Structures de données et algorithmes

Projet 3: Résolution de problèmes

2 mai 2022

On souhaite réaliser une application de recherche et de reconnaissance de croquis ("sketch" en anglais), c'est-à-dire des croquis réalisés rapidement à main levée. Pour ce faire, on a identifié dix classes d'intérêt et on dispose d'une base de données de référence [1] contenant 1000 croquis par classe (voir la figure 1 pour un exemple de croquis pour chacune des 10 classes).

A partir d'un croquis source (la requête), l'objectif est d'identifier le plus rapidement possible dans la base de données les k croquis "les plus proches" de cette requête selon une certaine mesure de distance. Les croquis ainsi retrouvés peuvent être ensuite utilisés pour faire une prédiction de la classe du croquis source, par exemple, en lui attribuant la classe majoritaire (*i.e.* la plus fréquente) parmi les k croquis retrouvés. Le processus est illustré sur un exemple à la figure 2.

1 Description de l'approche

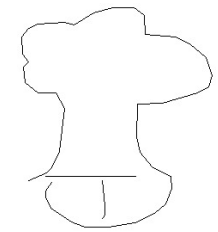
1.1 Représentation des croquis

Un croquis est représenté par un ensemble de *strokes*. Un *stroke* est une polyligne représentant un coup de crayon. Une polyligne est représentée par une succession de points formant des segments de droite. L'ordre des *strokes* et des points à l'intérieur d'un *strokes* suit l'ordre chronologique de tracé du croquis.

1.2 Dynamic time warping

Puisque le nombre de *strokes* et de segments par *stroke* varient d'un croquis à l'autre, on se propose de procéder en deux temps pour calculer la similarité entre deux croquis. D'abord, on concatène les *strokes* d'un croquis en une seule polyligne. On obtient alors une série temporelle décrivant l'évolution de la position du crayon. Ensuite, puisque ces séries ont des longueurs différentes, on propose de calculer la distance entre elles à l'aide d'un algorithme appelé *dynamic time warping (DTW)*. Cet algorithme générique calcule la distance entre deux séries temporelles en prenant en compte les variations en termes de vitesses et de longueurs entre les deux séries. Il semble donc tout à fait approprié pour prendre en compte les variabilités en termes de déplacements du crayon entre personnes.

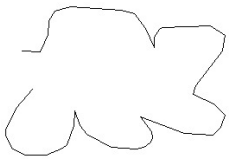
1. Ces croquis sont extraits de la base de données "Quick, Draw !" collectée par Google et disponible à l'adresse suivante : `https://github.com/googlecreativelab/quickdraw-dataset`.



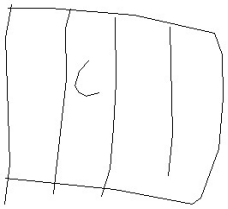
(a) Arbre (classe 0)

(b) Zigzag (classe 1)

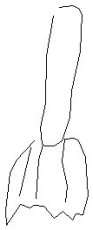
(c) Hamburger (classe 2)



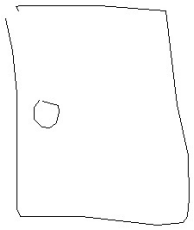
(d) Nuage (classe 3)



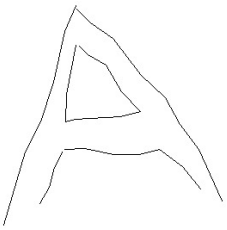
(e) Cellule (classe 4)



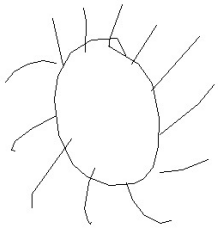
(f) Balais (classe 5)



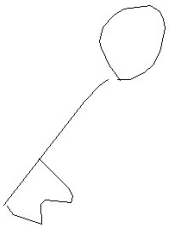
(g) Porte (classe 6)



(h) Tour Eiffel (classe 7)

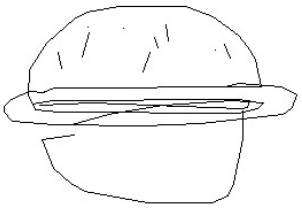


(i) Soleil (classe 8)



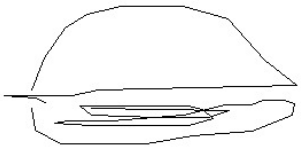
(j) Clé (classe 9)

FIGURE 1 – Exemples de croquis

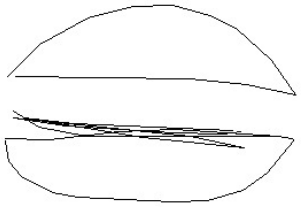


(a) Requête

(b) Plus proche voisin



(c) Deuxième plus proche voisin



(d) Troisième plus proche voisin

FIGURE 2 – Requête et plus proches voisins

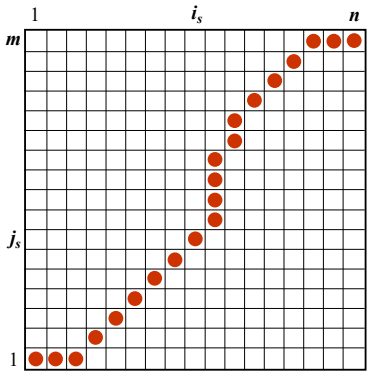


FIGURE 3 – Représentation d'un chemin considéré par le dynamic time warping.

L'idée de l'algorithme DTW est de rechercher l'appariement des points des deux séries temporelles (respectant l'ordre temporelles de ces points) qui minimisent la somme des distances entre points appariés. Plus formellement, soient deux séquences de points $X = \langle x_1, \ldots, x_m \rangle$ et $Y = \langle y_1, \ldots, y_n \rangle$. La distance DTW entre ces deux séquences est définie par :

$$DTW(X,Y) = \min_{\langle (i_1,j_1),\ldots,(i_k,j_k)\rangle \in \mathcal{P}(X,Y)} \sum_{l=1}^{k} d(x_{i_l}, y_{j_l}) \tag{1}$$

où d mesure la distance entre deux points et $\mathcal{P}(X,Y)$ est l'ensemble des séquences de paires d'indices $\langle (i_1,j_1),\ldots,(i_k,j_k)\rangle$ telles que :

- $(i_1,j_1) = (1,1)$, $(i_k,j_k) = (m,n)$,
- $0 \leq i_s - i_{s-1} \leq 1$ et $0 \leq j_s - j_{s-1} \leq 1$,
- $i_s > i_{s-1}$ ou $j_s > j_{s-1}$.

Sur une grille discrète de taille $m \times n$, les séquences de $\mathcal{P}$ sont donc tous les chemins allant du point $(1,1)$ au point (m,n) en se déplaçant d'une case maximum, soit vers la droite, vers la haut, ou en diagonale (voir figure 3). Bien que le nombre de chemins soit très important, on vous demande dans ce projet d'implémenter le calcul de la distance (1) de manière efficace en vous basant sur la programmation dynamique.

Dans le cadre de ce projet, nous utiliserons comme mesure de distance entre deux points $x_i = (x_{i,1}, x_{i,2})$ et $y_j = (y_{j,1}, y_{j,2})$ la distance absolue moyenne :

$$d(x_i, y_j) = \frac{1}{2}\Big(|x_{i,1} - y_{j,1}| + |x_{i,2} - y_{j,2}|\Big) \tag{2}$$

1.3 Recherche des k croquis les plus proches

La détermination des k croquis les plus proches d'un croquis source, noté Q, dans la base de données de référence, notée LS (pour "Learning Set"), sera effectuée par une fonction NEARESTNEIGHBOURS(LS, Q, K). Cette fonction devra calculer les distances selon l'algorithme DTW entre le croquis Q et successivement tous les croquis de l'ensemble LS. Afin de maintenir efficacement les k croquis les plus proches lors du parcours de la base de données de référence, nous vous demandons d'utiliser une file à priorité dont la capacité sera limitée à k éléments maximum et utilisant la distance à l'exemple Q comme valeur de priorité. Lors du parcours de la base de données de référence, un croquis sera ajouté à la file, soit si cette file contient moins que k croquis, soit si ce croquis est plus proche de Q que le croquis de la file le plus éloigné de Q. Dans ce dernier cas, le nouveau croquis devra remplacer dans la file le croquis le plus éloigné de Q.

1.4 Optimisation

Lorsque l'ensemble LS est grand, la recherche des k plus proches croquis d'une requête peut être relativement lente. Dans le cas où la file contient déjà k croquis, il est possible d'éviter d'avoir à effectuer tout le calcul de la distance DTW pour un nouveau croquis en prenant en compte la distance maximale courante, notée d_{max}, entre un croquis de la file et la requête. L'algorithme DTW calcule en effet une matrice de coût colonne par colonne ou ligne par ligne. Dès que le minimum d'une colonne ou d'une ligne [2] est supérieure à la distance d_{max}, on a la garantie que la distance DTW ne pourra pas être inférieure à d_{max} et il est alors possible d'arrêter prématurément le calcul de la distance. Pour implémenter cette optimisation, la fonction de calcul de la distance DTW prendra comme argument, en plus des deux séquences, la distance maximale d_{max} et elle pourra interrompre son calcul et renvoyer une valeur $+\infty$ [3] dès que la distance d_{max} sera dépassée.

2 Implémentation

Les fichiers suivants vous sont fournis :

— `Sketch.h/c` : une librairie définissant les structures utilisées pour représenter les croquis et les bases de données et implémentant des fonctions permettant de charger en mémoire un ensemble de croquis à partir d'un fichier, d'afficher leur classe et de générer une image (au format ppm) du croquis. Cette dernière fonction utilise une librairie graphique (fichiers `easyppm.h/c`) également fournie.

— `main.c` : un fichier main d'exemple utilisant les fonctions à implémenter pour retrouver les k plus proches croquis d'une requête ou pour calculer le taux d'erreur sur un ensemble de croquis de test.

— `Makefile` : un Makefile pour compiler et exécuter le code.

Les différents fichiers à fournir sont les suivants :

— `DynamicTimeWarping.c` contenant l'implémentation du *dynamic time warping* en répondant à l'interface définie dans `DynamicTimeWarping.h`. Votre implémentation doit être basée sur la programmation dynamique et exploiter l'optimisation décrite dans la section 1.4.

— `BoundedPriorityQueue.c` regroupant les différentes fonctions liées à la file à priorité dont l'interface est précisée dans le fichier `BoundedPriorityQueue.h`.

— `NearestNeighbours.c` implémentant la fonction de recherche des k croquis les plus proches selon l'interface définie dans `NearestNeighbours.h`.

En outre, nous vous fournissons les bases de données suivantes :

— `testset.txt` la base de données de test.

— `trainingsetverylarge.txt` la base de données de référence contenant 1000 exemples par classe.

— `trainingset.txt` un sous-ensemble de la base de données de référence contenant 100 exemples par classe.

2. auquel on pourra ajouter la distance entre les deux derniers points des deux séries temporelles, sauf pour la dernière ligne/colonne.

3. La constante `INFINITY` définie dans `math.h`.

3 Rapport

Répondez aux questions suivantes dans votre rapport :

1. Dans le cadre du Dynamic Time Warping, définissez la fonction de coût qu'on cherche à minimiser en précisant les paramètres dont elle dépend. Donnez ensuite la formulation récursive de cette fonction de coût à la base de la solution par programmation dynamique. Si vous consultez certaines sources pour dériver cette formulation, citez les.
2. Précisez quelle implémentation (ascendante ou descendante) vous avez adoptée et donnez la complexité en temps et en espace de cette implémentation en fonction des longueurs m et n des séquences comparées. Vous pouvez négliger l'optimisation de la section 1.4 pour cette analyse.
3. Donnez et justifiez (brièvement) la complexité de la fonction NearestNeighbours que vous avez implémentée au pire et au meilleur cas en fonction de $q = |Q|$, le nombre de points du croquis Q, de $N = |LS|$, le nombre d'échantillons de référence, de l, la longueur des croquis de LS (qu'on supposera constante) et de k, le nombre de voisins. Vous pouvez également négliger l'optimisation de la section 1.4 pour cette analyse.
4. Pour $k \in \{1, 3, 7\}$, calculez le taux d'erreur de reconnaissance des échantillons de tests en utilisant les deux bases de données de référence fournies. Le taux d'erreur est la fraction du nombre de fois où la vraie classe d'un croquis de l'ensemble de test ne correspond pas à la classe majoritaire parmis les k croquis les plus proches de l'ensemble de référence.
5. Reportez dans le rapport les temps nécessaires au calcul de ces taux d'erreur en fonction de k. Précisez s'ils sont en accord avec la complexité théorique.

4 Deadline et soumission

Le projet est à réaliser **par groupe de deux maximum** pour le vendredi **20 mai 2022, 23h59** au plus tard. Le rapport et les fichiers de code (`DynamicTimeWarping.c`, `NearestNeighbours.c` et `BoundedPriorityQueue.c`) doivent être soumis via deux projets séparés sur Gradescope.

Vos fichiers seront évalués avec les flags habituels (`--std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes - DNDEBUG`) sur les machines ms8xx. Respectez bien les noms des fichiers et n'incluez aucun fichier supplémentaires.

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat[4] avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur Ecampus.

Bon travail !

4. Des tests anti-plagiat seront réalisés.