Une distance entre deux ensembles de séquences avec la contrainte de continuité

Application à des données en-ligne

Jinpeng Li, Harold Mouchère, Christian Viard-Gaudin

IRCCyN (UMR CNRS 6597) - L'UNAM - Université de Nantes, France
{jinpeng.li,harold.mouchere,christian.viard-gaudin}@univ-nantes.fr

RÉSUMÉ. *L'algorithme DTW (Dynamic Time Warping) est très largement utilisé pour calculer une distance entre des données en-ligne. Il permet une mise en correspondance point à point en respectant des contraintes de séquentialité pendant l'alignement entre les deux séquences correspondant aux tracés. Toutefois lorsque le tracé se présente sous la forme d'un nombre variable de traits pour lesquels le sens de chaque trait n'est pas imposé alors cet algorithme n'est plus directement utilisable. À l'inverse la distance de Hausdorff fait fi de l'ordre des points dans la séquence mais aussi de la continuité des points à l'intérieur d'un même trait, et est très sensible aux points excentrés (outliers). Dans ce papier, nous discutons de l'alignement entre deux ensembles de séquences de points avec contrainte de continuité intra-séquence, et nous proposons une mesure de distance découlant de cet alignement.*

ABSTRACT. *Dynamic time warping (DTW) is an algorithm which can well measure the distance between two strokes (two point sequences) in on-line handwriting. It obeys the continuity constraint during the point-to-point alignment between two sequences. But considering two isolated multi-stroke similar symbols (two sequence sets) whose directions and orders are varied, DTW cannot be directly used anymore. On the contrary, the Hausdorff distance can settle such two problems, different stroke directions and orders. However, it is very sensitive to the outliers. In this paper, we will discuss the alignment between two sets of sequences with the continuity constraint, and a distance can be computed from this alignment.*

MOTS-CLÉS : *Déformation Temporelle Dynamique, Deux Ensembles de Séquences, Appariement de Deux Caractères Isolés, Distance de Hausdorff.*

KEYWORDS: *Dynamic Time Warping, Two Sets of Sequences, Two Isolated Multi-stroke Characters Matching, Hausdorff Distance.*

1. Introduction

Le processus de reconnaissance d'écritures en-ligne peut être divisé en deux étapes principales, la segmentation de symboles et la reconnaissance des symboles isolés (Chan *et al.,* 2000). Dans ce papier, nous discutons de la similarité entre deux symboles isolés. Le symbole isolé est composé d'un ensemble de traits. Chaque trait (*stroke* en anglais) est représenté par une séquence de points, du point de départ (posé) au point d'arrivée (levé). Un trait est donc orienté. Calculer la distance entre deux symboles isolés est donc un problème de comparaison de deux ensembles de séquences orientées de points.

Différentes personnes peuvent écrire un même symbole avec des sens différents pour chaque trait et utilisant des ordres différents entre les traits. Dans la recherche d'identification de scripteurs, cette caractéristique peut distinguer efficacement les scripteurs (Tan *et al.,* 2009). Néanmoins, pour comprendre ou communiquer avec un même symbole écrit par les personnes différentes, le sens et l'ordre des traits doivent être ignorés. Nous lisons des symboles manuscrits sans avoir accès ni au sens ni à l'ordre. Par exemple, un symbole contenant un trait horizontal"$-$" peut être écrit de deux manières, de gauche à droite "$\rightarrow$" ou l'inverse "$\leftarrow$".

Lors de la comparaison de deux symboles chacun écrit d'un seul trait, l'algorithme DTW (Dynamic Time Warping) permet une mise en correspondance point à point en respectant des contraintes de séquentialité pendant l'alignement entre les deux séquences (Vuori, 2002). Cet alignement décrit dans la section suivante respecte la contrainte de continuité temporelle de l'alignement.

Si l'on calcule l'alignement de deux traits avec deux directions opposées, la distance DTW $dist_{DTW}(\rightarrow, \leftarrow)$ produit naturellement une valeur importante du fait de l'inversion du sens de parcours. Une solution consiste à choisir la plus petite distance DTW entre les deux directions possibles d'un trait : $\min(dist_{DTW}(\rightarrow, \leftarrow), dist_{DTW}(inv(\rightarrow), \leftarrow))$ où $inv(.)$ est un opérateur d'inversion de l'ordre de parcours des points d'un trait. Cependant, le nombre de combinaisons augmente très rapidement par rapport au nombre de traits dans un symbole. Le Tableau 1 montre la complexité des ordonnancements et des sens de parcours des traits d'un tracé en-ligne. D'une manière générale, le nombre de séquences est donnée par la formule :

$$S = N! \times 2^N. \qquad [1]$$

Pour calculer la distance DTW entre deux symboles multi-traits, une solution directe consiste à concaténer les différents traits en prenant en compte un certain ordre de tracé. Ainsi pour calculer la distance entre  (4 traits) et  (2 traits) il faut calculer 384 $\times$ 8 = 3092 appariements possibles. Ce grand nombre d'appariements est dû à deux causes complémentaires : l'ordonnancement respectif des traits (en nombre N !) et le sens de parcours de chaque trait (pour chaque ordonnancement, 2^N variantes de parcours).

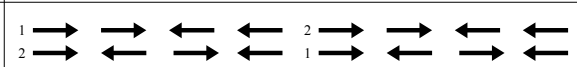
Nombre de traits(N)	Exemple	Nombre de séquences(S)	Illustration des tracés
1		2	
2		8	
3		48	
4		384	

Tableau 1 – Variabilité des ordonnancements et des sens de parcours des traits dans un tracé en-ligne

Pour calculer la distance entre deux symboles multi-traits une autre approche consiste à considérer ces deux ensembles de séquences de points comme deux ensembles de points, en ignorant l'information temporelle. Nous pouvons alors utiliser une métrique du domaine du traitement d'images, comme par exemple la distance de Hausdorff (Huttenlocher *et al.,* 1993) qui permet de mesurer l'éloignement de deux ensembles de séquences de points comme deux ensembles de points, $S_1 = \{p_1(i)\}$ et $S_2 = \{p_2(j)\}$:

$$d_H(S_1, S_2) = \max\{d_h(S_1, S_2), d_h(S_2, S_1)\}, \quad [2]$$

où $d_h(S_A, S_B) = \max_{p_A \in S_A} \min_{p_B \in S_B} d(p_A, p_B)$ et $d(p_A, p_B)$ est de la distance euclidienne entre deux points. Mais cette distance ne vérifie pas la contrainte de continuité temporelle intra-séquence.

De nombreux travaux (Uchida *et al.,* 2005, Levin *et al.,* 1992, Uchida *et al.,* 1999) étendent cette contrainte de continuité d'une dimension (time warping) aux deux dimensions spatiales (two-dimensional warping). Ils mettent en correspondance deux ensembles de points (pixels) avec la contrainte de continuité spatiale. Mais la continuité de séquences n'est pas considérée.

Dans ce papier, nous discutons de l'alignement entre deux ensembles de séquences de points avec la contrainte de continuité intra-séquence. Le meilleur alignement doit être trouvé à partir d'un grand nombre possible d'appariements. Une procédure de recherche directe serait très lente à cause des nombreuses possibilités. Pour estimer rapidement le meilleur alignement, l'algorithme de recherche A* (A étoile) (Hart *et al.,* 1968) permettant de réduire le nombre d'appariements est utilisé. Cet algorithme

est basé sur une évaluation heuristique du coût de chaque appariement. Pour optimiser l'algorithme A*, nous proposons aussi une stratégie particulière de parcours du graphe des possibilités d'alignement.

Dans la suite de ce papier nous introduisons la problématique de la distance entre deux ensembles de séquences. Puis l'algorithme A* et notre stratégie de recherche sont présentés. Dans la dernière section de ce papier nous présentons des résultats qualitatifs et concluons notre travail.

2. Définition de la problématique

Nous commençons ici par présenter la version classique de l'alignement de deux séquences de points en utilisant DTW. Puis nous présentons la problématique de l'extension de l'algorithme DTW à un ensemble de séquences.

2.1. *DTW entre deux séquences de points*

Notre objectif est de comparer deux symboles isolés. Nous commençons pas présenter le cas simple où les deux symboles contiennent chacun un seul trait. Deux traits $S_1 = (p_1(1), ..., p_1(N_1))$ et $S_2 = (p_2(1), ..., p_2(N_2))$ seront donc comparés. L'algorithme DTW (Dynamic Time Warping) permet de calculer la distance entre deux séquences de données qui varient temporellement. Cette méthode a d'abord été appliquée dans le domaine du traitement de la parole afin de mettre en correspondance des échantillons acoustiques. Comme pour la parole, les données manuscrites en-ligne contiennent une information temporelle. Bien que coûteuse en temps de calcul, cette distance a déjà montré son efficacité dans plusieurs travaux (Ruiz *et al.,* 1985, Vuori, 2002, Narita *et al.,* 2008).

Les grands principes de l'algorithme DTW (Vuori, 2002) sont résumés ci-après. Soit le chemin $P(h) = (i(h), j(h)), 1 \leqslant h \leqslant H$ permettant de décrire l'alignement point à point où h est de l'indice d'ordre d'appariement du $i(h)$ème point et du $j(h)$ème point des traits S_1 et S_2 respectivement.

$P(h)$ doit respecter la contrainte de frontière et la contrainte de continuité. La première contrainte de frontière est définie par :

$$\begin{aligned} &P(1) = (i(1), j(1)) = (1, 1), \\ &P(H) = P(i(H), j(H)) = (N_1, N_2). \end{aligned} \qquad [3]$$

Les deux points de départ et les deux points de fin doivent être appareillés respectivement dans les deux traits. L'Équation 4 ci-dessous explicite la seconde contrainte, la contrainte de continuité temporelle de l'alignement :

$$(\triangle\, i(h), \triangle\, j(h)) = (i(h) - i(h-1), j(h) - j(h-1)) = \begin{cases} (1,0) or \\ (0,1) or \\ (1,1). \end{cases} \quad [4]$$

Cette relation montre que le décalage entre paires de points dans l'appariement est au maximum de 1. De plus tous les points sont utilisés au moins une fois. Calculer la distance entre deux séquences consiste donc à chercher parmi tous les appariements possibles celui minimisant la somme des distances point à point :

$$D(S_1, S_2) = \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))), \quad [5]$$

où $d(.,.)$ est en général la distance euclidienne entre deux points. La solution à l'Équation 5 peut être résolue récursivement par programmation dynamique en définissant la matrice $D(i, j; h)$ des distances cumulées :

$$D(i, j; h) = d(p_1(i), p_2(j)) + \min \begin{cases} D(i-1, j; h-1) \\ D(i, j-1; h-1) \\ D(i-1, j-1; h-1), \end{cases} \quad [6]$$

avec $D(i, j; 0) = 0$ pour l'initialisation.

La Figure 1 donne un exemple de deux séquences de points à comparer. Les points de départ des traits sont représentés par les ronds rouges. Nous cherchons le chemin avec le coût minimum avec l'Équation 6. Nous calculons d'abord la matrice d'accumulation montrée dans la Figure 2. Le meilleur chemin (i.e. alignement) peut être déduit par retour arrière à partir du couple de points terminal : P(1),...,P(8) = (1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,6),(8,6). Si nous définissons un couple de points de départ et un couple de points de fin, le meilleur alignement sera trouvé. Dans la section suivante, nous introduirons la comparaison entre deux ensembles de séquences.

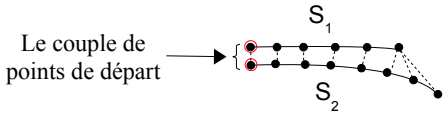


Figure 1 – Deux séquences de points (deux traits)

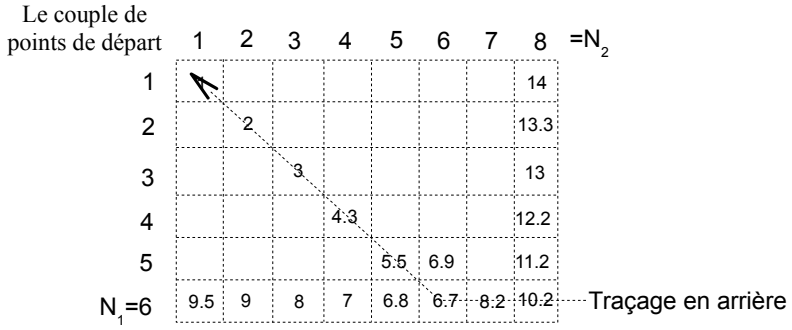


Figure 2 – Représentation de la matrice d'accumulation $D(i, j; h)$ de l'Équation 6 et d'un chemin de mise en correspondance.

2.2. *DTW avec deux ensembles de séquences*

Cette section décrit l'alignement entre deux ensembles de séquences de points. Nous utilisons pour illustrer notre explication la matrice des distances point à point. Soit deux symboles contenant un certain nombre de traits, les côtés de cette matrice représentent les traits des deux symboles respectivement en ligne et en colonne. L'ordre des traits dans la matrice n'est pas obligatoirement l'ordre du tracé effectif.

Le principe de notre approche consiste à construire itérativement un alignement par partie jusqu'à utilisation de tous les points. Si nous choisissons un couple de points de départ, quatre directions d'alignement sont possibles, elles correspondent aux quatre orientations diagonales qui définissent les quatre possibilités d'aligner deux traits. À chaque fois il faut chercher le meilleur chemin consommant au moins un des deux traits. Pour trouver ce meilleur chemin, les quatre matrices d'accumulation (la Figure 3) sont calculées pour chacune de ces directions. Les bornes de ces matrices sont définies par la fin de chaque trait.

Par exemple, soit deux symboles contenant respectivement deux et trois traits, les traits de ces deux symboles sont représentés respectivement en ligne et en colonne dans la matrice de la Figure 3. Nous pouvons voir qu'à partir d'un couple de points de départ (le rectangle bleu dans la figure) il existe quatre directions possibles d'alignement.

Dans chacune de ces quatre matrices d'accumulation, nous pouvons appliquer l'algorithme DTW classique comme présenté dans la section précédente. Toutefois, nous permettons à DTW de ne pas finir forcément dans l'angle opposé au point de départ.

En effet, dans la Figure 2 nous pouvons constater que l'alignement se terminant par P(8)=(8,6) n'est pas le meilleur en termes de distances cumulées. Nous pouvons couper l'alignement en choisissant la valeur minimum sur les côtés de la matrice :

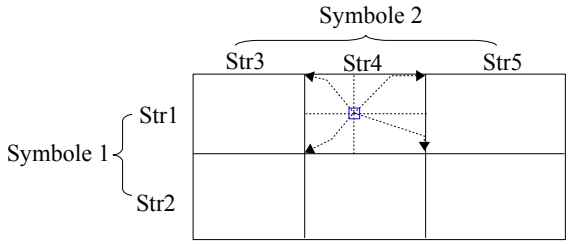


Figure 3 – Exemple de matrice de distance entre deux symboles avec 2 et 3 traits respectivement.

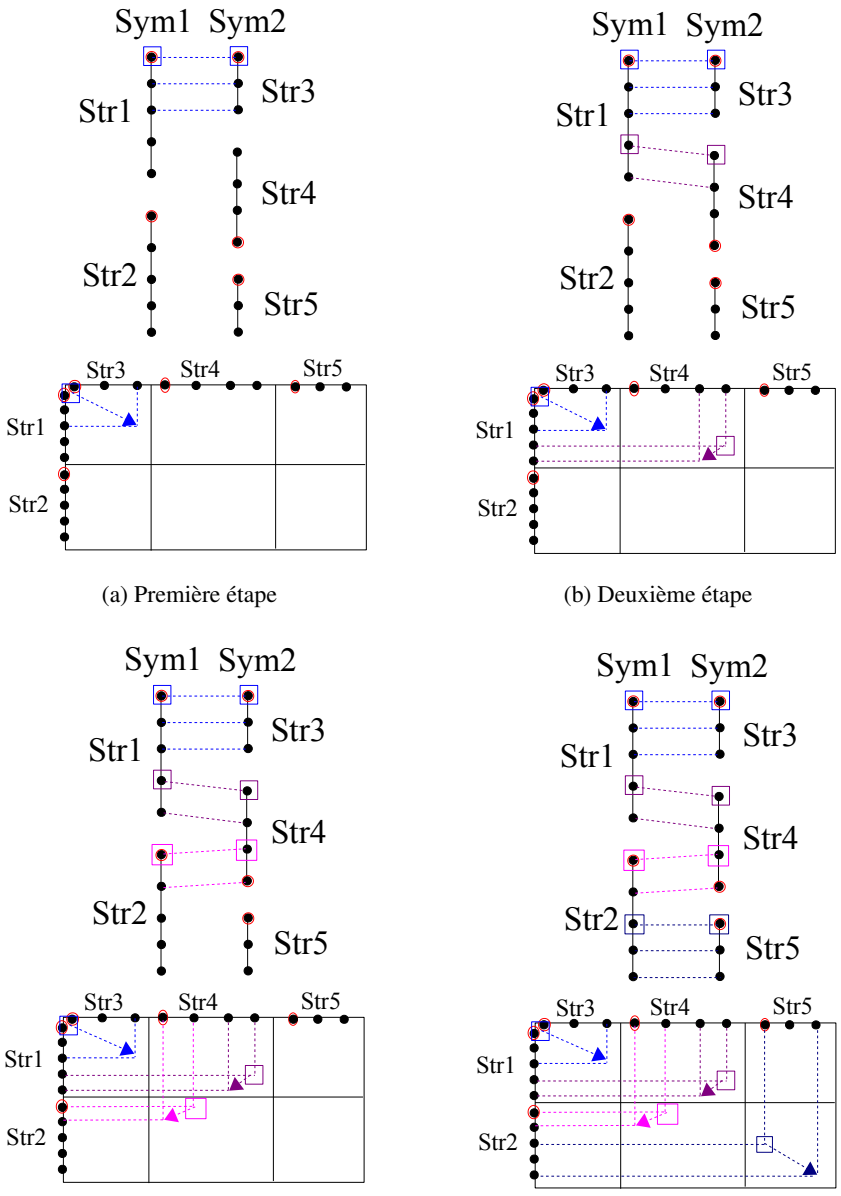
$D(1, N_2), D(2, N_2), ..., D(N_1, N_2)$ et $D(N_1, 1), D(N_1, 2), ..., D(N_1, N_2)$. Concrètement, nous calculons d'abord la matrice d'accumulation complète jusqu'à la fin des deux traits. Puis l'alignement sera arrêté sur la consommation complète d'un des deux traits. Un nouveau couple de points de fin est donc défini. Par exemple dans la Figure 2, nous choisissons le sous-chemin : (1,1),(2,2),(3,3),(4,4),(5,5),(6,6).

Dans cette stratégie, les couples de points de départ sont choisis pour associer les deux sous-séquences en respectant la contrainte de continuité dans chaque étape. À chaque étape nous répétons le choix de couple de points parmi les points non-utilisés. La procédure sera finie lorsque tous les points seront utilisés au moins une fois. Notre objectif est de trouver la somme minimum de coût du chemin d'appariements (Équation 5). Enfin la distance entre deux ensembles de séquences, $Sym1 = (p_1(1), ..., p_1(N_1))$ et $Sym2 = (p_2(1), ..., p_2(N_2))$, est normalisée par le nombre d'associations :

$$D(Sym1, Sym2) = \frac{1}{H} \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))). \quad [7]$$

La Figure 4 montre une solution pour associer deux ensembles de séquences, c'est-à-dire un chemin. Cette solution contient quatre sous-chemins de DTW. Les sens d'alignement entre deux traits ne sont pas forcément identiques. Nous allons chercher l'ensemble de sous-chemins qui minimise le coût d'association (somme des distances point à point).

Nous pouvons maintenant comprendre la complexité du problème. En effet, pour commencer chaque sous-chemin il existe un grand nombre de possibilités pour choisir le couple de départ. Á partir de ces point de départ il existe encore beaucoup de chemins potentiels (choix du point d'arrivée). Pour rechercher la meilleure combinaison de sous-chemins, nous choisissons l'algorithme A* (Hart *et al.,* 1968) qui permet d'accélérer la recherche, celle-ci est décrite dans la section suivante.



(a) Première étape

(b) Deuxième étape

(c) Troisième étape

(d) Quatrième étape

Figure 4 – Une solution pour associer deux ensembles de séquences (vues matricielles et graphiques)

3. Algorithme A*

L'algorithme A* recherche itérativement un chemin dans un graphe (ici un arbre n-aire) en partant d'un état de départ (aucun point aligné) à un état d'arrivée (tous les points alignés). Tout l'arbre de recherche n'est pas généré (c'est ce qui fait l'efficacité de A*). A chaque itération, seule la meilleure hypothèse est explorée par génération des états suivants.

Pour accélérer la recherche, l'algorithme de recherche A* utilise une évaluation heuristique $f(x) = g(x) + h(x)$ du coût de chaque étape x : un coût $g(x)$ donnant le coût du meilleur chemin arrivant à x plus $h(x)$ pour estimer le coût permettant d'atteindre le but final. La référence (Hart *et al.,* 1968) précise cet algorithme. Dans cette section, nous définissons les fonctions $g(x)$ et $h(x)$ adaptées à notre problème ainsi que le choix des points de départ permettant de créer les étapes qui suivront x.

3.1. *Le coût de chemin g(.) et l'heuristique h(.)*

Nous définissons chaque étape x par le chemin $P_x(h) = (i_x(h), j_x(h))$, $1 \leqslant h \leqslant H_x$ entre deux symboles $Sym1 = (p_1(1), ..., p_1(N_1))$ et $Sym2 = (p_2(1), ..., p_2(N_2))$. Le coût du chemin est défini par la somme de la distance entre deux points d'appariement :

$$g(x) = \sum_{1}^{H_x} d(p_1(i(h)), p_2(j(h))). \qquad [8]$$

Concernant l'heuristique $h(.)$, elle correspond au coût nécessaire minimum pour aller de l'étape actuelle x à l'étape du but. L'heuristique $h(.)$ doit être admissible, i.e. $h(.)$ ne surestime jamais la distance à l'étape finale. Pour l'admissibilité, nous définissons l'ensemble des points non-utilisés $NUPt(Sym, x)$ pour un symbole Sym à l'étape x. L'heuristique $h(x)$ est donc définie par

$$h(x) = \frac{1}{2}(h_{sub}(x, Sym1, Sym2) + h_{sub}(x, Sym2, Sym1)), \qquad [9]$$

où

$$h_{sub}(x, SymA, SymB) = \sum_{p_1(i) \in NUPt(SymA, x)} d(p_1(i), ppv(p_1, SymB))$$

$$ppv(p_1, SymB) = \underset{p_2(j) \in NUPt(SymB, x)}{\arg\min} d(p_1, p_2(j)). \qquad [10]$$

Cette heuristique est admissible car nous choisissons toujours la distance minimum entre tous les points non-utilisés (le coût de l'alignement de ces points sera forcément supérieur).

3.2. *Le choix des points de départ*

Pour générer les étapes suivantes, il faut choisir un couple de points non-utilisés qui permettra de démarrer la mise en correspondance suivant quatre directions au maximum. Pour chaque direction, une nouvelle étape est obtenue. Bien que l'algorithme A* puisse réduire la complexité de recherche, il existera encore de nombreuses possibilités si toutes les combinaisons entre les points non-utilisés sont choisies (cas le plus général). Dans cette section, une stratégie est donc proposée pour limiter la complexité en limitant les possibilités de points de départ.

Nous définissons les *segments* non-utilisés à l'étape x pour chaque trait dans un symbole par $Segs(Sym, x)$ ainsi que les points de frontière de ces segments par $FSeg(Sym, x)$. L'ensemble des nouveaux couples de points $\{(p_i, p_j)\}$ entre deux symboles, $Sym1$ et $Sym2$, sont produits depuis $FSeg(Sym1, x)$ vers les points plus proches dans $Segs(Sym, x)$ et inversement :

$$\begin{aligned} &\{(p_i, p_j)\} = \\ &\{\forall p_i \in FSeg(Sym1, x), \forall seg \in Segs(Sym2, x), (p_i, ppv(p_i, seg))\} \\ &\qquad\qquad \cup \\ &\{\forall p_j \in FSeg(Sym2, x), \forall seg \in Segs(Sym1, x), (ppv(p_j, seg), p_j)\} \end{aligned} \quad [11]$$

La Figure 5 montre les couples de départ possibles à partir de l'alignement de la Figure 4b. Il existe six couples de points qui peuvent s'étendre dans quatre directions respectivement. Toutes ces possibilités sont exploitées par l'algorithme de recherche A* pour trouver le meilleur alignement complet.

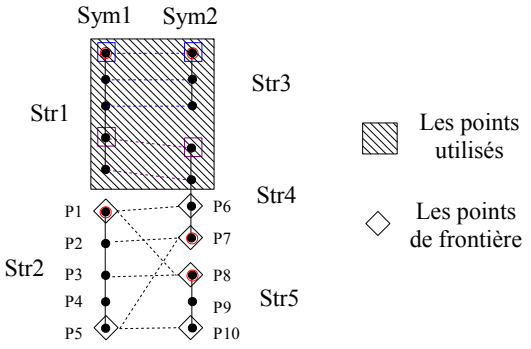


Figure 5 – Six couples de points de départ (P1,P6), (P1,P8), (P5,P7), (P5,P10), (P2,P7) et (P3,P8) pour continuer la mise en correspondance démarrée dans la Figure 4b

Notons que le choix des points de départ (mais aussi des points d'arrivée) agit sur deux propriétés de DTW A* :

– la qualité de la solution finale : si certaines possibilités sont trop limitées, elles ne pourront pas apparaître dans les alignements finaux,

– la rapidité d'exécution : moins il y a de possibilités, moins il y a de branches à évaluer et plus la recherche est rapide.

4. Etude expérimentale

L'algorithme est encore coûteux en temps et en mémoire pour le calcul malgré la réduction des points de départ présentée dans la section précédente. Nous proposons dans cette section une étude qualitative qui montre la qualité de l'alignement obtenu.

4.1. *L'alignement entre deux sembles de séquences*

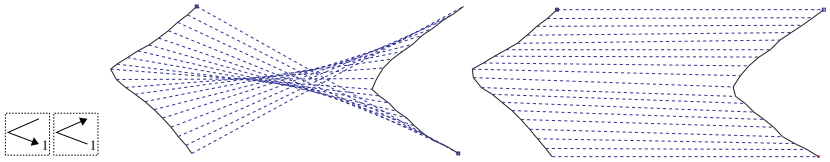
Avant la mise en correspondance, tous les symboles sont re-échantillonnés en un nombre fixe de 20 points et sont mis à la même échelle. Les seules coordonnées (x,y) sont utilisées pour calculer la distance euclidienne entre deux points. La Figure 6 compare les résultats obtenus par l'algorithme DTW classique et par notre DTW A* entre deux formes similaires mais écrites utilisant des sens d'écriture et des ordres de traits différents. La Figure 6a présente un exemple de deux séquences avec deux sens différents.

La distance trouvée avec l'algorithme DTW A* proposé ici, visible sur la Figure 6c est plus petite que celle obtenue avec la DTW classique, Figure 6b. Les exemples présentés des cas 2 (Figure 6d) au cas 4 (Figure 6j) comparent un symbole composé d'un trait avec le même symbole composés de deux traits écrits dans différents sens et ordres. Notre algorithme choisit le meilleur alignement parmi les différents sens et ordres possibles. Le dernier cas, Figure 6m, montre la capacité du système à aligner deux symboles multi-traits avec inversion du sens et de l'ordre. Ces exemples montrent bien que notre algorithme est indépendant du sens et de l'ordre des traits composant les symboles.

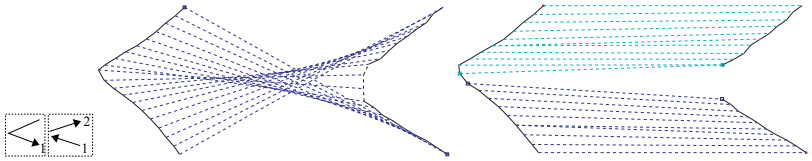
La Figure 7 montre un exemple plus compliqué de comparaison de deux allographes de x. Notre algorithme a trouvé la meilleure solution en cinq étapes. Les deux premières étapes montrent l'alignement de la branche en haut à gauche. La branche en bas à droite est alignée dans la troisième étape, etc. Notre algorithme peut couper les traits en sous-segments qui minimisent la distance DTW avec le segment correspondant dans l'autre symbole.

4.2. *Distance entre symboles différents*

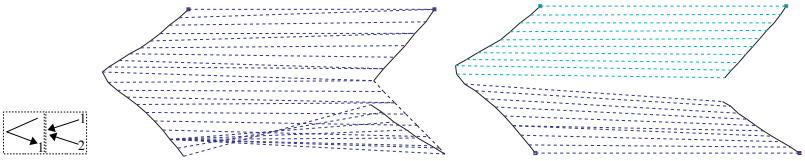
Nous choisissons six symboles pour illustrer l'utilisation de cette distance (Équation 7) dans une application de type classifieur basé plus-proche-voisin. Les exemples sont ré-échantillonné en 30 points. Nous pouvons constater que les formes '4' et '8'



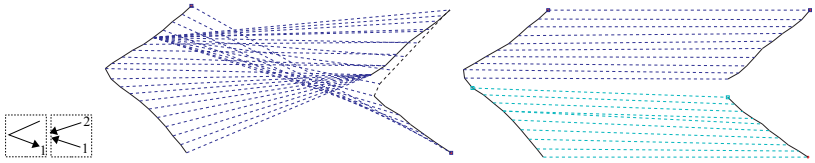
(a) Cas 1 (b) DTW classique (distance=1,08) (c) DTW A* (distance=0,10)



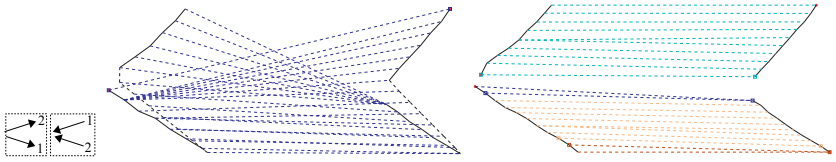
(d) Cas 2 (e) DTW classique (distance=1,09) (f) DTW A* (distance=0,11)



(g) Cas 3 (h) DTW classique (distance=0,214) (i) DTW A* (distance=0,15)



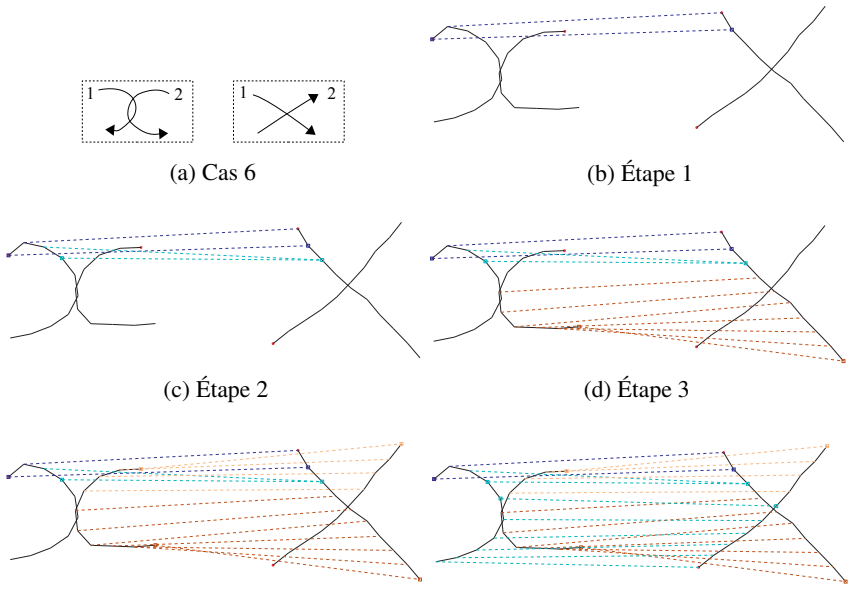
(j) Cas 4 (k) DTW classique (distance=0,69) (l) DTW A* (distance=0,10)



(m) Cas 5 (n) DTW classique (distance=0,55) (o) DTW A* (distance=0,10)

Figure 6 – Tests sur la mise en correspondance entre deux ensembles de sequences

sont bien classifiées bien que les allographes n'aient pas le même nombre de traits. Le temps de calcul et le nombre d'hypothèses (la mémoire requise) sont donnés à titre indicatif et dépendent de la complexité de la forme et surtout du nombre de points utilisé pour le ré-échantillonnage.



(a) Cas 6 (b) Étape 1

(c) Étape 2 (d) Étape 3

(e) Étape 4 (f) Étape 5

Figure 7 – La meilleure solution entre deux x

	(2 str)	(1 str)	(2 str)	(1 str)
(2 str)	dist=0,29 temps=127 sec 77.304 hyp	dist=0,37 temps<1sec 3.749 hyp	dist=0,23 temps=787sec 238.193 hyp	dist=0,17 temps<1sec 218 hyp
(1 str)	dist=0,15 temps<1 sec 112 hyp	dist=0,44 temps<1 sec 699 hyp	dist=0,27 temps=176 sec 88.820 hyp	dist=0,37 temps<1sec 16 hyp

Tableau 2 – Distance entre deux symboles et quatre symboles (dist=distance, sec=seconde, str=trait et hyp=hypothèse).

5. Conclusion

Dans ce papier, nous avons proposé une distance entre deux ensembles de séquences en préservant la continuité intra-séquence. En procédant directement, il fau-

drait trouver le meilleur alignement parmi un nombre exponentiel de combinaisons. Notre approche nommée DTW A*, basée sur la distance DTW et l'algorithme de A*, permet de réduire cette complexité en explorant les sous-alignements possibles. Nous limitons notamment les points de départ des alignements possibles pour réduire la combinatoire. Cette stratégie permet aussi d'obtenir des alignements qui n'aurait pas été possible par un simple ré-ordonnancement des traits. Les résultats qualitatifs présentés montrent l'intérêt de DTW A*. Il s'agit maintenant de réaliser une implémentation permettant la mise en œuvre d'un classifieur. En particulier il sera utile d'arrêter le calcul de la distance dès qu'elle dépasse la plus petite déjà disponible.

Remerciements : Nous remercions en particulier notre collègue Nicolas Normand pour ses idées constructives dans ces travaux. Ce travail est supporté par le projet DEPART `www.projet-depart.org` au niveau de la région des Pays de la Loire, France.

6. Bibliographie

Chan K.-F., Yeung D.-Y., « Mathematical Expression Recognition : A Survey », 2000.

Hart P., Nilsson N., Raphael B., « A Formal Basis for the Heuristic Determination of Minimum Cost Paths », *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, n° 2, p. 100-107, February, 1968.

Huttenlocher D. P., Klanderman G. A., Rucklidge W. A., « Comparing Images Using the Hausdorff Distance », *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, n° 9, p. 850-863, 1993.

Levin E., Pieraccini R., « Dynamic planar warping for optical character recognition », *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 3, p. 149-152 vol.3, 1992.

Narita H., Sawamura Y., Hayashi A., « Learning a Kernel Matrix for Time Series Data from DTW Distances », p. 336-345, 2008.

Ruiz E. V., Casacuberta F., Segovia H. R., « Is the DTW "distance" really a metric ? An algorithm reducing the number of DTW comparisons in isolated word recognition. », *Speech Communication*, vol. 4, n° 4, p. 333-344, 1985.

Tan G. X., Viard-Gaudin C., Kot A. C., « Automatic writer identification framework for online handwritten documents using character prototypes », *Pattern Recogn.*, vol. 42, n° 12, p. 3313-3323, 2009.

Uchida S., Sakoe H., « Handwritten Character Recognition Using Monotonic and Continuous Two-Dimensional Warping », *PROC. ICDAR*, p. 499-502, 1999.

Uchida S., Sakoe H., « A Survey of Elastic Matching Techniques for Handwritten Character Recognition », *IEICE Transactions*, vol. 88-D, n° 8, p. 1781-1790, 2005.

Vuori V., Adaptive Methods for On-Line Recognition of Isolated Handwritten Characters, PhD thesis, Helsinki University of Technology (Espoo, Finland), 2002.