



École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5^e année
2013 - 2014

Rapport de projet

**Matching élastique et robuste de
séquences**

Encadrants

Nicolas RAGOT
nicolas.ragot@univ-tours.fr

Université François-Rabelais, Tours

Etudiants

Abourahman ADEN HASSAN
abdourahman.adenhassan@etu.univ-tours.fr

DI5 2013 - 2014

Version du 3 février 2014

Table des matières

Rapport de projet	7
2 Introduction	7
3 Remerciements	8
4 Projet	9
4.0.1 Objectifs	9
4.0.2 Planning	10
4.1 Caractéristiques du projet	11
5 Séquence	12
5.0.1 Exemples de séquences	12
6 Etude Algorithmique	13
6.0.2 Levensthein : description	13
6.0.3 algorithme	14
6.0.4 Complexité	15
6.0.5 Exemple	16
6.0.6 LCS : description	17
6.0.7 algorithme	17
6.0.8 Complexité	18
6.0.9 Exemple	18
6.0.10 Dynamic Time Warping : description	20
6.0.11 algorithme	20
6.0.12 Complexité	21
6.0.13 Exemple	22
6.0.14 Minimum Variance Matching : description et exemple	23
6.0.15 algorithme	25
6.0.16 Complexité	26
7 Etude de Contraintes	27
7.0.17 Mots	27
7.0.18 Signatures	27
7.0.19 Image de mots	27
7.0.20 Résultats et remise en cause	28
8 Conception et développement	29
8.1 Structure de données et diagramme de classe	29
8.2 Développement et test	31
8.2.1 Formattage des fichiers d'entrée	31
8.2.2 Spécificités de développement	31
9 Resultat	35
9.1 Deploiement	35
9.2 Ligne de commande	35
9.2.1 Exemple 1	35
9.2.2 Exemple 2	37
9.3 GUI	37



10 Problèmes rencontrés et solutions apportées	40
10.1 Contraintes	40
10.2 Portabilité	41
11 Conclusion	42
12 Bibliographie	43

Table des figures

4.1	Diagramme de cas d'utilisation	9
4.2	Planning prévisionnel	10
4.3	Planning effectif	10
7.1	Image de mots	27
8.1	Modèle	29
8.2	Package de calcul	30
8.3	Diagramme de composant	30
8.4	Format de fichier d'entrée	31
8.5	Tests unitaires	32
8.6	Format de l'aide	33
8.7	Patron de correspondance	33
8.8	GUI	34
8.9	Choix d'affichages possibles selon l'algorithme et le type de séquence	34
9.1	Fichier 1 sous forme CSV	35
9.2	Fichier 2 sous forme CSV	36
9.3	Résultat levensthein ligne de commande	36
9.4	Résultat LCS ligne de commande	36
9.5	Résultat DTW ligne de commande	37
9.6	Résultat DTW ligne de commande	37
9.7	Résultat séquences DTW/MVM	38
9.8	Résultat séquences DTW/MVM	39
10.1	Futur package de projet	40

Liste des tableaux

6.1	tableau de valeurs "tabValue"	18
6.2	tableau de correspondance 1 "tabCorrespondance1"	18
6.3	tableau de correspondance 1 "tabCorrespondance2"	18
6.4	tableau de valeurs "tabValue"	18
6.5	tableau de correspondance 1 "tabCorrespondance1"	18
6.6	tableau de correspondance 1 "tabCorrespondance2"	19

Introduction

Ce document est le rapport de projet concernant le projet de fin d'études sur le sujet "Matching élastique et robuste de séquence". L'objectif de ce document est de décrire le contexte dans lequel le projet intervient, les choix effectués sur ce projet et la manière avec laquelle nous les avons réalisés.

Dans le cadre d'un projet, nous avons été sollicités pour mettre en place une solution logicielle qui calcule la distance entre des séquences. La nature des séquences est variable : chaîne de caractères, signatures, image de mots en priorité, séquences de protéines, séries boursières.

La solution obtenue est sous la forme d'une toolbox. Cette toolbox prend en entrée différentes séquences en entrée et effectue la correspondance.

Comment ce projet a-t-il été conduit, quels étaient les objectifs initiaux et quel est le résultat ? Nous vous expliquerons tous ces points dans ce rapport de projet.

Les acteurs mis en jeu au cours du projet sont :

MOE : Abdourahman Aden Hassan

MOA : Nicolas Ragot

Remerciements

Je tiens à remercier pour mon projet de fin d'étude :

Nicolas Ragot, Maître de Conférences, qui m'a offert la possibilité d'effectuer ce projet passionnant.

Hubert Cardot, pour sa disponibilité concernant les séquences boursières.

Christophe Lenté, qui a pu m'offrir un planning adapté pour le projet.

Ameur Soukhal, pour m'avoir aiguillé sur les points à améliorer lors de la soutenance mi-parcours.

Je remercie également les autres personnes du laboratoire d'informatique pour la bonne ambiance.

Projet

Ce projet est un projet scientifique d'étude et développement. Sur une première partie du projet, nous étudions plusieurs types de données (mots, signatures, images...). Ces données doivent être traduites en séquence. La problématique est de définir un outil suffisamment complet pour pouvoir traiter des flux très différents.

4.0.1 Objectifs

L'objectif de ce projet est de créer une boîte à outils qui calcule la distance entre des séquences. Une étude des différentes séquences et spécificités a été effectuée. A partir des spécificités on étudiera les algorithmes existants. Ces algorithmes devaient être intégrés dans notre boîte à outils voir adaptés pour pouvoir répondre aux différents contextes applicatifs.

Dans un premier temps (l'essentiel) l'objectif a été de concevoir la toolbox. Puis à partir de cette boîte nous devions essayer d'en faire un algorithme général. Cet algorithme unique gèrerait les différents flux et leurs contraintes à la fois. Ainsi dans l'idéal il fera tout ce que fait la toolbox. Si l'on remplace les algorithmes existants cela ne remet pas en cause notre application elle devra fonctionner directement suite à ce remplacement.

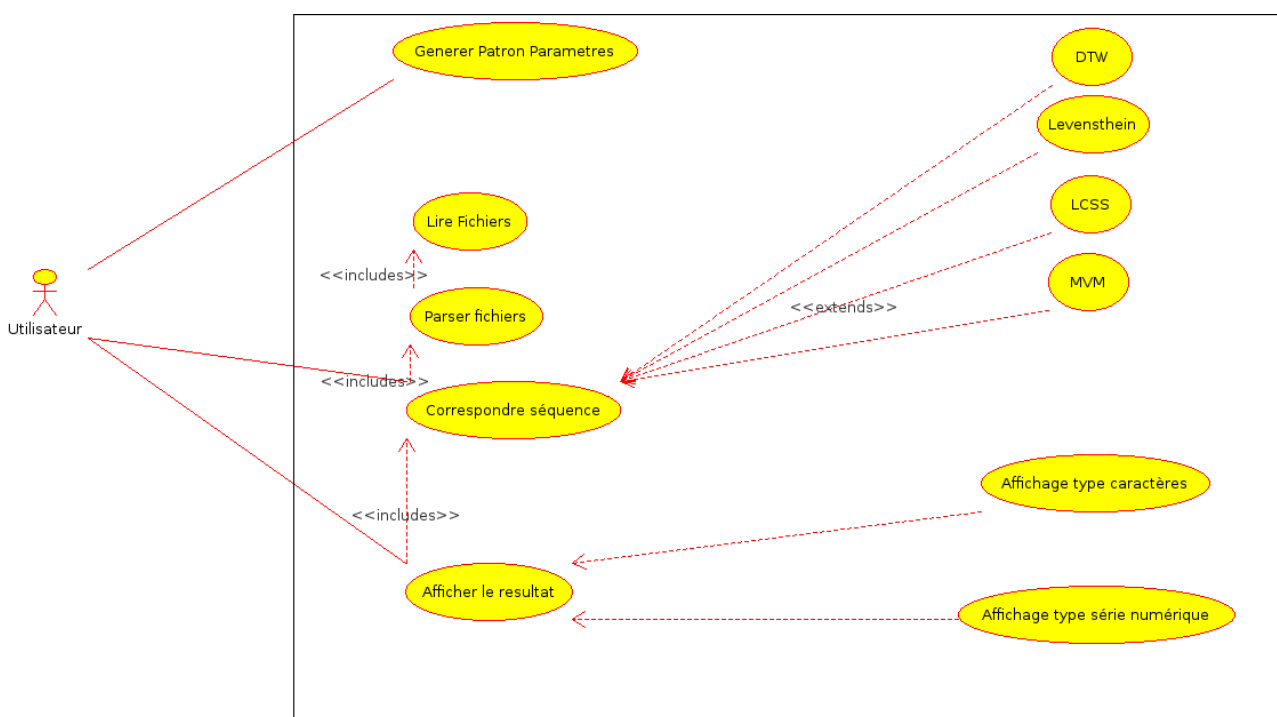


FIGURE 4.1 – Diagramme de cas d'utilisation

4.0.2 Planning

Nous avons utilisé un cycle en V pour la gestion du projet. Le projet a été divisé en lots, ceux-ci sont découpés en tâches. Chaque tâche aura une estimation de durée. Cette estimation sera ensuite corrigée par les résultats réels. Certains outils nous seront nécessaire pour bien modéliser la structure du logiciel comme l'UML pour la création du diagramme de classe pour les structures de données. Nous avons séparé notre projet en 5 lots.

- **Lot 1** : Découverte du sujet et étude documentaire sur les algorithmes
- **Lot 2** : Etude des contraintes
- **Lot 3** : Etude des algorithmes par rapport aux contraintes
- **Lot 4** : Conception de la structure de données, développement et tests
- **Lot 5** : Rapport

Voici le planning prévisionnel prévu lors de la spécification. Le projet s'est déroulé légèrement diéremment du planning prévisionnel, ce qui a conduit à un planning eectif, mis à jour jusqu'à la n du projet.

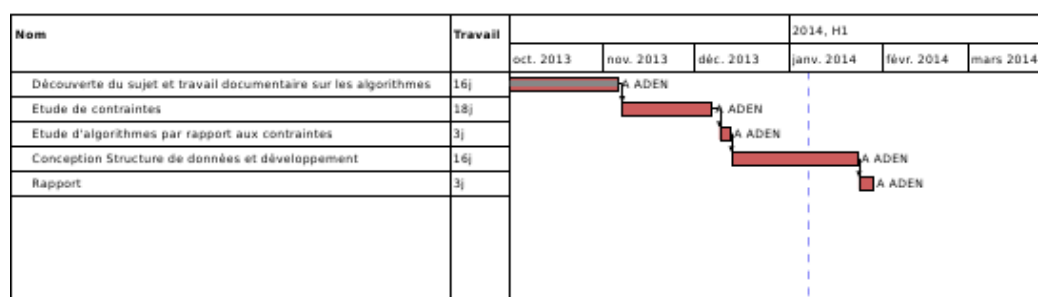


FIGURE 4.2 – Planning prévisionnel

Voici le planning effectif du projet. Certaines tâches ont été abandonnées (surtout la partie algorithmes par rapport aux contraintes). D'autres ont été créés, la partie graphique est devenue une fonctionnalité importante du projet.

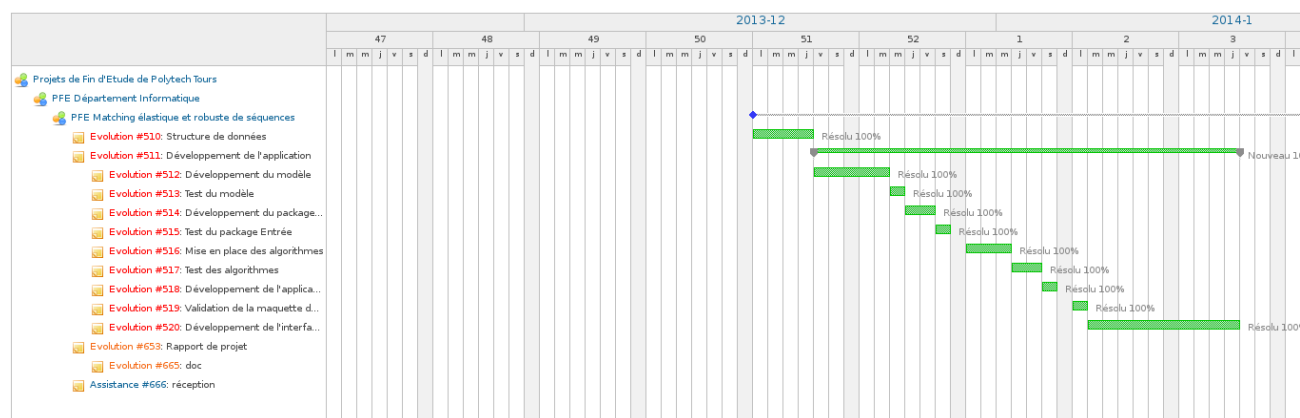


FIGURE 4.3 – Planning effectif

4.1 Caracteristiques du projet

L'autre outil de gestion de projet a été Redmine.

Toutes les tâches y étaient placées, avec leur durée respective. Si besoin certaines prévisions étaient remises en cause. Voici les autres caracteristiques du projet.

- Langage de programmation adopté : le langage C++ Standard
- Normes de documentation : Doxygen
- Dépôt des documents : SVN
- Cahier de Spécification à rendre avant Fin novembre 2013.
- Projet à réaliser avant le 27/01/2014 : 54 jours
- Documentation de développeur à effectuer
- Partie graphique avec QT
- Campagne de test à effectuer pour juger de l'efficacité de la méthode
- Architecture logicielle permettant l'ajout de nouveaux logiciels

Séquence

La séquence est une suite de données ordonnée. Ces données peuvent être des caractères, des chiffres ou encore des vecteurs numériques.

On peut comparer des séquences entre elles à condition que les éléments soient de même type. Les vecteurs de caractéristiques sont de même taille par la correspondance.

5.0.1 Exemples de séquences

Exemple de séquence de type caractères : $(a, b, c, v, c, d, r, r, e, a, z, e, d, d, s, s)$

Exemple de séquence de type numérique : $(1, 12, 2, 5, 8, 7, 8, 1, 54, 5, 7, 7)$

Exemple de séquence de type vecteur : $\left(\begin{pmatrix} x = 1 \\ y = 2 \\ z = 3 \end{pmatrix}, \begin{pmatrix} x = 8 \\ y = 7 \\ z = 4 \end{pmatrix}, \begin{pmatrix} x = 0 \\ y = 4 \\ z = 3 \end{pmatrix}, \begin{pmatrix} x = 1 \\ y = 2 \\ z = 8 \end{pmatrix} \right)$

La dimension de ce vecteur est de 3.

Soit A une séquence de départ et B une séquence d'arrivée.

Nous posons qu'une correspondance f est une fonction injective c'est à dire que pour tout élément i de A il existe au plus 1 élément j de B lié à i .

Une correspondance $f : A \rightarrow B$ se définira ainsi : $\forall j \in B, \forall i \in A, f(i) = j \Rightarrow \exists k \in A \text{ t.q. } f(k) = j$

La distance de la correspondance est notée $d_{f_{ij}}$ ou d_f

Une solution S est un ensemble de correspondances. Soit p la taille de S, la distance de la solution créée s'exprime ainsi :

$$d_S = \sqrt{\sum_{k=1}^p d_{S_k}}$$

Nous pouvons définir des contraintes sur les séquences. Ces contraintes permettent de donner du sens à nos correspondances. Selon le type de flux, certaines contraintes sont importantes pour gérer des caractéristiques propres à ces flux.

Par exemple on peut créer la contrainte c_1 qui veut que l'on compare seulement un élément sur 2.

Etude Algorithmique

Dans cette étape du projet nous avons étudié 4 algorithmes :

- Levensthein
- LCS (Longest Common Subsequence)
- DTW (Dynamic Time Warping)
- MVM (Minimum Variance Matching)

Cette étude a permis de comprendre les outils algorithmiques qui nous permettront de créer notre toolbox et de manipuler les séquences.

Ce lot a duré **16** jours.

6.0.2 Levensthein : description

Cet algorithme permet d'effectuer un matching entre deux séquences. Il s'appuie sur la programmation dynamique c'est à dire que la solution optimale trouvée pour une taille donnée s'appuie sur des solutions optimales de tailles intermédiaires.

L'algorithme s'appuie sur cette formule de récurrence :

$$\begin{aligned}d_{0,0} &= 0 \\d_{i,0} &= d_{i-1,0} + \text{deleteCost} \\d_{0,j} &= d_{0,j-1} + \text{insertCost} \\d_{i,j} &= \min \begin{cases} d_{i-1,j} + \text{deleteCost} \\ d_{i,j-1} + \text{insertCost} \\ 0 & s_i = t_j \\ d_{i-1,j-1} + \text{substCost} & s_i \neq t_j \end{cases}\end{aligned}$$

Dans l'algorithme nous aurons deux tableaux. Un tableau contenant les distances à chaque itération. Un autre tableau contenant les choix effectués à chaque itération.

L'algorithme se fait en deux étapes. D'abord le remplissage des tableaux de distance et de correspondance. Puis la recherche d'une solution par backtrack.

6.0.3 algorithme

Remplissage

```

input : s1  $\leftarrow$  Sequence , tailleS1  $\leftarrow$  entier , s2  $\leftarrow$  Sequence , tailleS2  $\leftarrow$  entier, addCost  $\leftarrow$  flottant,
        delCost  $\leftarrow$  flottant, subCost  $\leftarrow$  flottant
output: vecValue  $\leftarrow$  vecteur[tailleS1+1][tailleS2+1] de flottants, vecCorrespondance  $\leftarrow$ 
        vecteur[tailleS1+1][tailleS2+1] d'entiers

    Création des tableaux;
    Création des constantes;
    HAUT  $\leftarrow$  1;
    GAUCHE  $\leftarrow$  2;
    DIAGONALE  $\leftarrow$  4;
    Initialisation des variables;
    vecValue[0][0]  $\leftarrow$  0;
    vecCorrespondance[0][0]  $\leftarrow$  0;
    it  $\leftarrow$  1;
    while it < tailleS1+1 do
        | vecValue[it][0]  $\leftarrow$  it*addcost;
        | vecCorrespondance[it][0]  $\leftarrow$  HAUT;
        | it  $\leftarrow$  it + 1;
    end
    it  $\leftarrow$  1;
    while it < tailleS2+1 do
        | vecValue[0][it]  $\leftarrow$  it*delcost;
        | vecCorrespondance[0][it]  $\leftarrow$  GAUCHE;
        | it  $\leftarrow$  it + 1;
    end
    it1  $\leftarrow$  1;
    it2  $\leftarrow$  1;
    while it < tailleS1+1 do
        | while it < tailleS2+1 do
            | vecCorrespondance[it][it2]  $\leftarrow$  0;
            | distance  $\leftarrow$  distanceSequences(s1,it-1,s2,it2-1);
            | diag  $\leftarrow$  0;
            | if distance != 0 then
                | | diag  $\leftarrow$  vecValue[it-1][it2-1] + transCost;
            | end
            | minVal  $\leftarrow$  min(vecValue[it][it2-1] + delcost, vecValue[it-1][it2] + addcost, diag);
            | vecValue[it][it2]  $\leftarrow$  minVal;
            | it2  $\leftarrow$  it2 + 1;
            | if vecValue[it][it2-1] + delcost = minVal then
                | | vecCorrespondance[it][it2]  $\leftarrow$  vecCorrespondance[it][it2] + HAUT;
            | end
            | if vecValue[it-1][it2] + addcost = minVal then
                | | vecCorrespondance[it][it2]  $\leftarrow$  vecCorrespondance[it][it2] + GAUCHE;
            | end
            | if diag = minVal then
                | | vecCorrespondance[it][it2]  $\leftarrow$  vecCorrespondance[it][it2] + DIAGONALE;
            | end
        | end
        | it2  $\leftarrow$  0;
        | it  $\leftarrow$  it + 1;
    end

```

Algorithm 1: Levenstein

Backtrack

```

input : vecValue  $\leftarrow$  vecteur[tailleS1+1][tailles2+1] de flottants, vecCorrespondance  $\leftarrow$ 
    vecteur[tailleS1+1][tailles2+1] d'entiers, tailleS1  $\leftarrow$  entier, tailleS2  $\leftarrow$  entier,
output: listCorrespondance  $\leftarrow$  Liste de correspondances

it1  $\leftarrow$  tailleS1;
it2  $\leftarrow$  tailleS2;
HAUT  $\leftarrow$  1;
GAUCHE  $\leftarrow$  2;
DIAGONALE  $\leftarrow$  4;
while  $!(it1 < 0 \vee it2 < 0)$  do
    valMin  $\leftarrow$   $+\infty$ ;
    if  $it = 0$  then
        Ajout correspondance de type suppression pour la séquence 2 à l'indice it2 ;
        it2  $\leftarrow$  it2 -1;
    end
    else if  $it2 = 0$  then
        Ajout correspondance de type ajout pour la séquence 1 à l'indice it ;
        it  $\leftarrow$  it -1;
    end
    else
        valMin  $\leftarrow$  min(vecValue[it-1][it2-1],vecValue[it][it2-1],vecValue[it-1][it2]);
        Traitement diagonal;
        if  $valMin = vecValue[it-1][it2-1] \vee valCorrespondance[it][it2] \geq DIAGONALE$  then
            Ajout correspondance de type substitution séquence 1(it) par séquence 2(it2) ;
            it2  $\leftarrow$  it2 -1;
            it  $\leftarrow$  it -1;
        end
        Traitement gauche;
        if  $valMin = vecValue[it-1][it2] \vee valCorrespondance[it][it2] - DIAGONALE \geq GAUCHE$  then
            Ajout correspondance de type ajout séquence 1(it) ;
            it  $\leftarrow$  it -1;
        end
        Traitement haut;
        if  $valMin = vecValue[it][it2-1] \vee valCorrespondance[it][it2-1] - DIAGONALE - GAUCHE \geq HAUT$ 
            then
                Ajout correspondance de type suppression séquence 2(it2) ;
                it2  $\leftarrow$  it2 -1;
            end
        end
    end
end

```

Algorithm 2: Backtrack Levensthein

6.0.4 Complexité

Soit S_1 la première séquence S_2 la seconde séquence, V la dimension des éléments dans les séquences, la complexité du remplissage est de $CARD(S_1) * CARD(S_2) * V$

Soit O la complexité du backtrack. On peut noter O avec cette inégalité :

$$\sqrt{CARD(S_1)^2 + CARD(S_2)^2} \leq O \leq CARD(S_1) + CARD(S_2)$$

6.0.5 Exemple

Nous allons appliquer l'algorithme avec les séquences "LEVENSTHEIN" et "MELENSTHEIN"

Initialisation

-	-	m	e	i	l	e	n	s	t	e	i	n
-	0	1	2	3	4	5	6	7	8	9	10	11
l	1	-	-	-	-	-	-	-	-	-	-	-
e	2	-	-	-	-	-	-	-	-	-	-	-
v	3	-	-	-	-	-	-	-	-	-	-	-
e	4	-	-	-	-	-	-	-	-	-	-	-
n	5	-	-	-	-	-	-	-	-	-	-	-
s	6	-	-	-	-	-	-	-	-	-	-	-
h	7	-	-	-	-	-	-	-	-	-	-	-
t	8	-	-	-	-	-	-	-	-	-	-	-
e	9	-	-	-	-	-	-	-	-	-	-	-
i	10	-	-	-	-	-	-	-	-	-	-	-
n	11	-	-	-	-	-	-	-	-	-	-	-

Remplissage et parcours

-	-	m	e	i	l	e	n	s	t	e	i	n
-	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

En gras vous pouvez observer les chemins possibles.

Un des chemins donne cette solution :

```
l e v e n s h t e i n
o = + o = = = . = = = =
m e i l e n s t e i n
```


6.0.6 LCS : description

Cet algorithme permet de trouver la plus longue sous séquence entre deux séquences. Cette sous séquence n'est pas forcément contiguë. Par exemple nous avons la séquence aba et la séquence tata, la plus longue sous-séquence commune à ces deux séquences est a,a. Il s'appuie sur la programmation dynamique c'est à dire que la solution optimale trouvée pour une taille donnée s'appuie sur des solutions optimales de tailles intermédiaires.

L'algorithme s'appuie sur cette formule de récurrence :

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } x_i = y_j \\ \text{longest}(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

Dans l'algorithme nous aurons un deux tableau. Un tableau contenant la plus longue sous séquence à chaque itération. Un autre tableau contenant une des solutions à chaque itération.

L'algorithme se fait en une seule étape, sans backtrack cette fois : on remplit le tableau. La solution de l'algorithme se trouve à la position (tailleS1,tailleS2) des tableaux.

6.0.7 algorithme

```

input : s1 ← Sequence , tailleS1 ← entier , s2 ← Sequence , tailleS2 ← entier
output: vecValue ← vecteur[tailleS1+1][tailleS2+1] de flottants, vecCorrespondance1 ←
    vecteur[tailleS1+1][tailleS2+1][tailleS1] d'entiers, vecCorrespondance2 ←
    vecteur[tailleS1+1][tailleS2+1][tailleS2] d'entiers

Création et initialisation des tableaux vecValue, vecCorrespondance1 et vecCorrespondance2 à 0;
it1 ← 1; it2 ← 1;
while it < tailleS1+1 do
    while it < tailleS2+1 do
        if distanceSequences(s1,it-1,s2,it2-1) then
            vecValue[it][it2] ← vecValue[it-1][it2-1];
            Copie de tableaux : le contenu du tableau est copié dans l'autre tableau (expression simplifiée ici);
            vecCorrespondance1[it][it2] ← vecCorrespondance1[it-1][it2-1];
            vecCorrespondance2[it][it2] ← vecCorrespondance2[it-1][it2-1];
        end
        else
            if vecValue[it][it2-1] > vecValue[it-1][it2] then
                vecValue[it][it2] ← vecValue[it][it2-1];
                Copie de tableaux;
                vecCorrespondance1[it][it2] ← vecCorrespondance1[it][it2-1];
                vecCorrespondance2[it][it2] ← vecCorrespondance2[it][it2-1];
            end
            else
                vecValue[it][it2] ← vecValue[it-1][it2];
                Copie de tableaux;
                vecCorrespondance1[it][it2] ← vecCorrespondance1[it-1][it2];
                vecCorrespondance2[it][it2] ← vecCorrespondance2[it-1][it2];
            end
        end
        it2 ← it2 + 1;
    end
    it2 ← 1; it ← it + 1;
end

```

Algorithm 3: LCS

6.0.8 Complexité

Soit S_1 la première séquence S_2 la seconde séquence, V la dimension des éléments présents dans les séquences, la complexité du remplissage est de :

$$\text{CARD}(S_1) * \text{CARD}(S_2) * (\text{CARD}(S_1) + \text{CARD}(S_2)) * V$$

6.0.9 Exemple

Nous allons appliquer l'algorithme avec les séquences "GAC" et "AGCAT"

Initialisation

-	-	A	G	C	A	T
-	0	0	0	0	0	0
G	0	0	0	0	0	0
A	0	0	0	0	0	0
C	0	0	0	0	0	0

TABLE 6.1 – tableau de valeurs "tabValue"

-	-	A	G	C	A	T
-	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}
G	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}
A	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}
C	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}

TABLE 6.2 – tableau de correspondance 1 "tabCorrespondance1"

-	-	A	G	C	A	T
-	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}
G	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}
A	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}
C	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}

TABLE 6.3 – tableau de correspondance 1 "tabCorrespondance2"

Remplissage

-	-	A	G	C	A	T
-	0	0	0	0	0	0
G	0	0	1	1	1	1
A	0	1	1	1	2	2
C	0	1	1	2	2	2

TABLE 6.4 – tableau de valeurs "tabValue"

-	-	A	G	C	A	T
-	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}	{0, 0, 0}
G	{0, 0, 0}	{0, 0, 0}	{1, 0, 0}	{1, 0, 0}	{1, 0, 0}	{1, 0, 0}
A	{0, 0, 0}	{0, 1, 0}	{0, 1, 0}	{0, 1, 0}	{1, 1, 0}	{1, 1, 0}
C	{0, 0, 0}	{0, 1, 0}	{0, 1, 0}	{0, 1, 1}	{1, 1, 0}	{1, 1, 0}

TABLE 6.5 – tableau de correspondance 1 "tabCorrespondance1"

-	-	A	G	C	A	T
-	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}
G	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 1, 0, 0, 0}	{0, 1, 0, 0, 0}	{0, 1, 0, 0, 0}	{0, 1, 0, 0, 0}
A	{0, 0, 0, 0, 0}	{1, 0, 0, 0, 0}	{1, 0, 0, 0, 0}	{1, 0, 0, 0, 0}	{0, 1, 0, 1, 0}	{0, 1, 0, 1, 0}
C	{0, 0, 0, 0, 0}	{1, 0, 0, 0, 0}	{1, 0, 0, 0, 0}	{1, 0, 1, 0, 0}	{0, 1, 0, 1, 0}	{0, 1, 0, 1, 0}

TABLE 6.6 – tableau de correspondance 1 "tabCorrespondance2"

Une de solution de l'algorithme est donc s de taille 2 $\{g, a\}$

6.0.10 Dynamic Time Warping : description

Cet algorithme permet d'effectuer un matching entre deux séquences. Le plus souvent ce sont des séquences numériques. Il s'appuie sur la programmation dynamique c'est à dire que la solution optimale trouvée pour une taille donnée s'appuie sur des solutions optimales de tailles intermédiaires.

L'algorithme s'appuie sur cette formule de récurrence :

$$DTW(X_i, Y_j) = \begin{cases} 0 & \text{si } i = 0 \cap j = 0 \\ +\infty & \text{si } (i = 0 \cap j > 0) \cup (j = 0 \cap i > 0) \\ d_{X_i, Y_j} + \min(DTW(X_{i-1}, Y_{j-1}), DTW(X_i, Y_{j-1}), DTW(X_{i-1}, Y_j)) & \text{sinon.} \end{cases}$$

Dans l'algorithme nous aurons un tableau contenant les distances à chaque itération. L'algorithme se fait en deux étapes. D'abord le remplissage des tableaux de distance et de correspondance. Puis la recherche d'une solution par backtrack.

6.0.11 algorithme

Remplissage

```

input : s1 ← Sequence , tailleS1 ← entier , s2 ← Sequence , tailleS2 ← entier
output: vecValue ← vecteur[tailleS1+1][tailleS2+1] de flottants

Création des tableaux;
Initialisation des variables;
vecValue[0][0] ← 0;
it ← 1;
while it < tailleS1+1 do
    | vecValue[it][0] ← +∞;
    | it ← it + 1;
end
it ← 1;
while it < tailleS2+1 do
    | vecValue[0][it] ← +∞;
    | it ← it + 1;
end
it1 ← 1;
it2 ← 1;
while it < tailleS1+1 do
    | while it < tailleS2+1 do
        | vecValue[it][it2] ←
        | dXit1, Yit2 + min(vecValue[it1-1][it2-1], vecValue[it1][it2-1], vecValue[it1-1][it2])
        | it2 ← it2 + 1;
    | end
    | it2 ← 1;
    | it ← it + 1;
end

```

Algorithm 4: DTW

Backtrack

```

input : vecValue  $\leftarrow$  vecteur[tailleS1+1][tailles2+1] de flottants, tailleS1  $\leftarrow$  entier, tailleS2  $\leftarrow$  entier,
output: listCorrespondance  $\leftarrow$  Liste de correspondances

it1  $\leftarrow$  tailleS1;
it2  $\leftarrow$  tailleS2;
while  $!(it1 < > 0 \quad it2 < > 0)$  do
    valMin  $\leftarrow$   $+\infty$ ;
    if  $it = 0$  then
        | Ajout correspondance de type suppression pour la séquence 2 à l'indice it2 ;
        | it2  $\leftarrow$  it2 -1;
    end
    else if  $it2 = 0$  then
        | Ajout correspondance de type ajout pour la séquence 1 à l'indice it ;
        | it  $\leftarrow$  it -1;
    end
    else
        valMin  $\leftarrow$  min(vecValue[it-1][it2-1], vecValue[it][it2-1], vecValue[it-1][it2]);
        if  $valMin = vecValue[it-1][it2-1]$  then
            | it2  $\leftarrow$  it2 -1;
            | it  $\leftarrow$  it -1;
        end
        else if  $valMin = vecValue[it-1][it2]$  then
            | it2  $\leftarrow$  it2 -1;
        end
        else
            | it2  $\leftarrow$  it2 -1;
        end
        Ajout correspondance séquence 1(it-1) avec séquence 2(it2-1) ;
    end
end
end

```

Algorithm 5: Backtrack DTW

6.0.12 Complexité

Soit S_1 la première séquence S_2 la seconde séquence, V la dimension des éléments de chaque séquences, la complexité du remplissage est de $CARD(S_1) * CARD(S_2) * V$

Soit O la complexité du backtrack. On peut noter O avec cette inégalité :

$$\sqrt{CARD(S_1)^2 + CARD(S_2)^2} \leq O \leq CARD(S_1) + CARD(S_2)$$

6.0.13 Exemple

Nous allons appliquer l'algorithme avec les séquences $S1 = "1\ 1\ 2\ 3\ 2\ 0"$ et $S2 = "0\ 1\ 1\ 2\ 3\ 2\ 1"$. La distance entre deux éléments sera la différence au carré.

Initialisation

-	-	0	1	1	2	3	2	1
-	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	$+\infty$	-	-	-	-	-	-	-
1	$+\infty$	-	-	-	-	-	-	-
2	$+\infty$	-	-	-	-	-	-	-
3	$+\infty$	-	-	-	-	-	-	-
2	$+\infty$	-	-	-	-	-	-	-
0	$+\infty$	-	-	-	-	-	-	-

Remplissage et parcours

-	-	0	1	1	2	3	2	1
-	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	$+\infty$	1	0	0	1	4	1	0
1	$+\infty$	1	0	0	1	4	1	0
2	$+\infty$	4	1	1	0	1	0	1
3	$+\infty$	9	4	4	1	0	1	4
2	$+\infty$	4	1	1	0	1	0	1
0	$+\infty$	0	1	1	4	9	4	1

En gras vous pouvez observer le plus court chemin.

La distance est alors $\sqrt{2}$

6.0.14 Minimum Variance Matching : description et exemple

Cet algorithme permet d'effectuer un matching entre deux séquences. Le plus souvent ce sont des séquences numériques. Il s'appuie sur la programmation dynamique c'est à dire que la solution optimale trouvée pour une taille donnée s'appuie sur des solutions optimales de tailles intermédiaires. Contrairement à l'algorithme DTW il permet de sauter des éléments de la séquence cible (on appelle cette notion l'élasticité). La précondition est que **la taille de la séquence cible est de taille supérieure ou égale à la taille de la séquence source**

Soit une séquence a de taille m et une autre séquence b de taille n , le but de l'algorithme est de trouver la sous séquence b' la plus proche de a .

L'élasticité est le nombre d'éléments que l'on peut ignorer lors de la correspondance, ce nombre est inférieur ou égal à $n - m$. Nous avons donc une fonction f tel que :

$$\begin{cases} \forall m, n \text{ tel que } m \leq n \\ f : \{1..m\} \rightarrow \{1..n\} \\ f(i) \leq f(i+1) \text{ si } i \in \{1..m\} \\ a_i \text{ Correspond à } b_{f_i} \forall i \in \{1..m\} \end{cases}$$

Ainsi $\forall i \in \{1..m\}$ f_1 correspond à la position d'un des éléments de b' dans b . Le coût optimal se traduit alors

$$d(a, b) = d(a, b, \hat{f}) = \sqrt{\sum_{i=1}^m (b_{\hat{f}(i)} - a_i)^2}.$$

par :

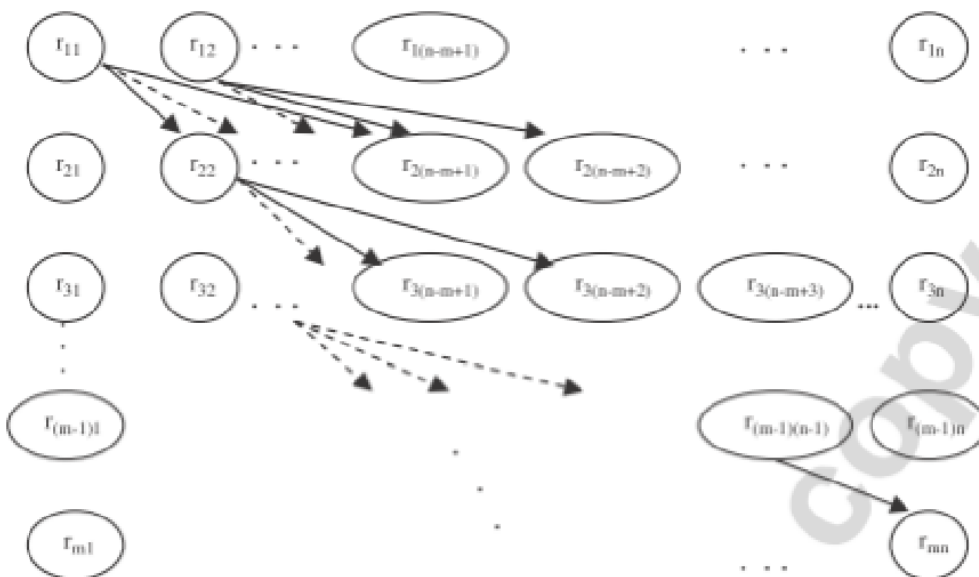
Soit $a : \{1, 2, 8, 6, 8\}$ la séquence source avec $m = 5$ et $b : \{1, 2, 9, 3, 3, 5, 9\}$ la séquence cible avec $n = 7$. Dans l'algorithme nous aurons ensuite une matrice r_{ij} contenant les différences entre les éléments tel que $r_{ij} = b_j - a_i$.

$$r_{ij} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 1 & 8 & 2 & 2 & 4 & 8 \\ -1 & 0 & 7 & 1 & 1 & 3 & 7 \\ 8 & -7 & 6 & 1 & -5 & -5 & -3 & -1 \\ 6 & -5 & -4 & 3 & -3 & -3 & -1 & 3 \\ 8 & -7 & -6 & 1 & -5 & -5 & -3 & 1 \end{bmatrix} \end{matrix}$$

On peut voir la résolution du minimum variance matching comme la résolution du problème du plus court chemin dans r . Cette matrice se comporte alors comme un graphe acyclique tel que $\forall i \in m, j \in n$ un sommet de ce graphe sera représenté avec le sommet r_{ij} . Tout successeurs r_{kl} devra respecter ces contraintes :

- $k = i + 1$: les successeurs se trouvent sur la ligne juste en dessous
- $l > j$: les successeurs se trouvent sur une colonne à droite du sommet.
- $n - l > m - k$ le nombre de colonnes restantes à droite doit être supérieur au nombre de lignes restantes.

Ainsi le graphe acyclique prend cette forme :



D'après les contraintes énoncées plus haut on peut écrire la matrice de coûts sous cette forme :

$$linkcost(r_{ij}, r_{kl}) = \begin{cases} (r_{kl})^2 = (b_k - a_i)^2 & \text{if } k = i + 1 \\ & \text{and } j+1 \leq l \leq j+1+(n-m)-(j-i), \\ \infty & \text{otherwise.} \end{cases}$$

L'enjeu est ensuite de prendre le successeur de plus faible coût. Dans l'exemple présenté plus haut, le chemin optimal se trouve de en gras ici $r_{ij} = b_j - a_i$.

	1	0	1	8	2	2	4	8
	1	-1	0	7	1	1	3	7
$r_{ij} =$	8	-7	6	1	-5	-5	-3	-1
	6	-5	-4	3	-3	-3	-1	3
	8	-7	-6	1	-5	-5	-3	1

Le coût optimal est donc $\sqrt{3}$ L'algorithme se fait en deux étapes. D'abord le remplissage des tableaux de coût. Puis la recherche d'une solution par backtrack.

6.0.15 algorithme

Remplissage

```

input : s1  $\leftarrow$  Sequence , tailleS1  $\leftarrow$  entier , s2  $\leftarrow$  Sequence , tailleS2  $\leftarrow$  entier
output: pathCost  $\leftarrow$  vecteur[tailleS1][tailleS2] de flottants, path  $\leftarrow$  vecteur[tailleS1][tailleS2] d'entiers

Précondition : tailleS2  $\geq$  tailleS1;
Création des tableaux;
Création du tableau de différence tabDiff;
Initialisation des variables;
elasticity  $\leftarrow$  max(tailleS2-tailleS1, 1+ $\lfloor$ tailleS1/10 $\rfloor$ );
i  $\leftarrow$  1;
j  $\leftarrow$  1;
while i < tailleS1+1 do
    while j < tailleS2+1 do
        pathCost[i-1][j-1]  $\leftarrow$   $+\infty$ ;
        path[i-1][j-1]  $\leftarrow$  0;
        j  $\leftarrow$  j + 1;
    end
    j  $\leftarrow$  1;
    i  $\leftarrow$  i + 1;
end
j  $\leftarrow$  1;
while j <= elasticity+1 do
    pathCost[1-1][j-1] = distancetabDiffi-1,j-1;
    j  $\leftarrow$  j + 1;
end
i  $\leftarrow$  2;
while i < tailleS1 do
    stopk  $\leftarrow$  min(i-1+elasticity,tailleS2);
    debk  $\leftarrow$  max(1,i-1);
    k  $\leftarrow$  debk; while k  $\leq$  stopk do
        stopj  $\leftarrow$  min(k+1+elasticity,tailleS2);
        j  $\leftarrow$  k + 1;
        while j  $\leq$  stopk do
            if pathCost[i-1][j-1] > pathCost[i-1][k-1] + distancetabDiffi-1,j-1 then
                pathCost[i-1][j-1] = pathCost[i-1][k-1] + distancetabDiffi-1,j-1;
                path[i-1][j-1] = k;
            end
            j  $\leftarrow$  j + 1;
        end
        k  $\leftarrow$  k + 1;
    end
    it2  $\leftarrow$  1;
    i  $\leftarrow$  i + 1;
end

```

Algorithm 6: MVM

Backtrack

```
input : pathCost  $\leftarrow$  vecteur[tailleS1][tailleS2] de flottants, path  $\leftarrow$  vecteur[tailleS1][tailleS2]
        d'entiers, tailleS1  $\leftarrow$  entier, tailleS2  $\leftarrow$  entier,
output: listCorrespondance  $\leftarrow$  Liste de correspondances

création du tableau s2index[tailleS1] contenant les indices cible sélectionnés pour la correspondance ;
s2index[tailleS1-1]  $\leftarrow$  0;
i  $\leftarrow$  1;
while  $i < \text{tailleS2}$  do
    if  $\text{pathCost}[\text{tailleS1} - 1][i] \leq \text{pathCost}[\text{tailleS1} - 1][\text{s2index}[\text{tailleS1} - 1]]$  then
        | s2index[tailleS1 - 1]  $\leftarrow$  i;
    end
    i  $\leftarrow$  i+1;
end
i  $\leftarrow$  1;
while  $i < \text{tailleS1}$  do
    | s2index[tailleS1-i-1] = path[tailleS1-i][s2index[tailleS1-i]]-1;
    | i  $\leftarrow$  i+1;
end
i  $\leftarrow$  1;
while  $i < \text{tailleS1}$  do
    | Ajout correspondance séquence 1(i-1) avec séquence 2(s2index[i-1]) ;
    | i  $\leftarrow$  i+1;
end
```

Algorithm 7: Backtrack

6.0.16 Complexité

Soit S_1 la première séquence S_2 la seconde séquence, V la dimension des séquences la complexité du remplissage est de $\text{CARD}(S_1) * \text{CARD}(S_2) * V$

Soit O la complexité du backtrack. Cette complexité est de $\max(\text{CARD}(S_1), \text{CARD}(S_2))$

Etude de Contraintes

Dans cette étape du projet l'objectif était de dégager des contraintes sur les séquences. Ce lot a duré **18** jours. La première entrée à étudier a été la recherche de contraintes sur les mots. Puis celle sur les signatures et des images de mots.

7.0.17 Mots

Une des contraintes était de se concentrer, en plus du mots sur sa prononciation. C'est là que les soundex peuvent nous aider. Soundex est un algorithme phonétique d'indexation de noms par leur prononciation en anglais britannique. L'objectif basique est que les noms ayant la même prononciation soient codés avec la même chaîne de manière à pouvoir trouver une correspondance entre eux malgré des différences mineures d'écriture. Soundex est le plus largement connu des algorithmes phonétiques et est souvent utilisé incorrectement comme synonyme de « algorithme phonétique ».

Une autre contrainte dégagée a été de privilégier les similarités en début de mots (prefixe) en terme de poids plutôt que les similarités globales.

Soit p_i la pénalité d'une différence de caractère sur un indice i .

Soit $j > i$, alors $p_j > p_i$.

Ce qui fait qu'une différence sur le préfixe a plus de poids qu'une différence sur le suffixe.

7.0.18 Signatures

Une signature est représentée sous la forme d'une séquences de type vecteur de caractéristique. Dans ce vecteur de caractéristique on y trouve la position en X, en Y, la pression en fonction du temps. Selon le système d'acquisition on peut y trouver d'autres informations comme la vitesse ou l'accélération. La contrainte dégagée pour les signatures est de donner moins d'importance aux débuts de tracés. En effet l'utilisateur est plus hésitant en début de tracé surtout dans le cas de signatures on-line.

7.0.19 Image de mots

La recherche de mots par image permet d'extraire des mots dans les images. On distingue deux types : la transcription (OCR) très efficace pour les textes récents utilisant des polices type arial et le word spotting (repérage de mots) qu'on utilise sur les textes manuscrits et/ou anciens. Nous nous intéressons au word spotting.



FIGURE 7.1 – Image de mots

Une image de mots sera ensuite représentée sous la forme d'une séquences de type vecteur de caractéristique. Dans un vecteur de caractéristique on y trouve des informations comme la position du caractère, le pourcentage de "noir", l'inclinaison etc. Nous aurons une ligne représentant un mot, avec un découpage déjà établi en séquence. Concernant les images de mots les idées dégagées étaient :

- La gestion de l'italique
- Les accents



- La gestion du bruit

7.0.20 Résultats et remise en cause

Nous avons considéré que bien qu'intéressantes, ces idées sont trop informelles pour nous permettre une exploitation. En effet le lien entre applicatif et algorithmique n'est pas effectué.

Ainsi suite à la soutenance mi-parcours nous avons décidé de supprimer le lot 3 (étude des algorithmes par rapport aux contraintes) et de se concentrer sur la conception et le développement de la toolbox.

Conception et développement

Dans cette étape du projet et suite à la soutenance mi-parcours l'objectif était de concevoir et de développer une toolbox qui implémente les 4 algorithmes étudiés. Nous avons d'abord conçu une structure de données pour les séquences avant de créer les briques applicatives autour. Ce lot a duré **16** jours.

8.1 Structure de données et diagramme de classe

Cette tâche a nécessité 2 jours. Voici la structure de données imaginée pour l'application sous forme d'un diagramme de classe :

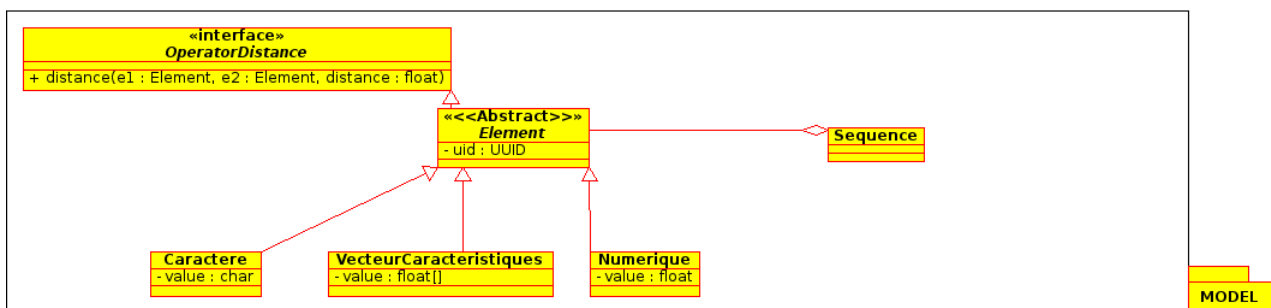


FIGURE 8.1 – Modèle

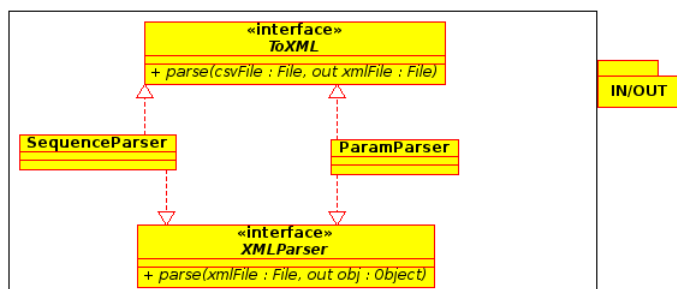
Une séquence est une liste d'éléments. Un élément peut être de type Caractère, Vecteur de caractéristiques ou numérique. Chaque type d'éléments a sa propre implémentation de l'opérateur de distance. Pour les caractères nous avons utilisé un opérateur booléen. Pour le vecteur de caractéristiques et les éléments numériques nous utilisons la distance euclidienne.

Cette structure appartient au package modèle. Plusieurs autres entités y sont greffés :

- le package d'entrée/sortie
- le package de calcul
- l'application en ligne de commande

Entrée Sortie

Ce package permet de gérer le chargement et la restitution textuelle de séquences.



Deux choses importantes sont à considérer pour ce package. D'abord les séquences doivent être sous un format XML pour sa portabilité pour d'autres futures applications. Puis pour faciliter l'exploitation actuelle on doit pouvoir gérer également des fichiers de type CSV. Ainsi deux transformations seront implémentées : le passage CSV → XML et le passage XML → objet.

Calcul

Ce package permet d'utiliser les 4 algorithmes sur les séquences. Puis paramétrer les séquences.

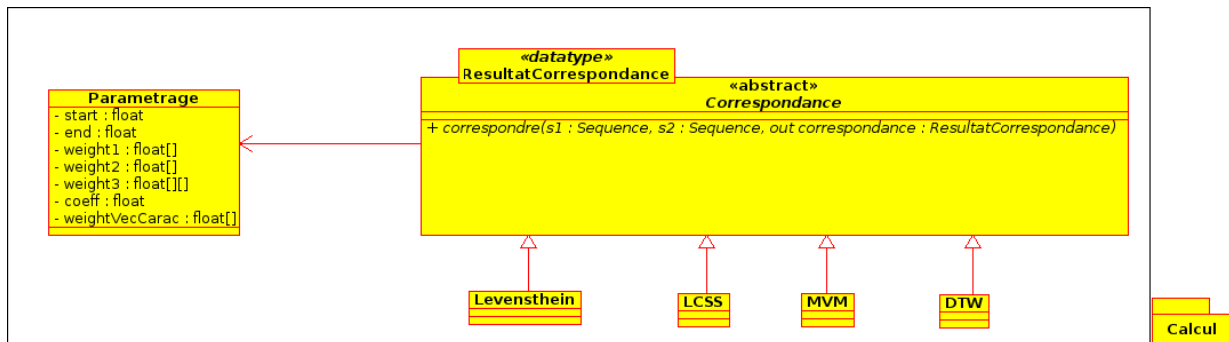


FIGURE 8.2 – Package de calcul

Une correspondance prend en entrée deux séquences et renvoie une structure données de type "ResultatCorrespondance".

Cette structure contient

- la distance
- un tableau contenant les indices concernés par la correspondance dans la séquence 1
- un tableau contenant les indices concernés par la correspondance dans la séquence 2

Lien entre les packages

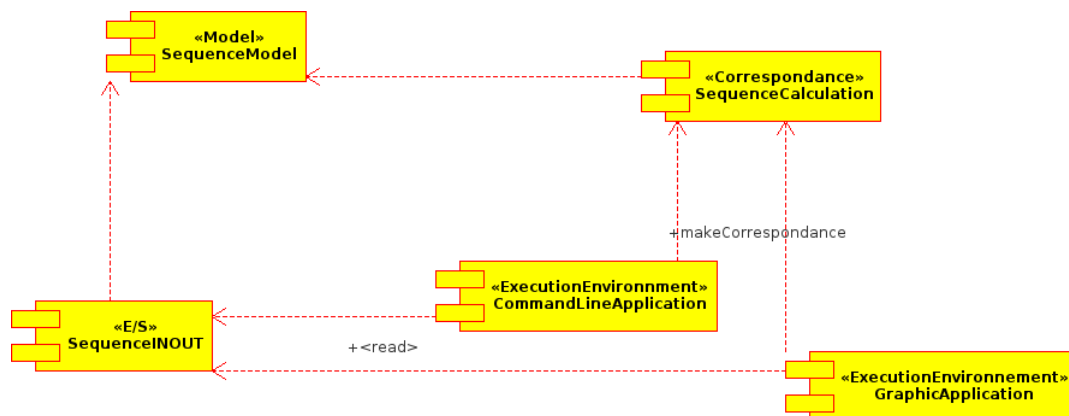


FIGURE 8.3 – Diagramme de composant

8.2 Développement et test

Pour chaque module, nous avons deux tâches. Une tâche de développement et une tâche de tests unitaires. Cette tâche a nécessité environ 11 jours.

8.2.1 Formattage des fichiers d'entrée

Nous avons convenu d'un format pour les fichiers d'entrée de type CSV

- Marque de début de séquence
- Type de séquence : Caractère/Vecteur/Numérique
- Mot clé "NEXT" pour entrer plusieurs séquences dans le même fichier
- Marque de fin de séquence

SEQUENCE
TYPE
Character
BEGIN
w
b
d
c
z
d
NEXT
a
b
d
c
u
m
NEXT
a
c
b
END

FIGURE 8.4 – Format de fichier d'entrée

8.2.2 Spécificités de développement

Tests



Avant chaque développement d'une fonction, il faut réfléchir si un test unitaire serait utile. Si c'est le cas on l'inscrit dans un fichier excel contenant les tests unitaires pour chaque module.

	A	B	C	D	E
1	Sequence Matching Tests				
2	Test	Package	Class	Method	OK/KO
3	Copy Character	Model	Character	Character(Character const & copy);	OK
4	Copy CharacteristicVector	Model	CharacteristicVector	CharacteristicVector(CharacteristicVector const & copy);	OK
5	Copy Numeric	Model	Numeric	Numeric(Numeric const & copy);	OK
6	Copy Sequence	Model	Sequence	Sequence(Sequence const & copy);	OK
7	Is Character	Model	Character	float distance(Element *eOD1, Element *eOD2);	OK
8	Is CharacteristicVector	Model	CharacteristicVector	float distance(Element *eOD1, Element *eOD2);	OK
9	Is Numeric	Model	Numeric	float distance(Element *eOD1, Element *eOD2);	OK
10	Same Size	Model	CharacteristicVector	float distance(Element *eOD1, Element *eOD2);	OK
11	Remove	Model	CharacteristicVector	void removeValue(int index)	OK
12	Add	Model	Sequence	void addValue(Element elt)	OK
13	Remove	Model	Sequence	void removeValue(int index)	OK
14	invalid Weight (must be btwn 0 and 1)	Calc	Parametrage	void setS1Weight(unsigned int index, float value)	OK
15	invalid Index	Calc	Parametrage	void setS1Weight(unsigned int index, float value)	OK
16	invalid Weight (must be btwn 0 and 1)	Calc	Parametrage	void setS2Weight(unsigned int index, float value)	OK
17	invalid Index	Calc	Parametrage	void setS2Weight(unsigned int index, float value)	OK
18	invalid Weight (must be btwn 0 and 1)	Calc	Parametrage	void setVecCaracWeight(unsigned int index, float value)	OK
19	invalid Index	Calc	Parametrage	void setVecCaracWeight(unsigned int index, float value)	OK
20	No Vec Carac	Calc	Parametrage	void setVecCaracWeight(unsigned int index, float value)	OK
21	invalid Weight (must be btwn 0 and 1)	Calc	Parametrage	void setMatrixWeight(unsigned int index1, unsigned int index2, float value)	OK
22	invalid Index	Calc	Parametrage	void setMatrixWeight(unsigned int index1, unsigned int index2, float value)	OK
23	invalid Weight (must be btwn 0 and 1)	Calc	Parametrage	void setDistanceWeight(float value)	OK
24	invalid Index	Calc	Parametrage	float getS1Weight(unsigned int index)	OK
25	invalid Index	Calc	Parametrage	float getS2Weight(unsigned int index)	OK
26	invalid Index	Calc	Parametrage	float getVecCaracWeight(unsigned int index)	OK
27	No Vec Carac	Calc	Parametrage	float getVecCaracWeight(unsigned int index)	OK
28	invalid Index	Calc	Parametrage	float getMatrixWeight(unsigned int index1, unsigned int index2)	OK

FIGURE 8.5 – Tests unitaires

Quand le module est terminé les tests sont effectués avec l'outil

Portabilité

La portabilité de la toolbox est assurée par un développement le plus souvent indépendant de la machine. On utilise quelques routines propres à linux ou à windows (cas des uid). Dans ce cas, on utilise des directives préprocesseur.

Exception

Une classe d'exception personnalisée a été créée avec un numéro d'erreur, un type d'erreur et une description.

Xml

La lecture et l'écriture de fichiers XML est assurée grâce à l'outil "RapidXML" portable et simple d'utilisation.

Commandes de la toolbox

Nous avons créé un système de commandes à envoyer en ligne de commande. Voici un extrait de l'aide :


```

Gestion des paramètres toolbox :

arguments :

-sequences <fichier1.csv> <fichier2.csv>

Permet d indiquer les séquences d entrée

-method lvn|lcd|dtw|mvmm

Permet d indiquer la méthode à utiliser

-patron <name> <fichier1.csv> <fichier2.csv>

Permet de créer un patron

-sequences <fichier1.csv> <fichier2.csv> -param <fichier.xml>

Permet d utiliser un fichier de paramètres créé auparavant

```

FIGURE 8.6 – Format de l'aide

Parametrage

L'utilisateur peut générer des fichiers de paramétrages. Ces fichiers seront ensuite entrés en plus des fichiers de séquences. Les fichiers de paramétrage permettent de donner des coefficients aux éléments des séquences. Si l'on place un coefficient de 25% sur le premier élément de la première séquence le résultat en sera impacté et le poids d'une correspondance avec cet élément sera réduit.

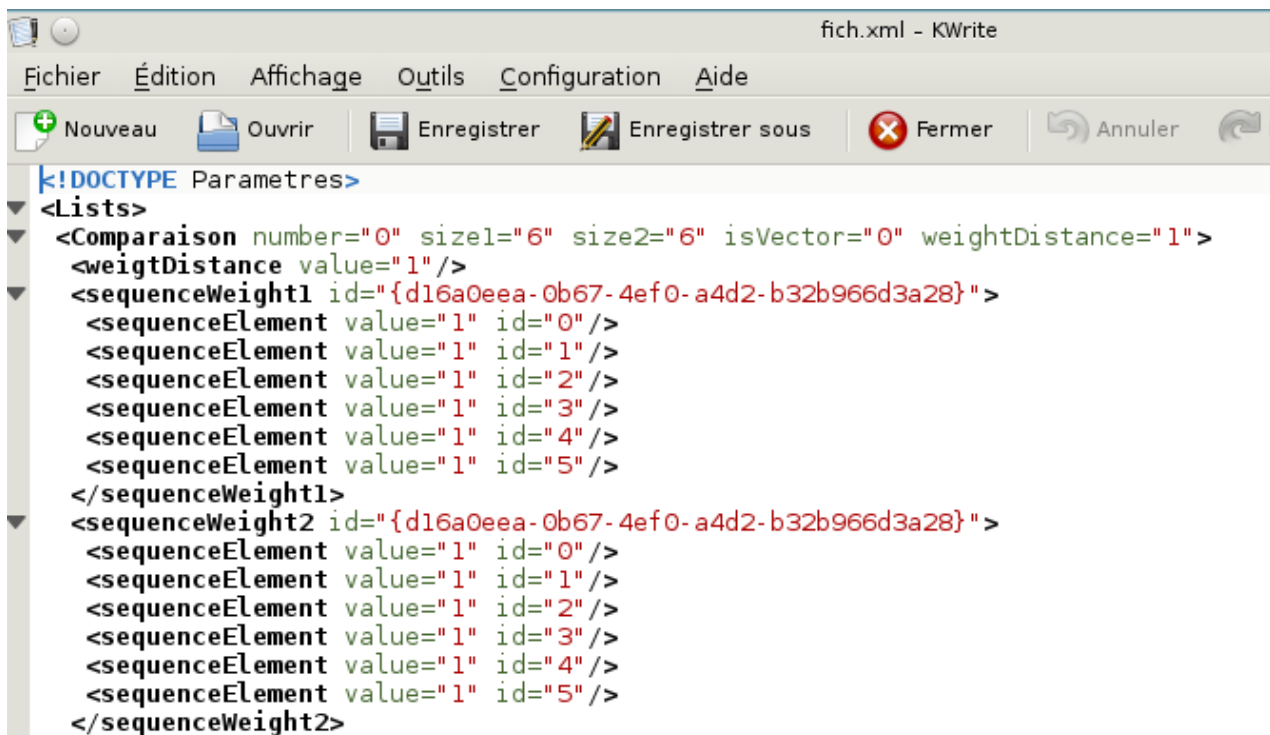


FIGURE 8.7 – Patron de correspondance

Partie graphique

Cette partie est conçue avec l'outil QT. Voici le diagramme de classe de l'interface graphique :

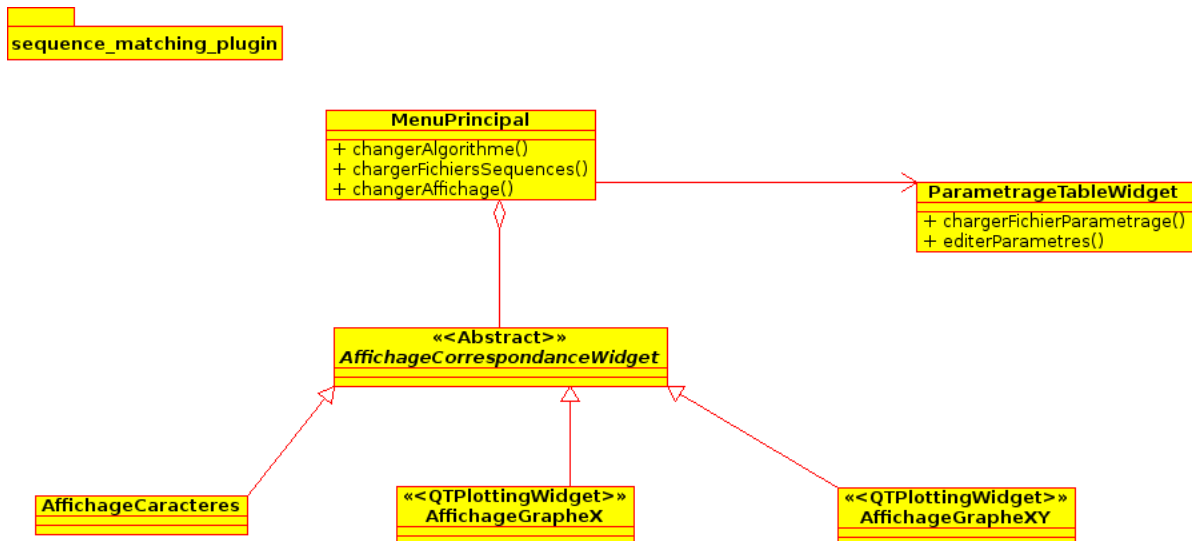


FIGURE 8.8 – GUI

Nous pouvons choisir nos algorithmes et 3 types d'affichage disponibles selon l'algorithme choisi. Voici les options possibles dans notre UI :

Algorithme	Type de séquence	Texte	X	X/Y
Levenshtein	Caractères	x	-	-
	Numérique	x	x	-
	Vecteur	x	x	x
LCS	Caractères	x	-	-
	Numérique	x	x	-
	Vecteur	x	x	x
DTW	Caractères	x	-	-
	Numérique	x	x	-
	Vecteur	x	x	x
MVM	Caractères	x	-	-
	Numérique	x	x	-
	Vecteur	x	x	x

FIGURE 8.9 – Choix d'affichages possibles selon l'algorithme et le type de séquence

Le menu principal envoie des signaux aux différents "Panel" d'affichage. Ceux-ci rafraîchissent ensuite leurs données.

L'affichage de courbes est assuré grâce à l'objet QCustomPlot.

Resultat

9.1 Deploiement

La livraison contient :

- Une version en ligne de commande de l'application sous l'IDE QtCreator
- Une version en ligne de commande de l'application sous l'IDE Visual Studio
- Un livret utilisateur
- Un livret de développeur (avec une partie Doxygen)
- Une version "bibliothèque statique" de l'application sous QtCreator
- Une version Graphique de l'application sous l'IDE QtCreator

Nous allons illustrer des résultats d'exécution de notre toolbox. D'abord en ligne de commande. Puis de manière graphique.

9.2 Ligne de commande

Nous allons illustrer les résultats à travers deux exemples.

9.2.1 Exemple 1

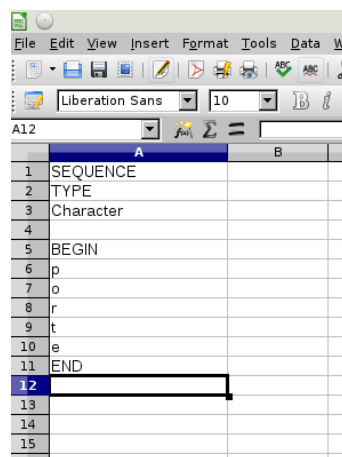
Nous allons comparer deux séquences de type chaîne de caractère.

Séquence 1 : (p, o, r, t, e)

Séquence 2 : $(r, a, p, p, o, r, t, e, r)$

D'abord on aimerait savoir quelles sont les opérations nécessaires pour passer d'une chaîne à l'autre ?

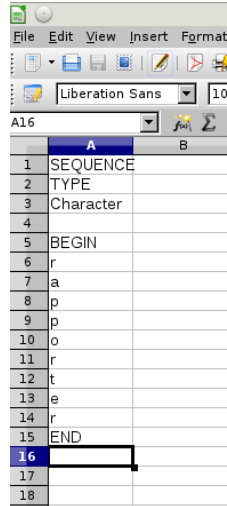
Créons d'abord nos fichiers :



The image shows a screenshot of a spreadsheet application window. The window has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Format', 'Tools', 'Data', and 'Window'. Below the menu bar is a toolbar with various icons. The spreadsheet has two columns, 'A' and 'B', and rows numbered 1 to 15. Column A contains the following text: 'SEQUENCE', 'TYPE', 'Character', 'BEGIN', 'p', 'o', 'r', 't', 'e', 'END'. Column B is empty. The cell A12 is highlighted with a blue background.

	A	B
1	SEQUENCE	
2	TYPE	
3	Character	
4		
5	BEGIN	
6	p	
7	o	
8	r	
9	t	
10	e	
11	END	
12		
13		
14		
15		

FIGURE 9.1 – Fichier 1 sous forme CSV



	A	B
1	SEQUENCE	
2	TYPE	
3	Character	
4		
5	BEGIN	
6	r	
7	a	
8	p	
9	p	
10	o	
11	r	
12	t	
13	e	
14	r	
15	END	
16		
17		
18		

FIGURE 9.2 – Fichier 2 sous forme CSV

Nous devons exécuter levensthein, la commande sera donc de cette forme :

```
abdourahman@debian:~/Share/Dropbox/Private/workspace/PFE/Projet/matching-elastique/sequence-matching-build-desktop-Qt_4_8_2_in_PATH_System_Debug$ ./app -sequences ../ex_rapport.csv ../ex_rapport2.csv -method -lvn
=====
Result
Distance = 4
Correspondance
    p o r t e
- - - - -
r a p p o r t e r
=====
```

FIGURE 9.3 – Résultat levensthein ligne de commande

Pour passer de rapporter à porte il faut donc retirer r,a,p et r. Le coût pour passer d'une expression à l'autre est donc de 4.

Maintenant on voudrait savoir quelle est la plus longue sous-séquence commune aux deux mots.

Utilisons la toolbox avec cette fois l'algorithme LCS :

```
abdourahman@debian:~/Share/Dropbox/Private/workspace/PFE/Projet/matching-elastique/sequence-matching-build-desktop-Qt_4_8_2_in_PATH_System_Debug$ ./app -sequences ../ex_rapport.csv ../ex_rapport2.csv -method -lcs
=====
Result
Distance = 5
Correspondance
p o r t e
p o r t e
- - p o r t e -
r a p p o r t e r
Longest common sequence : {p o r t e }
=====
abdourahman@debian:~/Share/Dropbox/Private/workspace/PFE/Projet/matching-elastique/sequence-matching-build-desktop-Qt_4_8_2_in_PATH_System_Debug$
```

FIGURE 9.4 – Résultat LCS ligne de commande

La plus longue sous séquence est évidemment "porte".

9.2.2 Exemple 2

Nous allons comparer deux séquences de type numérique.

Séquence 1 : (1, 10, 2, 8, 5, 9)

Séquence 2 : (1, 7, 3, 8, 4, 9, 13, 11, 4)

Faisons une correspondance de type DTW d'abord :

```
abdourahman@debian:~/Share/Dropbox/Private/Workspace/PFE/Projet/matching-elastique/sequence-matching-b
_4_8_2_in_PATH_System_Debug$ ./app -sequences ../ex_rapport3.csv ../ex_rapport4.csv -method -dtw
=====
Result
Distance = 7.48331
Correspondance
    1  7  3  8  4  9  13 11  4
1  x  -  -  -  -  -  -  -
10 -  x  -  -  -  -  -  -  -
2  -  -  x  -  -  -  -  -  -
8  -  -  -  x  -  -  -  -  -
5  -  -  -  -  x  -  -  -  -
9  -  -  -  -  -  x  x  x  x
=====
```

FIGURE 9.5 – Résultat DTW ligne de commande

Nous obtenons le chemin optimal en observant les croix.

Regardons le résultat avec une correspondance de type MVM

```
abdourahman@debian:~/Share/Dropbox/Private/Workspace/PFE/Projet/matching-elastique/sequence-matching-build-desktop-
_4_8_2_in_PATH_System_Debug$ ./app -sequences ../ex_rapport3.csv ../ex_rapport4.csv -method -mvm
=====
Result
Distance = 6
Correspondance
    1  7  3  8  4  9  13 11  4
1  x  -  -  -  -  -  -  -
10 -  x  -  -  -  -  -  -  -
2  -  -  x  -  -  -  -  -  -
8  -  -  -  x  -  -  -  -  -
5  -  -  -  -  x  -  -  -  -
9  -  -  -  -  -  x  -  -  -
=====
```

FIGURE 9.6 – Résultat DTW ligne de commande

On peut observer que l'algorithme MVM saute les correspondances jugées trop coûteuses afin d'optimiser la distance.

9.3 GUI

Expérimentons nos résultats obtenus lors de la dernière partie mais dans un repère.

Pour rappel

Séquence 1 : (1, 10, 2, 8, 5, 9)

Séquence 2 : (1, 7, 3, 8, 4, 9, 13, 11, 4)

Nous allons projeter les deux séquences sur un repère ayant en X l'indice des séquences et en Y leur valeur avec l'algorithme DTW puis MVM :

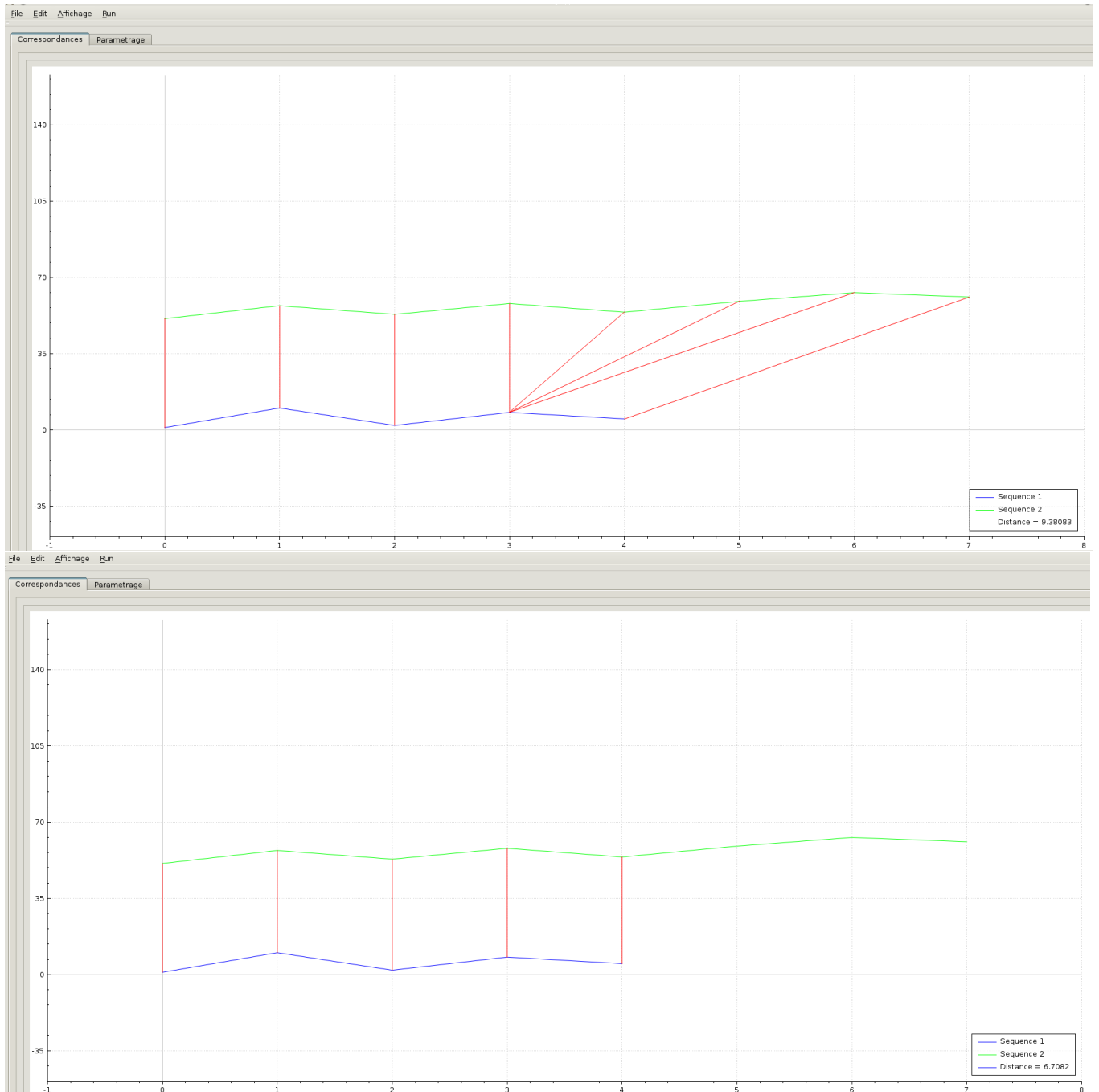


FIGURE 9.7 – Résultat séquences DTW/MVM

Comparons maintenant les lettres 'M' et 'W' sous forme de vecteurs de caractéristiques de dimension 2 avec en X l'abscisse et en Y l'ordonnée.

Séquence 1 : $\begin{pmatrix} x = 0 \\ y = 15 \end{pmatrix}, \begin{pmatrix} x = 2 \\ y = 15 \end{pmatrix}, \begin{pmatrix} x = 4 \\ y = 0 \end{pmatrix}, \begin{pmatrix} x = 6 \\ y = 15 \end{pmatrix},$

$\begin{pmatrix} x = 8 \\ y = 0 \end{pmatrix}, \begin{pmatrix} x = 10 \\ y = 15 \end{pmatrix}, \begin{pmatrix} x = 12 \\ y = 15 \end{pmatrix}$

Séquence 2 : $\begin{pmatrix} x = 0 \\ y = 0 \end{pmatrix}, \begin{pmatrix} x = 2 \\ y = 0 \end{pmatrix}, \begin{pmatrix} x = 4 \\ y = 15 \end{pmatrix}, \begin{pmatrix} x = 6 \\ y = 0 \end{pmatrix},$

$\begin{pmatrix} x = 8 \\ y = 15 \end{pmatrix}, \begin{pmatrix} x = 10 \\ y = 0 \end{pmatrix}, \begin{pmatrix} x = 12 \\ y = 0 \end{pmatrix}$

Voici les résultats de correspondance obtenus avec DTW et MVM.

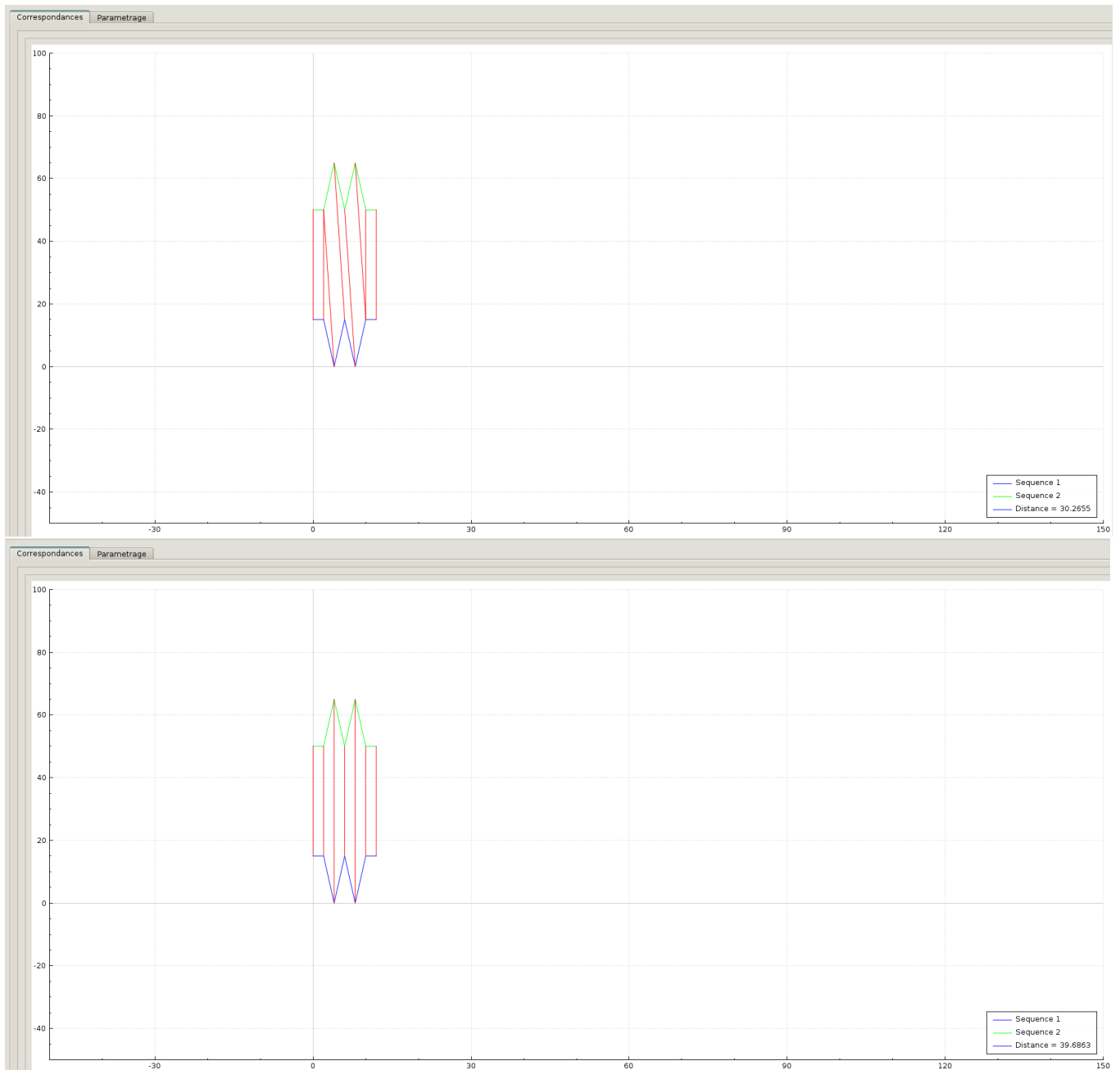


FIGURE 9.8 – Résultat séquences DTW/MVM

Nous pouvons observer que MVM saute les correspondances trop éloignées.

Problèmes rencontrés et solutions apportées

10.1 Contraintes

Suite à la soutenance mi-parcours nous avons considéré que bien qu'intéressantes, ces idées sont trop informelles pour nous permettre une exploitation. En effet le lien entre applicatif et algorithmique n'est pas effectué. La partie "contraintes" sera développée lors de futurs développement. C'est pourquoi une réflexion sur la mise en place des contraintes était nécessaire. Ainsi une classe "Parametrage" est disponible. Cette classe permet de "coefficients" les correspondances entre les séquences.

Ainsi une contrainte est une classe qui prend en entrée un objet "Parametrage" et le modifie.

Voici un exemple de future conception possible :

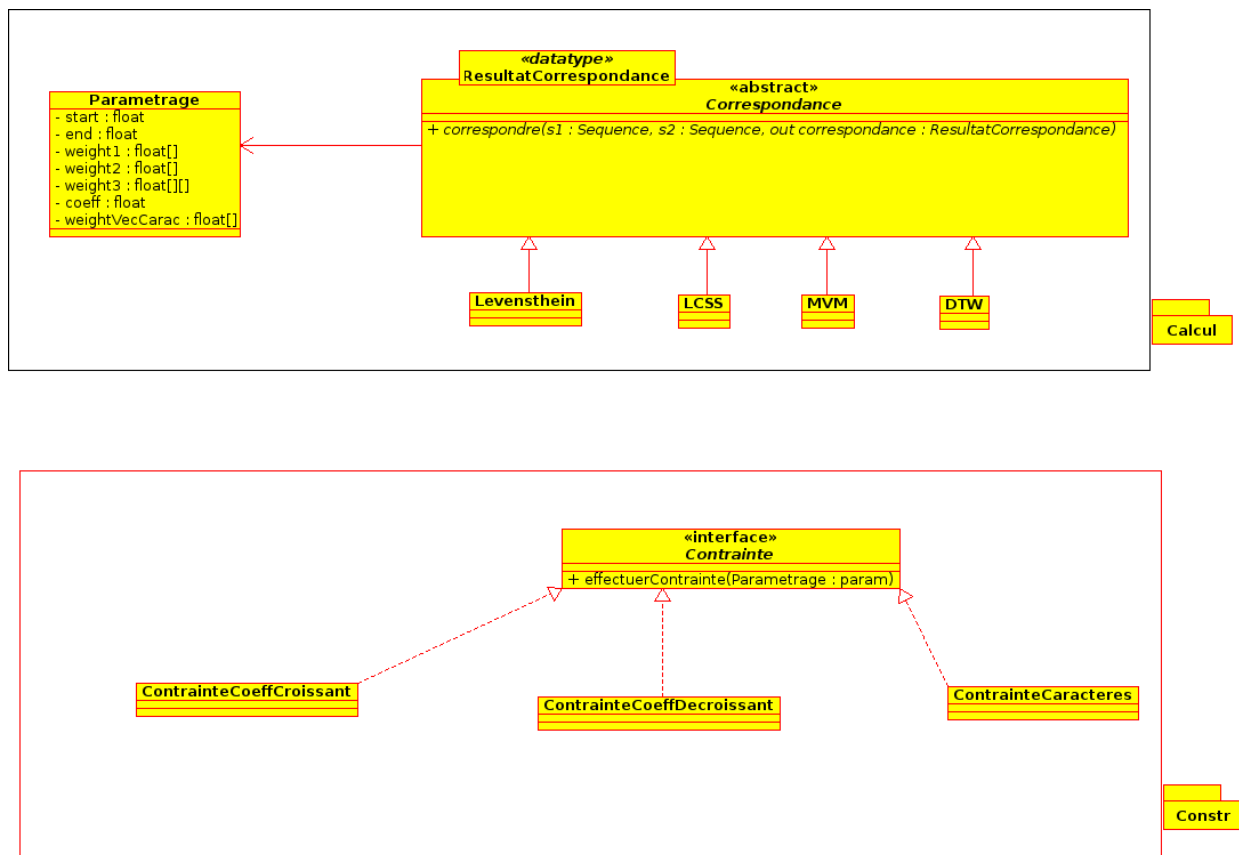


FIGURE 10.1 – Futur package de projet

10.2 Portabilité

En début de développement l'accent n'était pas mis sur la portabilité. Ainsi certaines librairies inutiles étaient utilisées. Ces librairies alourdissent le livrable et nuisent à la lisibilité (QTXML, Quid). Des solution alternatives ont été utilisées comme RapidXML.

Conclusion

Ce projet a été une bonne occasion d'explorer les différentes possibilités qu'offrent les algorithmes de matching. L'utilisation des différentes techniques telles que Levensthein ou DTW m'a permis d'en apprendre plus sur la programmation dynamique, la manière de parcourir et remonter les résultats. Cela m'a par ailleurs permis de développer un projet en partant de zéro, en m'occupant de toute la partie gestion de projet et développement, de réaliser des analyses. Le travail d'ailleurs pu être réparti selon un cycle de développement qui correspondait à mes attentes et à mes habitudes. J'ai également pu ré-apprendre à utiliser Qt ainsi que la gestion de librairies, qui a été une étape cruciale lors du développement de l'application. J'ai pu aussi voir mes limites actuelles concernant l'analyse. En effet j'ai éprouvé des difficultés à faire le lien entre algorithmique et applicatif et je compte m'améliorer sur cet aspect. Le fait que les résultats de l'application seront peut-être utilisés pour un projet plus tard m'a motivé à être rigoureux et exhaustif.

Bibliographie

- <http://www.cs.cmu.edu/afs/cs/academic/class/15828-s98/lectures/0128/sld015.htm>
- <http://www.levenshtein.net/>
- http://en.wikipedia.org/wiki/Longest_common_subsequence_problem
- rapport Facomprez Maxime Béchu Jean-François algo C
- http://www.phon.ox.ac.uk/jcoleman/old_SLP/Lecture_5/DTW_explanation.html
- An elastic partial shape matching technique
- <http://www.highprogrammer.com/alan/numbers/soundex.html>
- <http://fr.wikipedia.org/wiki/Soundex>
- Longin Jan Latecki , Vasileios Megalooikonomou, Qiang Wang, Deguang Yu

Matching élastique et robuste de séquences

Département Informatique
5^e année
2013 - 2014

Rapport de projet

Résumé : Ce projet de fin d'études concerne une application. Ce logiciel effectue un matching entre différentes séquences de formes multiples. Chaînes de caractères, signatures, images de mots... Notre application doit pouvoir prendre en compte les contraintes de chacun de ces domaines

Mots clefs : Séquence, Distance, Dynamic Time Warping, Élastique, Contrainte

Abstract: In this project the goal is to make a software which match several sequences. These sequences can be characters, signatures, handwritten words images, protein sequences... Our software must consider the constraints coming from very different sources

Keywords: Sequence, matching, DTW, Elastic, constraint

Encadrants

Nicolas RAGOT
nicolas.ragot@univ-tours.fr

Université François-Rabelais, Tours

Etudiants

Abourahman ADEN HASSAN
abdourahman.adenhassan@etu.univ-tours.fr

DI5 2013 - 2014