Université de Liège

Structures des données et algorithmes (INFO0902-1)

Projet 3 : Résolution de problèmes

Auteurs :
Louis Hogge s192814
Tom Weber s203806

Professeur : P. Geurts
Année : 2021-2022

1 DynamicTimeWraping

1.1 Définition

Dans le cadre du Dynamic Time Warping, la fonction de coût, $d(x_i, y_j)$ qu'on cherche à minimiser est la distance moyenne absolue. Elle est définie ci-dessous :

$$d(x_i, y_j) = \frac{1}{2}\left(\mid x_{i,1} - y_{j,1} \mid + \mid x_{i,2} - y_{j,2} \mid\right)$$

— $\boldsymbol{x_i}$: point $\in$X $= \langle x_1, ..., x_m \rangle$, $1^{ère}$ série temporelle

— $\boldsymbol{y_j}$: point $\in$Y $= \langle y_1, ..., y_n \rangle$, 2^{e} série temporelle

Cette fonction de coût, à la base de la solution par programmation dynamique, peut être formulée de manière récursive de la façon suivante :

$$DTW(i,j) = d(x_i, y_j) + min \begin{cases} DTW(i-1, j-1) & \text{si les 2 points correspondent} \\ DTW(i-1, j) & \text{si ajouter le point } \nearrow \text{ correspondance} \\ DTW(i, j-1) & \text{si supprimer le point } \nearrow \text{ correspondance} \end{cases}$$

Où DTW est une matrice de coût sauvegardant les solutions intermédiaires permettant à la fonction de s'y référer à chaque demande de résolution d'un sous-problème déjà rencontré. Elle rend donc possible l'avancement ascendant vers la solution finale correspondant au coût total pour aligner les 2 séries. Cela, en échangeant du temps de calcul contre de la mémoire, afin de transformer une possible solution du problème en temps exponentiel en la solution en temps polynomial présentée dans le cadre de ce projet.

source : `https://www.youtube.com/watch?v=9GdbMc4CEhE`

1.2 Implémentation et complexité

Nous avons adopté une implémentation ascendante du Dynamic Time Wraping. Voici la complexité en temps et en espace de cette dernière :

	Espace	Temps
DynamicTimeWraping	$\Theta(N.M)$	$\Theta(N.M)$

TABLE 1 – Complexité en espace et en temps

Où N et M sont les longueurs des 2 séquences reçues en entrées. Ces complexités sont propres à notre implémentation, d'autres sont possibles avec des méthodes différentes.

2 NearestNeighbours

De multiples fonctions et opérations sont imbriquées dans NearestNeighbours. Vous trouverez dans chaque sous-section un résumé agrémenté d'un code couleur de leurs complexités en temps/espace qui sont toutes, quelque soit le cas, constantes :

2.1 Complexité en temps

— DynamicTimeWraping : $\Theta(q.l)$

— bpqReplaceMaximum : $\Theta(log(k))$

— bpqGetItems : $\Theta(k)$

— Boucle for de remplissage du tableau de type SketchDistance : $\Theta(k)$

(Opérations en $\Theta(1)$ volontairement omises)

1. **Pire cas**
   Si l'on néglige l'optimisation, ce qui donne une complexité constante pour DynamicTimeWraping, le pire cas correspond au cas où l'on trouve à chaque itération un croquis plus proche à ajouter dans la priority queue.

$$\begin{aligned}
&\Theta(k).\Theta(q.l) + \Theta(N-k).(\Theta(q.l) + \Theta(log(k))) + \Theta(k) + \Theta(k) \\
\Leftrightarrow &\Theta(N-k).(\Theta(q.l) + \Theta(log(k))) + \Theta(k).\Theta(q.l) \\
\Leftrightarrow &\Theta(N-k).\Theta(q.l) + \Theta(k).\Theta(q.l) \\
\Leftrightarrow &(\Theta(N-k) + \Theta(k)).\Theta(q.l) \\
\Leftrightarrow &\Theta(N).\Theta(q.l) \\
\Leftrightarrow &\Theta(N.q.l)
\end{aligned}$$

2. **Meilleur cas**
   Si l'on néglige l'optimisation, ce qui donne une complexité constante pour DynamicTimeWraping, le meilleur cas correspond au cas où les k croquis les plus proches sont directement trouvés. Cette situation évite la mise à jour de la priority queue après k itérations mais n'empêche pas la calcul des distances des autres croquis de la base de données.

$$\begin{aligned}
&\Theta(k).\Theta(q.l) + \Theta(N-k).\Theta(q.l) + \Theta(k) + \Theta(k) \\
\Leftrightarrow &\Theta(k).\Theta(q.l) + \Theta(N-k).\Theta(q.l) \\
\Leftrightarrow &(\Theta(N-k) + \Theta(k)).\Theta(q.l) \\
\Leftrightarrow &\Theta(N).\Theta(q.l) \\
\Leftrightarrow &\Theta(N.q.l)
\end{aligned}$$

2.2 Complexité en espace

— DynamicTimeWraping : $\Theta(q.l)$

— bpqCreate : $\Theta(k)$

— Tableau contenant les distances des croquis passant par la priority queue : $\Theta(N)$

— bpqGetItems : $\Theta(k)$

— Tableau de type SketchDistance : $\Theta(k)$

(Opérations en $\Theta(1)$ volontairement omises)

Si l'on néglige l'optimisation, ce qui donne une complexité constante pour DynamicTimeWraping, il n'y a pas de distinction entre pire et meilleur cas.

1. **Pire et meilleur cas**

$$\begin{aligned}
&\Theta(k) + \Theta(N) + \Theta(N.q.l) + \Theta(k) + \Theta(k) \\
\Leftrightarrow &\Theta(k + N + N.q.l + k + k) \\
\Leftrightarrow &\Theta(N.q.l)
\end{aligned}$$

3 Taux d'erreur

Valeur k	trainingset	trainingsetverylarge
1	17.40%	8.00%
3	16.60%	8.80%
7	18.60%	9.40%

TABLE 2 – Taux d'erreur pour différents k et 2 base de données

4 Temps de calculs

Valeur k	trainingset	trainingsetverylarge
1	4.906881	39.764447
3	5.722580	45.828936
7	6.424147	50.433319

TABLE 3 – Temps de calculs pour différents k et 2 base de données en secondes [s]

Ces résultats sont en accord avec la complexité théorique. Les paramètres q et l (correspondants respectivement au nombre de points du croquis à catégoriser et à la longueur des croquis de la base de données) étant supposés constants, on remarque la nette dépendance des temps de calculs vis à vis de N, le nombre d'échantillons de référence.

On observe également une dépendance des temps de calculs par rapport au paramètre k, le nombre de voisins. Celui-ci est présent dans le calculs de la complexité théorique mais pas dans le résultat final. Cela s'explique en remarquant l'impacte minime qu'a ce paramètre comparé à N :

— Quand N est multiplié par 10, les temps de calculs sont approximativement augmentés de 800%
— Quand k est multiplié par 7, les temps de calculs sont approximativement augmentés de 30% pour trainingset et de 25% pour trainingsetverylarge