision. Thus, according to the Rules (I) and (II), the chosen local continuity constraint and the symmetric weighting function, $D_A(i,j)$ will be found by solving the following simple optimization problem

$$D_A(i,j) = \min \begin{Bmatrix} D_A(i-1,j) + d(i,j) \\ D_A(i-1,j-1) + 2 \cdot d(i,j) \\ D_A(i,j-1) + d(i,j) \end{Bmatrix} \tag{18}$$

Now, because at this point it is not known whether the $(i,j)$ point lies on the optimal path, we have to store our decision as to which of the three alternatives in Eq. 18 was selected. This procedure (that is, Eq. 18 and storage of the optimal local transition) has to be done for all the $(i,j)$ points that lie in the allowable search area, that is, the shaded area of Figure 3b. Note however, that if the optimal path does not need to be reconstructed, these optimal local transitions do not have to be stored.

Thus, one would start from the point $(1,1)$ as

$$D_A(1,1) = 2 \cdot d(1,1) \tag{19}$$

(the weight of 2 is according to the assumed weighting function of Eq. 13a) and would proceed recursively via two iterations, one nested in the other, until the $(t,r)$ grid point is reached. This constitutes the forward phase of the dynamic programming recursion. The outer iteration will progress on the time index $i$ of the trajectory placed on the horizontal axis, whereas the inner iteration will progress on the allowable range of the time index $j$ of the trajectory placed on the vertical axis. As Figure 3b shows, for any value of the horizontal time index, there is an allowable range (shaded region) for the vertical time index. Thus, the index of the outer loop $i$ goes from 1 to $t$, while the range of the inner loop index $j$ depends on the value of the outer iteration index. $M$ (the maximum deviation from the diagonal path) defines the extent of the search area for the optimal path; it must be $M \geq |t-r|$ so that the band of width $2 \cdot M$ includes the $(t,r)$ grid point and the two constraints (that is, the band constraint and the fixed endpoint constraints) are compatible. In general, $M$ should reflect the uncertainty in locating the first and the last points of the trajectories.

The iterative procedure of Eq. 18 finishes when the $D_A(t,r)$ distance is computed and, subsequently, the minimum normalized total distance $D^*(t,r)$ is computed via Eq. 17. To reconstruct the optimal path, one has to proceed in a backward manner, starting from the $(t,\ r)$ point and using the stored information on the optimal decisions at the allowable $(i,j)$ grid points. Thus, first the predecessor of the $(t,r)$ point is located, then the predecessor of the latter is located and this is repeated until the $(1,1)$ point is reached.

In terms of memory requirements, these are not large if only the minimum distance is sought. In that case, only two vectors of accumulated distances have to be stored. At any outer iteration $i$ a vector that stores the $D_A(i-1,j)$ distances is required and the $D_A(i,j)$ vector of distances is computed via Eq. 18. At the next iteration, $i+1$, the $D_A(i-1,j)$ distances are not required anymore and their memory space can be used to store the $D_A(i,j)$ distances. The memory space for the latter can be used to store the new $D_A(i+1,j)$ distances and the whole storage-updating procedure is repeated.

If the optimal path is also sought, then for any $(i,j)$ point in the allowable search area in the grid, an integer index (from a set of three indices, each associated with a possible predecessor) has to be stored indicating the optimal predecessor. This is done because any of the points in the allowable space can be a point of the optimal path. Thus, for our example, this information has to be stored for all points that lie in the shaded region of Figure 3b.

The above procedure, with minor alterations depending on the type of endpoint, local continuity and global constraints and weighting function, is applied in any DTW algorithm (for more details, see Kassidas, 1997). For example, when the Itakura local continuity constraints are used, the recursive relationship is

$$D_A(i,j) = \min \begin{Bmatrix} D_A(i-1,j) + d(i,j) \text{ or } \infty \text{ if predecessor of } (i-1,j) \text{ is } (i-2,j) \\ D_A(i-1,j-1) + d(i,j) \\ D_A(i-1,j-2) + d(i,j) \end{Bmatrix} \tag{20}$$

with

$$D_A(1,1) = d(1,1) \tag{21}$$

The DTW algorithm of Eq. 18 is a symmetric one; both trajectories are equivalent and if their placement in the grid in reversed, the same optimal path and optimal distance will be obtained. The number of path points $K$ will be greater than either $t$ or $r$ ($K \geq \max(t,r)$). Both $R$ and $T$ are locally expanded and/or translated so that common features are synchronized and their timing differences are reconciled. In contrast, the Itakura constraint of Eq. 20 results in an asym-

metric DTW algorithm. The optimal path contains exactly $t$ points, as many as the trajectory placed on the horizontal axis (that is, $\boldsymbol{T}$) contains. $\boldsymbol{R}$ is locally translated, contracted and expanded, so that its timing differences with $\boldsymbol{T}$ are reconciled; the latter remains intact. These characteristics of symmetric and asymmetric DTW algorithms must be taken into consideration when DTW is applied for the synchronization of batch trajectories; this is discussed in the next section.

## Synchronization of Batch Trajectories Using DTW

### *Dynamic time warping algorithm*

Let $\boldsymbol{B}_i$, $i = 1, \ldots, \mathrm{I}$ be a set of I trajectories of good quality batches. Each $\boldsymbol{B}_i$ is a matrix of $b_i \times N$ where $b_i$ is the number of observations and $N$ is the number of measured variables. Also assume that some appropriate scaling for the variables has been applied. Finally, assume that a reference batch trajectory, $\boldsymbol{B}_{\mathrm{RFF}}$ has been somehow defined; this is a matrix of $b_{\mathrm{REF}} \times N$. The issues of scaling and choice of $\boldsymbol{B}_{\mathrm{RFF}}$ will be discussed in the following subsection. Now the objective is to synchronize each $\boldsymbol{B}_i$ with $\boldsymbol{B}_{\mathrm{RFF}}$.

As discussed in the previous section, DTW works with pairs of patterns. Thus, one needs to separately synchronize each $\boldsymbol{B}_i$ with $\boldsymbol{B}_{\mathrm{RFF}}$. The main question is what kind of DTW algorithm should be used; specifically, whether it should be a symmetric or an asymmetric algorithm. As mentioned, after DTW is performed using a symmetric DTW algorithm, the synchronized trajectories have equal duration, which is greater than the duration of the trajectories before synchronization. This common duration is determined by the DTW algorithm and cannot be specified *a priori*. Furthermore, it will be different for each $\boldsymbol{B}_i$ that is synchronized with $\boldsymbol{B}_{\mathrm{RFF}}$. Therefore, if a symmetric DTW is used to synchronize each $\boldsymbol{B}_i$ with $\boldsymbol{B}_{\mathrm{RFF}}$, the result will be a set of expanded trajectories with unequal duration; each $\boldsymbol{B}_i$ will be individually synchronized with $\boldsymbol{B}_{\mathrm{RFF}}$ but not with each other. Therefore, the choice of a symmetric algorithm would still result in the situation of having a set of batch trajectories with unequal duration.

On the other hand, the most common asymmetric DTW algorithms treat one trajectory preferentially. The optimal path goes through all points in one of them (which can be viewed as the defining trajectory) and can skip points of the other. After DTW is performed, the synchronized trajectories have equal duration, equal to the duration of the defining trajectory. For the current problem, one would use $\boldsymbol{B}_{\mathrm{RFF}}$ as the defining trajectory and map its time axis onto the time axis of each $\boldsymbol{B}_i$. The end result will be a set of synchronized trajectories with equal duration $b_{\mathrm{REF}}$ all of them synchronized with $\boldsymbol{B}_{\mathrm{RFF}}$ and synchronized with each other.

Although this appears to be a reasonable solution, it has the disadvantage that the synchronized trajectories may not contain all the data points of the original trajectories because the optimal path may have skipped selected points in them. This is an undesirable side effect because features that appear in some $\boldsymbol{B}_i$ and do not appear in $\boldsymbol{B}_{\mathrm{RFF}}$ may be left out. In effect, a subtle filtering is performed that removes inconsistent features. If an MPCA/MPLS model is constructed from the "filtered" trajectories, it will be biased towards false alarms since it will not consider inconsistent features that may be present in a new batch trajectory.

In summary, symmetric DTW algorithms include all points in the original trajectories and result in expanded trajectories of various lengths. Asymmetric DTW algorithms may eliminate points, but will produce synchronized trajectories of equal length. The following method (symmetric DTW algorithm combined with an asymmetric synchronization procedure) proposes to achieve a compromise between the two extremes.

*Step A: Symmetric DTW Algorithm*

For each $\boldsymbol{B}_i$, apply DTW between $\boldsymbol{B}_i$ and $\boldsymbol{B}_{\mathrm{RFF}}$ using the following constraints

(i) fixed-endpoint constraints
(ii) band global constraint
(iii) local constraint

$$D_A(i,j) = \min\begin{Bmatrix} D_A(i-1,j) + d(i,j) \\ D_A(i-1,j-1) + d(i,j) \\ D_A(i,j-1) + d(i,j) \end{Bmatrix}, \quad D_A(1,1) = d(1,1) \tag{22}$$

At the end, reconstruct the optimal path.

*Step B: Asymmetric Synchronization*

When more than one point of $\boldsymbol{B}_i$ is aligned with one point of $\boldsymbol{B}_{\mathrm{RFF}}$ do the following:

(i) Take the average of these points of $\boldsymbol{B}_i$.
(ii) Align this average point with the particular point of $\boldsymbol{B}_{\mathrm{RFF}}$.

After synchronization, $\boldsymbol{B}_i$ contains as many data points as $\boldsymbol{B}_{\mathrm{RFF}}$, that is, $b_{\mathrm{REF}}$.

*Asymmetric Synchronization (Step B)* can be best illustrated by means of an example. Assume that $\boldsymbol{B}_i$ is placed on the horizontal and $\boldsymbol{B}_{\mathrm{RFF}}$ on the vertical axis. This arrangement does not affect the DTW algorithm in *Step A* since it is symmetric. Also assume that after DTW, the following three points are included in the optimal path: $(i-1,j)$, $(i,j)$ and $(i+1,j)$. According to them, the $(i-1)$th, $i$th and $(i+1)$th points of $\boldsymbol{B}_i$ are all aligned with the $j$th point of $\boldsymbol{B}_{\mathrm{RFF}}$. The proposed method takes the average of the three

$$\frac{\boldsymbol{B}_i(i-1,:) + \boldsymbol{B}_i(i,:) + \boldsymbol{B}_i(i+1,:)}{3}$$

and synchronizes this average with $\boldsymbol{B}_{\mathrm{RFF}}(j,:)$.

The proposed DTW algorithm is still a symmetric algorithm and as such the optimal path passes through all the points in both patterns. All points of $\boldsymbol{B}_i$ (even if some of them have been averaged) are included in the synchronized trajectory. On the other hand, the local continuity constraint in Eq. 22 favors diagonal over horizontal or vertical local transitions. The local constraint is a modification of the one shown in Eq. 18. The local constraint in Eq. 18 gives a weight of 2 to the local distance $d(i,j)$ for a diagonal local transition [from $(i-1,j-1)$ to $(i,j)$ point]. This weight was the result of a symmetric weighting function; its purpose was to provide independence of the final distance to the number of points in the optimal path. However, in this problem, only the optimal path is of interest and not the final distance found by DTW. Using the smaller weight of 1 (as in Eq. 22), diagonal local transitions are preferred over horizontal or vertical ones, and

it is the horizontal and vertical transitions that distort the time axes of $\boldsymbol{B}_i$ and $\boldsymbol{B}_{\mathrm{RFF}}$. Thus, the constraint of Eq. 22 results in smaller distortions of the time axes of both $\boldsymbol{B}_i$ and $\boldsymbol{B}_{\mathrm{RFF}}$ and consequently requires less averaging in Step B.

Although the DTW algorithm of Step A is a symmetric algorithm, the algorithm in Step B is an asymmetric operation that synchronizes all $\boldsymbol{B}_i$ in a way that all have the same duration $b_{\mathrm{REF}}$. One can now use the synchronized trajectories from the proposed method to build an MPCA/MPLS model; the model will be still slightly biased towards false alarms (due to averaging of inconsistent features). However, it will be less prone to false alarms than a model, which was based on synchronized batches from an asymmetric DTW algorithm.

In the above discussion, it was assumed that the raw trajectories had been scaled appropriately and that also a reference trajectory had been defined. The next subsection deals with these issues, and proposes a complete method for synchronization of batch trajectories.

### *Iterative method for synchronization of batch trajectories*

As a distance-based method, DTW is sensitive to the scaling of variables. In the case of batch processes an intelligent scaling should accomplish two objectives. The first is to remove the effect of the various engineering units used to record the variables. This is easily achieved by dividing each variable by its standard deviation or its range. The second and most important objective is to give more weight to variables that are consistent from batch to batch. The synchronization of batch trajectories should rely more on these variables. This relative importance of variables is expressed through the weight matrix $\boldsymbol{W}$ used in the local distance computation in DTW, that is

$$d(i,j)=[\boldsymbol{B}_i(i,:)-\boldsymbol{B}_{\mathrm{REF}}(j,:)]\cdot\boldsymbol{W}\cdot[\boldsymbol{B}_i(i,:)-\boldsymbol{B}_{\mathrm{REF}}(j,:)]^T \quad (23)$$

One choice would be to assign subjective weight to each variable; however, this would require process knowledge and perhaps a number of ad hoc decisions. A more appealing choice would be to devise a procedure that would automatically detect and increase the weight of consistent variables and decrease the weights of the rest. For each variable, the sum of the squared deviation from the average trajectory over all batches could be used as an indicator of its consistency over different realizations.

Regarding $\boldsymbol{B}_{\mathrm{RFF}}$, a reasonable choice would be to set it equal to the average trajectory. However, at the start of the synchronization procedure it is not possible to average the batch trajectories since each one of them has a different duration. Thus, one trajectory from the set could be used as $\boldsymbol{B}_{\mathrm{RFF}}$. One could then synchronize all other trajectories to this particular one using the DTW/synchronization method of the previous subsection. After synchronization, all trajectories would have the same duration and so an average trajectory could be defined. The whole procedure could then be repeated and in the next iteration the average trajectory could be used as the referenced one.

These are essentially the main steps of the iterative procedure proposed for the synchronization of unequal batch trajectories, which is now being presented in detail.

*Step A: Scaling*

Let $\boldsymbol{B}_{\mathrm{RAW},i}$, $i=1,\ldots,I$, be a referenced set of trajectories which contain the raw measurements from $I$ good quality batches.

For each variable, find its average range by averaging the range from each batch; store these values because they will be used in the off-line and on-line monitoring of a new batch.

Divide each variable in all batches with its average range.

Let $\boldsymbol{B}_i$, $i=1,\ldots,I$ be the resulting scaled batch trajectories.

*Step B: Synchronization*

Step 0: Select one of the trajectories $\boldsymbol{B}_k$ as the referenced trajectory: $\boldsymbol{B}_{\mathrm{REF}}=\boldsymbol{B}_k$.

Consequently: $b_{\mathrm{REF}}=b_k$.

Set $\boldsymbol{W}$ (the weight matrix in the DTW algorithm) equal to the identity matrix.

Execute the following steps for a specified maximum number of iterations.

Step 1. Apply the DTW/synchronization method between $\boldsymbol{B}_i$, $i=1,\ldots,I$, and $\boldsymbol{B}_{\mathrm{REF}}$ as described in the previous subsection.

Let $\tilde{\boldsymbol{B}}_i$, $i=1,\ldots,I$ be the synchronized trajectories with $b_{\mathrm{REF}}$ now being their common duration.

Step 2. Compute the average trajectory $\bar{\boldsymbol{B}}$, that is, $\bar{\boldsymbol{B}}=\sum_{i=1}^{I}\tilde{\boldsymbol{B}}_i/I$.

Step 3. For each variable, compute the sum of squared deviations from $\bar{\boldsymbol{B}}$.

The inverse of this value will be the weight of the particular variable for the next iteration, that is, $\boldsymbol{W}$ will be a diagonal matrix with

$$\boldsymbol{W}(j,j)=\left[\sum_{i=1}^{I}\sum_{k=1}^{b_{\mathrm{REF}}}\left[\tilde{\boldsymbol{B}}_i(k,j)-\bar{\boldsymbol{B}}(k,j)\right]^2\right]^{-1}$$

Normalize $\boldsymbol{W}$ so that the sum of the weights is equal to the number of variables, that is, replace $\boldsymbol{W}$ with $\boldsymbol{W}\left\{N/\left[\sum_{j=1}^{N}\boldsymbol{W}(j,j)\right]\right\}$.

Step 4. For the first three iterations, keep the same referenced trajectory: $\boldsymbol{B}_{\mathrm{REF}}=\boldsymbol{B}_k$.

For subsequent iterations, set the reference equal to the average trajectory: $\boldsymbol{B}_{\mathrm{REF}}=\bar{\boldsymbol{B}}$.

The length of the synchronized trajectories at the end of the iterative procedure will be the length of the trajectory initially used as the referenced trajectory. Alternatively, one could estimate the average duration from the initial trajectories and the trajectory whose duration is closest to the average duration could be used as $\boldsymbol{B}_{\mathrm{REF}}$ for the first three iterations. By doing that, the duration of the synchronized trajectories at the end will be the average duration of the available realizations. The choice of the initial referenced trajectory is a matter of user preference. Finally, the maximum number of iterations is another parameter of the method set by the user. One could also monitor the change of the weight matrix $\boldsymbol{W}$ from one iteration to the next and use it as an indicator for convergence.

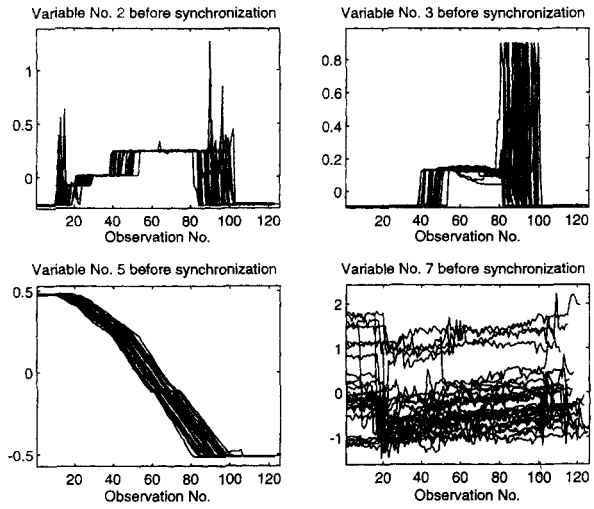


**Figure 5. Four (out of 10) variables during the 31 good quality batches before synchronization.**

The variables have been divided with their average range.

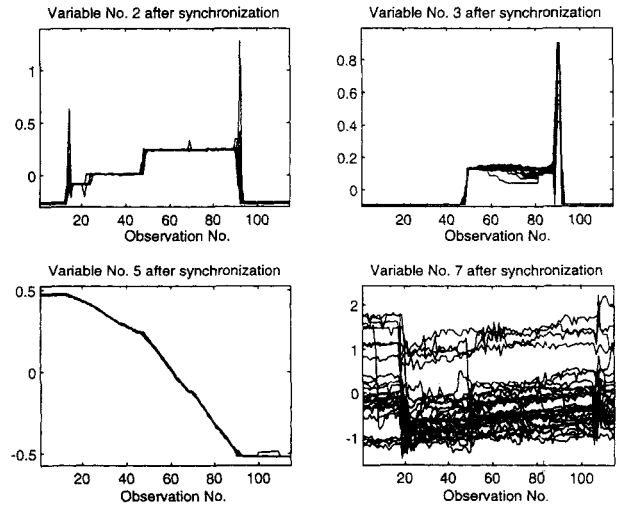


**Figure 6. Four variables of Figure 5 for all 31 batches after synchronization.**

## Case Studies

Figure 5 shows 4 variables (selected from the 10 variables in the data set) for 31 good quality batches from an industrial emulsion polymerization process; all variables have been scaled with their average range. The variables shown in Figure 5 illustrate a number of issues relating to batch process data. The most important is that the trajectories are not synchronized and do not have the same duration. Variable No. 5, with the exception of the starting and ending part, is smooth and strictly monotonic; thus, it could be used as an indicator variable. Variables No. 2 and 3 are piecewise constant with occasional step changes in their level. As such, they do not contain enough information to make them useful as indicator variables, but the times where their values step from one level to the next could be used to test the quality of the synchronization. Variable No. 7 is a noisy variable, and, therefore, one would not use it as an indicator variable.

The durations of the trajectories in this industrial data set vary from 106 to 126 data points and the average duration is 115. Three trajectories have the average duration and one of them ($\boldsymbol{B}_{21}$) was chosen to be the referenced trajectory for the first three iterations. For the DTW/synchronization procedure, the band global constraint was used with maximum allowable deviation $M = 35$ (from the linear path emanating from point (1,1)). The iterative procedure was executed for 10 iterations. The band global constraint was never active at any iteration and for any $\boldsymbol{B}_i - \boldsymbol{B}_{\mathrm{REF}}$ pair; it simply helped to speed up the computations.

The results after the final (10th) iteration are shown in Figures 6 and 7. Figure 6 shows the 4 variables after the trajectories have been synchronized and Figure 7 shows how the weights of the 4 variables changed with respect to the iterations. As Figure 6 illustrates, the variables are now synchronized. This is more apparent by looking at the times of the step changes in Variables Nos. 2 and 3 and the spike in Variable No. 3. Due to the averaging of selected points by the asymmetric synchronization procedure, some of the spikes in Variable No. 2 (see Figure 5) have been filtered; however, they are not completely removed.

Variable No. 5 is a smooth variable and it could be used as an indicator variable to synchronize the trajectories as Nomikos and MacGregor (1994) proposed. Such a synchronization was previously used in analyzing these batch trajectories by Kourti et al. (1996). The proposed iterative procedure validated this approach (as Figure 7 shows) since the weight of Variable No. 5 accounts for about 85% of the total weight (the indicator variable solution essentially gives 100% of the total weight to this variable). Finally, Variable No. 7 is a noisy variable (as Figure 5 shows) and is of little value for synchronization. The iterative procedure recognized this and gave small weight to Variable No. 7 after the first iteration (see Figure 7).

The iterative procedure could also be used to pinpoint the most appropriate variable to be used as an indicator variable if one wants to use this simpler method for synchronization without relying on expert process knowledge. There may be situations where several variables are smooth and monotonic; thus, they could all be candidates for the role of the indicator variable. The proposed method could assist in choosing the

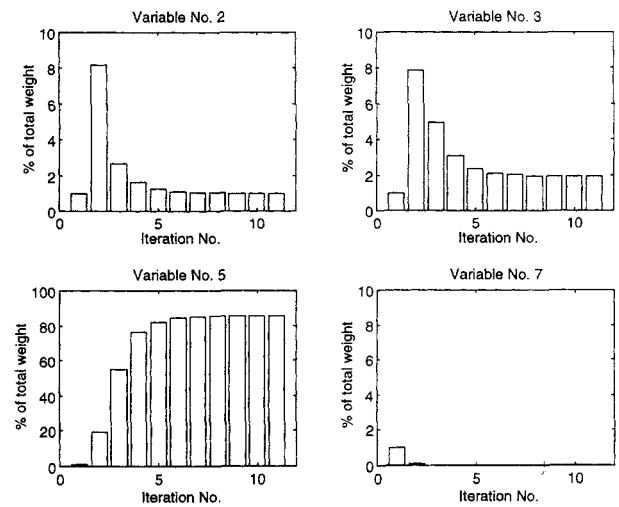


**Figure 7. Percentage of total weight vs. iteration number for the 4 variables in Figures 5 and 6.**

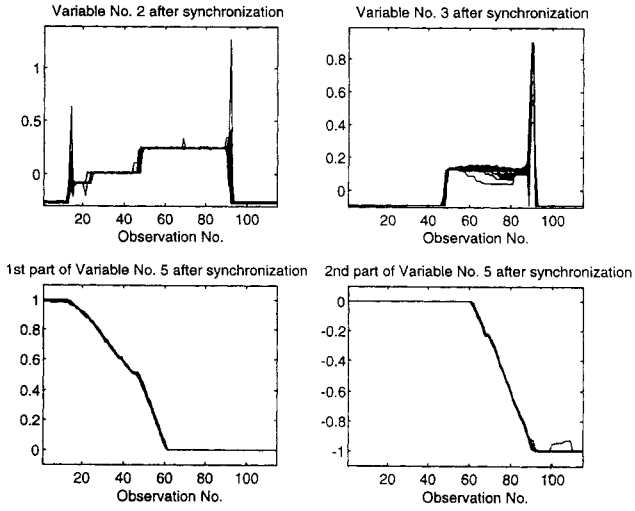


**Figure 8. Synchronization of trajectories after Variable No. 5 was spliced into 2 artificial variables.**

most appropriate one by selecting the variable that gets the largest weight in matrix $\boldsymbol{W}$.

To investigate the situation where there is no single variable that can be easily identified as an indicator variable, the following case study was performed. Variable No. 5 was spliced into two parts and two artificial variables were created. The first contains the initial part of Variable No. 5 up to the point that it reaches the value of zero; then, it is padded with zeros up to the end of the trajectory. Similarly, the second artificial variable contains initially a number of zeros, followed by the second part of Variable No. 5. Thus, for this case study, the batch trajectories contained 11 variables: the other 9 original variables (excluding Variable No. 5) plus the two artificial ones created from Variable No. 5.

Next, the synchronization method was applied with the same parameters described before and the results are presented in Figures 8 and 9. As Figure 8 shows, the synchronization of the variables is again quite good and almost identical to the one obtained before (Figure 6). The variable weights are shown in Figure 9. Interestingly, about 85% of

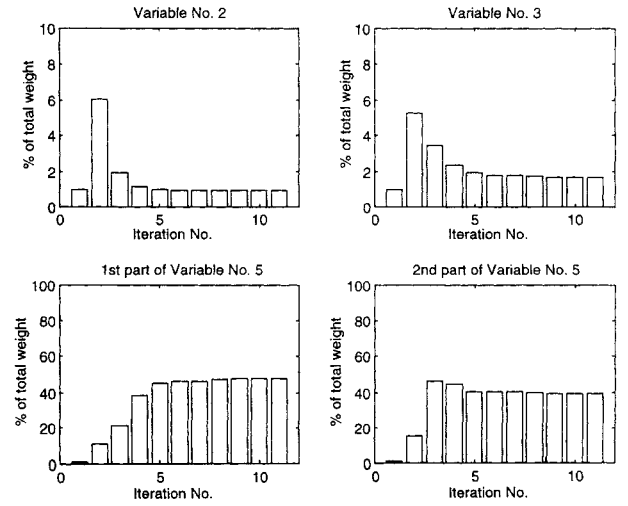


**Figure 9. Percentage of total weight vs. iteration number for the synchronization in Figure 8.**

the total weight is now distributed between the two artificial variables. Although not a proof, this case study shows that the proposed method can still perform well in cases where it is not possible to find a single variable that indicates the progression of the batch process.

## Batch Monitoring Using MPCA/MPLS

### *Off-line implementation*

Once timing differences have been removed by the synchronization procedure, the resulting trajectories $\tilde{\boldsymbol{B}}_i$, $i = 1, \ldots, I$ can be used to build an MPCA/MPLS model for process monitoring as proposed by Nomikos and MacGregor (1994, 1995b). However, one important feature has been removed from the raw data (the timing differences) and it could be the case that this feature is indeed affecting the final product quality. To account for this possibility, the amount of time distortion (expansion or compression) imposed by DTW on the duration of each batch and on the times between any identifiable checkpoint during each batch should be included in the MPLS model. These time distortions can be treated as additional variables in the initial condition matrix of the Multiblock MPLS model (see Kourti et al., 1995).

Now, assume that the complete trajectory of a new batch $\boldsymbol{B}_{\mathrm{RAW,NEW}}$ (a matrix of $b_{\mathrm{NEW}} \times N$) is available. The objective is to use the MPCA/MPLS-based monitoring scheme to assess the product quality of the new batch. Most probably the duration of the new batch $b_{\mathrm{NEW}}$ will not be equal to the duration of the synchronized batches $b_{\mathrm{REF}}$ that were used to construct the monitoring model. Even if $b_{\mathrm{NEW}} = b_{\mathrm{REF}}$, some stages of the new batch may not be synchronized with the corresponding stages of the referenced trajectory. In either case, $\boldsymbol{B}_{\mathrm{RAW,NEW}}$ has to be synchronized before the monitoring scheme is applied. The following method proposes to accomplish this task.

*Step A: Scaling*

Divide each variable in the new batch with the average range estimated from the trajectories of the referenced set.

Let $\boldsymbol{B}_{\mathrm{NEW}}$ be the resulting scaled new trajectory.

*Step B: Synchronization*

Let $\boldsymbol{B}_{\mathrm{REF}}$ and $\boldsymbol{W}$ be the referenced trajectory and the weight matrix used in the last iteration of the synchronization procedure.

Apply the DTW/synchronization method to synchronize $\boldsymbol{B}_{\mathrm{NEW}}$ with $\boldsymbol{B}_{\mathrm{REF}}$.

Let $\tilde{\boldsymbol{B}}_{\mathrm{NEW}}$ be the synchronized new trajectory.

$\tilde{\boldsymbol{B}}_{\mathrm{NEW}}$ is synchronized with the reference trajectories, its duration is $b_{\mathrm{REF}}$ and the MPCA/MPLS-based batch monitoring scheme can now be applied. Note that some points in $\tilde{\boldsymbol{B}}_{\mathrm{NEW}}$ will be averages of selected points of $\boldsymbol{B}_{\mathrm{NEW}}$ as a result of the asymmetric synchronization procedure. Since the averaging operation smooths spurious features, $\tilde{\boldsymbol{B}}_{\mathrm{NEW}}$ is slightly biased towards the null hypothesis, that is, the new batch being of good quality. This is a compromise that one has to accept if one wants to use the same MPCA/MPLS model to monitor each new batch.

### *On-line implementation*

The on-line implementation of the MPCA/MPLS-based monitoring scheme is similar to the off-line implementation with one important difference: in the on-line case, the prediction of the future behavior of the batch trajectory up to its

expected end is required. Nomikos and MacGregor (1995a) discuss possible methods to carry out these predictions. However, they assume that the new trajectory is synchronized with the referenced set trajectories either in time or with respect to an indicator variable. In real time, this assumption means that the progress of the new batch up to the current time $t$ is equivalent to the progress of the referenced set batches up to time $t$. Therefore, one has to predict the behavior of the new batch from the current time and onward up to its end; the end time for the new batch is assumed to be the common duration of the referenced set batches. This assumption may not be always true in industrial batch processes for the reasons given in the Introduction.

Let $\boldsymbol{B}_{\mathrm{RAW,NEW}}$ be the raw measurements of the evolving new batch, a matrix of $t \times N$, with $t$ being the number of data points from time zero up to the current time. To monitor the progress of the batch on-line, one would have to answer the following question: which point $r$ of the referenced trajectory best represents the progress of the new batch up to the current time? DTW can provide an answer to this question as follows:

*Step A: Scaling*

Divide each variable in $\boldsymbol{B}_{\mathrm{RAW,NEW}}$ with its average range estimated from the trajectories of the referenced set.

Let $\boldsymbol{B}_{\mathrm{NEW}}$ be the resulting scaled new trajectory.

*Step B: Synchronization*

Let $\boldsymbol{B}_{\mathrm{REF}}$ and $\boldsymbol{W}$ be the referenced trajectory and the weight matrix used in the last iteration of the synchronization procedure.

*Step 1:* Apply the DTW symmetric algorithm as previously presented. However, since only the first $t$ data points of the new batch are available, one would have a set of accumulated distances: $\boldsymbol{D}_A\ (t, j),\ j = \boldsymbol{l}(t), \ldots, \boldsymbol{u}(t)$; $\boldsymbol{l}(t)$ and $\boldsymbol{u}(t)$ are the lower and upper bound imposed by the band constraint on the index of the inner iteration.

Let $r$ be the point in the $\boldsymbol{D}_A(t,:)$ vector where the minimum occurs, that is, $r = \underset{j}{\operatorname{argmin}}[\boldsymbol{D}_A(t, j)]$.

*Step 2:* Synchronize the $t$ points of $\boldsymbol{B}_{\mathrm{NEW}}$ to the first $r$ points of $\boldsymbol{B}_{\mathrm{REF}}$ using the asymmetric synchronization method previously presented. After synchronization, the new trajectory, $\tilde{\boldsymbol{B}}_{\mathrm{NEW}}$, will have $r$ points.

*Step 3:* Predict the progress of the new batch from point $(r+1)$ up to the final point of the referenced trajectory $\boldsymbol{b}_{\mathrm{REF}}$. Now the MPCA/MPLS-based monitoring scheme can be applied on-line as described by Nomikos and MacGregor (1994, 1995b).

The above method has to be repeated as soon as another measurement from the new batch is available.

## Conclusions

An application of DTW for the synchronization and monitoring of batch processes was presented. Industrial batch processes are often characterized by unsynchronized trajectories of variable duration. However, the synchronization of the trajectories to a common length is a necessary condition for the application of many analysis and monitoring schemes. To solve the problem of batch synchronization, an iterative method was proposed based on DTW. The method is multivariate since it does not rely on a single variable to perform the synchronization in contrast to the indicator variable method. Moreover, the method can be used to pinpoint the most consistent variable, that is, the variable with the smallest deviation about its average trajectory. This variable could be used as the indicator variable at subsequent studies, if one wants to use this simpler approach. Once the batch trajectories are synchronized, one can then build the MPCA/MPLS batch monitoring model. To monitor a new batch, its timing differences with the referenced set batches should first be reconciled. To achieve this synchronization, a DTW-based procedure is proposed; guidelines are given for both off-line and on-line implementation.

## Literature Cited


Bellman, R. E., and S. E. Dreyfus, *Dynamic Programming*, Princeton Univ. Press, Princeton, NJ (1962).

Bertsekas, D. P., *Dynamic Programming*, Prentice-Hall, Englewood Cliffs, NJ (1987).

Gollmer, K., and C. Postens, "Detection of Distorted Pattern Using Dynamic Time Warping Algorithm and Application for Supervision of Bioprocesses," Preprints, IFAC Workshop on On-Line Fault Detection and Supervision in the Chemical Process Industries, Newcastle (1995).

Itakura, F., "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, **ASSP-23**(1), 67 (1975).

Kassidas, A., "Fault Diagnosis Using Speech Recognition Methods," PhD Diss., Dept. of Chemical Engineering, McMaster Univ., Hamilton, Ontario, Canada (1997).

Kourti, T., P. Nomikos, and J. F. MacGregor, "Analysis, Monitoring and Fault Diagnosis of Batch Processes Using Multiblock and Multiway PLS," *J. of Process Control*, **5**(4), 277 (1995).

Kourti, T., J. Lee, and J. F. MacGregor, "Experiences with Industrial Applications of Projection Methods for Multivariate Statistical Process Control," *Computers & Chem. Eng.*, **20**, Suppl. A., 745 (1996).

Lakshminarayanan, S., R. D. Gudi, S. L. Shah, and K. Nandakumar, "Monitoring Batch Processes using Multivariate Statistical Tools: Extensions and Practical Issues," IFAC Triennial World Cong., San Francisco (1996).

MacGregor, J. F., and P. Nomikos, "Monitoring Batch Processes," Batch Processing System Engineering, G. V. Reklaitis et al., eds., NATO ASI, Ser. F, Vol. 143, p. 242 (1992).

Myers, C., L. R. Rabiner, and A. E. Rosenberg, "Performance Tradeoffs in Dynamic Time Warping Algorithms for Isolated Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, **ASSP-28**(6), 623 (1980).

Nadler, M., and E. P. Smith, *Pattern Recognition Engineering*, Wiley, New York (1993).

Ney, H., "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, **ASSP-32**(2), 263 (1984).

Nomikos, P., and J. F. MacGregor, "Monitoring Batch Processes Using Multiway Principal Component Analysis," *AIChE J.*, **40**, 1361 (1994).

Nomikos, P., and J. F. MacGregor, "Multivariate SPC Charts for Monitoring Batch Processes," *Technometrics*, **37**(1), 41 (1995a).

Nomikos, P., and J. F. MacGregor, "Multi-way Partial Least Squares in Monitoring Batch Processes," *Chemometrics and Intelligent Laboratory Systems*, **30**, 97 (1995b).

O'Shaughnessy, D., "Speaker Recognition," *IEEE ASSP Mag.*, **3**, 4 (1986).

Rabiner, L. R., A. E. Rosenberg, and S. E. Levinson, "Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Process.*, **ASSP-26**(6), 575 (1978).

Sakoe, H., and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Process.*, **ASSP-26**(1), 43 (1978).

Silverman, H. F., and D. P. Morgan, "The Application of Dynamic Programming to Connected Speech Recognition," *IEEE ASSP Mag.*, **7**, 7 (1990).


*Manuscript received Apr. 14, 1997, and revision received Jan. 9, 1998.*