



ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Spécialité Informatique

64, Avenue Jean Portalis

37200 TOURS, FRANCE

Tél. +33 (0)2 47 36 14 14

www.polytech.univ-tours.fr

Rapport de projet de fin d'étude 2015

Recherche par mots-clés "image" dans une masse d'image de documents : méthode

Laboratoire

Laboratoire Informatique de Tours(EA630)
Équipe Reconnaissance des Formes, Analyse
d'Images (RFAI),
64, avenue Jean Portalis
37200, Tours



Étudiant

Bastien MEUNIER
bastien.meunier@gmail.fr
DI5 2014 - 2015

Encadrant

Nicolas RAGOT
nicolas.ragot@univ-tours.fr

Table des matières

INTRODUCTION	6
REMERCIEMENTS	7
1 PROJET	8
1.1 Contexte	8
1.2 Objectif	8
1.3 Méthodologie	9
1.4 Planning	9
2 SEQUENCES	12
2.1 Définition et distance	12
2.2 Type de séquences	12
2.2.1 Séquences de flottant simple	12
2.2.2 Séquences de flottant vecteur	12
2.3 Obtention des séquences	12
2.4 Formatage des séquences	13
2.4.1 Simple	13
2.4.2 Vecteur	14
2.4.3 UCR	15
3 ÉTUDE ALGORITHMIQUE	16
3.1 Introduction	16
3.1.1 Structure de résultat	16
3.1.2 Fonctions nécessaires	17
3.2 Continuous Dynamic Programming	19
3.2.1 Présentation	19
3.2.2 Pseudo-code	19
3.2.3 Exemple	20
3.3 Flexible Sequences Matchnig	22
3.3.1 Présentation	22
3.3.2 Pseudo-code	23
3.3.3 Exemple	25
3.4 Exemplary Sequences Cardinality	26
3.4.1 Présentation	26
3.4.2 Pseudo-code	27
3.4.3 Exemple	29
4 ÉTUDE DE L'EXISTANT	32
4.1 Présentation	32
4.2 Fonctionnalités	33
4.3 L'architecture de la <i>ToolBox</i>	33
4.3.1 Architecture générale	33



4.3.2	Diagrammes de classes	34
4.3.3	Exécution	35
4.4	Remarques	36
4.4.1	Remarque général	36
4.4.2	Remarque spécifique à la reprise	36
5	DEVELOPPEMENT	38
5.1	Modification et ajout	38
5.1.1	Ajout	38
5.1.2	Modification	39
5.1.3	Diagramme final	40
5.2	Exécution	41
5.3	Tests	41
6	UTILISATION	42
6.1	Commande	42
6.2	Paramétrage et Valeur par défaut	42
6.2.1	CDP	43
6.2.2	FSM et ESC	43
6.3	Interprétation	43
7	RESULTATS	45
7.1	Exemple de résultat de type 1	45
7.2	Exemple de résultat de type 2	46
8	PROBLÈMES RENCONTRÉS	47
8.1	Problème de la <i>ToolBox</i>	47
8.2	Communication avec les acteurs du projet	47
8.3	Problème de gestion de projet	48
9	CONCLUSION	49

Table des figures

1.1	Planning prévisionnel	10
1.2	Planning final	11
2.1	premier format texte de séquences	14
2.2	second format texte de séquences	14
2.3	troisième format texte de séquences	15
3.1	CDP : Calcule distance	21
3.2	CDP : Calcule des coûts	21
3.3	CDP : Backtrack	21
3.4	FSM DAG	22
3.5	FSM : Calcule distance	25
3.6	FSM : Calcule des coûts	26
3.7	FSM : Backtrack	26
3.8	ESC : Determination des sauts	30
3.9	FSM : Calcule des coûts	30
3.10	ESC : Backtrack	31
4.1	Cas d'utilisations original	33
4.2	Ancien model	34
4.3	Ancien calc	34
4.4	Ancien inout	35
4.5	Schémas d'exécution	35
4.6	Ancien affichage de résultat	36
4.7	Ancien format de séquences	37
5.1	Package calc	40
5.2	Package inout	40
5.3	Schémas d'exécution	41
6.1	commande d'exécution	42
6.2	commande d'exécution	42
6.3	Fichier de paramètre	43
6.4	Fichier de résultat	44
7.1	resultat sequences simple	45
7.2	resultat sequences simple	46

INTRODUCTION

Ce document est le rapport de projet concernant le projet de fin d'études 197 : **Recherche par mots-clés "image" dans une masse d'image de documents : méthode**. L'objectif de ce document est de décrire le contexte dans lequel le projet intervient, les divers choix choisis durant ce projet et la manière avec laquelle nous les avons réalisé. Dans le cadre d'un projet, nous avons été sollicités pour améliorer une boîte à outils (*ToolBox*) qui calcule des distances entre deux séquences selon différents algorithmes de correspondance.

Nous pouvons donc nous demander : Comment ce projet a-t-il été abordé, quels étaient les objectifs initiaux et quel est le résultat ?

Pour pouvoir y répondre, nous introduirons en premier temps le projet, son contexte et son déroulement. En second temps nous verrons un des points importants du projet : les types de séquences à mettre en correspondance. Puis nous étudierons les nouveaux algorithmes à intégrer à la *ToolBox*. Enfin nous étudierons la *ToolBox* à améliorer et donnerons les modifications apportées à la *ToolBox* et les résultats obtenus. En dernier lieu, nous donnerons les divers problèmes rencontrés durant l'année.

REMERCIEMENTS

Je tiens à remercier :

— Nicolas RAGOT, Maître de Conférence

— Tanmoy MONDAL, Doctorant

pour leurs aides et le suivi qu'ils ont effectué durant l'année, et le temps qu'ils ont pris pour m'expliquer les différents points du projet que je comprenais mal.

Je voudrais également remercier tous les élèves ayant effectué des projets de correspondance au préalable et les chercheurs du laboratoire RFAI dont les rapports ont permis ma compréhension et la réalisation du projet.

Enfin je voudrais remercier tous les autres étudiants de la salle de travail pour leurs aides et la bonne ambiance, ce qui a favorisé la réalisation de ce projet dans de bonnes conditions.

PROJET

1.1 Contexte

Depuis plusieurs années, de grandes quantités de documents d'archive sont numérisés et mis en ligne pour un accès public et dans une optique de conservation du patrimoine. Ces documents, imprimés ou manuscrits, sont souvent dégradés. De ce fait, la recherche dans la majeure partie de ces documents ne peut se faire que par feuilletage.

L'équipe RFAI du Laboratoire LI EA 6300 travaille sur des outils permettant de faciliter l'accès à ces documents sans passer par une phase de transcription. En particulier, dans le cadre d'un projet Franco-Indien, une plateforme de "wordspotting" est mise en place.

Le wordspotting repose sur la comparaison directe d'images de texte afin de trouver toutes les occurrences d'une requête. Cette requête est soit une image de mots issue de l'ouvrage, soit une image synthétisée provenant d'une requête composée à partir d'un clavier virtuel reposant sur un alphabet issu de l'ouvrage. Cette méthode permet donc en théorie de rechercher des noms propres ou des mots usuels dans n'importe quelle langue dès lors que l'on peut composer graphiquement l'image du mot recherché dans le style des documents souhaités.

Le projet s'inscrit dans le cadre du PFE de dernière année d'école ingénieur : Polytech Tours. Il a pour but de travailler sur les méthodes de spotting au sein d'images en étudiant de nouvelles techniques de correspondances.

Ce PFE fait suite à celui d'Abdourahman Aden Hassan : "Matching élastique et robuste de séquence". Il est en lien avec le PFE de Loreen Lambin : "Recherche par mots-clés "image" dans des masses d'images de documents : plateforme" qui intégrera la *ToolBox* à son projet.

1.2 Objectif

Le but est d'étudier les méthodes de spotting dans une masse de documents images.

L'objectif est d'améliorer une *ToolBox* créée par Abdourahman Aden Hassan. Cette *ToolBox* implémente déjà quatre algorithmes de correspondances entre des séquences numériques ou alpha-numérique :

- L'algorithme de Levenshtein
- L'algorithme du Longest Common SubSequences (LCS ou LCSS)
- L'algorithme du Dynamic Time Warping (DTW)
- L'algorithme du Minimum Variance Matching (MVM)

Trois autres méthodes de correspondances ont été étudiées afin d'obtenir des algorithmes fonctionnels à la *ToolBox*. Ils seront ensuite codés. Ces trois méthodes sont :

- la méthode du Continuous Dynamic Programming (CDP)
- la méthode du Flexible Sequence Matching (FSM)
- la méthode du Exemplary Sequence Cardinality (ESC)

De plus, un des autres objectifs est d'améliorer la *ToolBox* afin que ses fichiers d'entrées et ses fichiers de résultats soient facilement exploitable par l'application de L. Lambin

Enfin chaque algorithme doit pouvoir être paramétrable afin de pouvoir obtenir toutes les distances et mises en correspondances voulues.

1.3 Méthodologie

Le projet sera géré avec la plateforme Redmine et un dépôt SVN sera utilisé pour le versionning. Le déroulement du projet sera fait similairement à la méthode agile RUP. Ainsi il sera découpé en itération comprenant des sprints d'analyses, de développements et de tests.

A la fin de chaque itération, une implémentation de la Toolbox sera opérationnelle et un rapport de cycle sera fait.

Toutes les documentations seront faites en parallèle du développement et n'appartiendront à aucune itération.

Le langage de développement sera le C++, avec des commentaires écrits selon la norme Doxygen. Les rapports seront effectués avec Latex.

Tout au long du projet deux IDE seront utilisés :

- Visual Studio 2012, pour le développement sur Windows
- CodeBlocks, pour le développement sur Linux

1.4 Planning

Au départ, nous avons décidé de créer 3 itérations :

- CDP : cette itérations comprenait l'analyse de la *ToolBox*, du CDP et l'implémentation et les tests du CDP.
- FSM : cette itération comprenait l'analyse du FSM et son implémentation.
- Integration : cette itération permettait l'intégration dans l'application de L. Lambin et les tests sur la plateforme.

Voici le planning prévisionnel créé et donné dans le cahier de spécification :

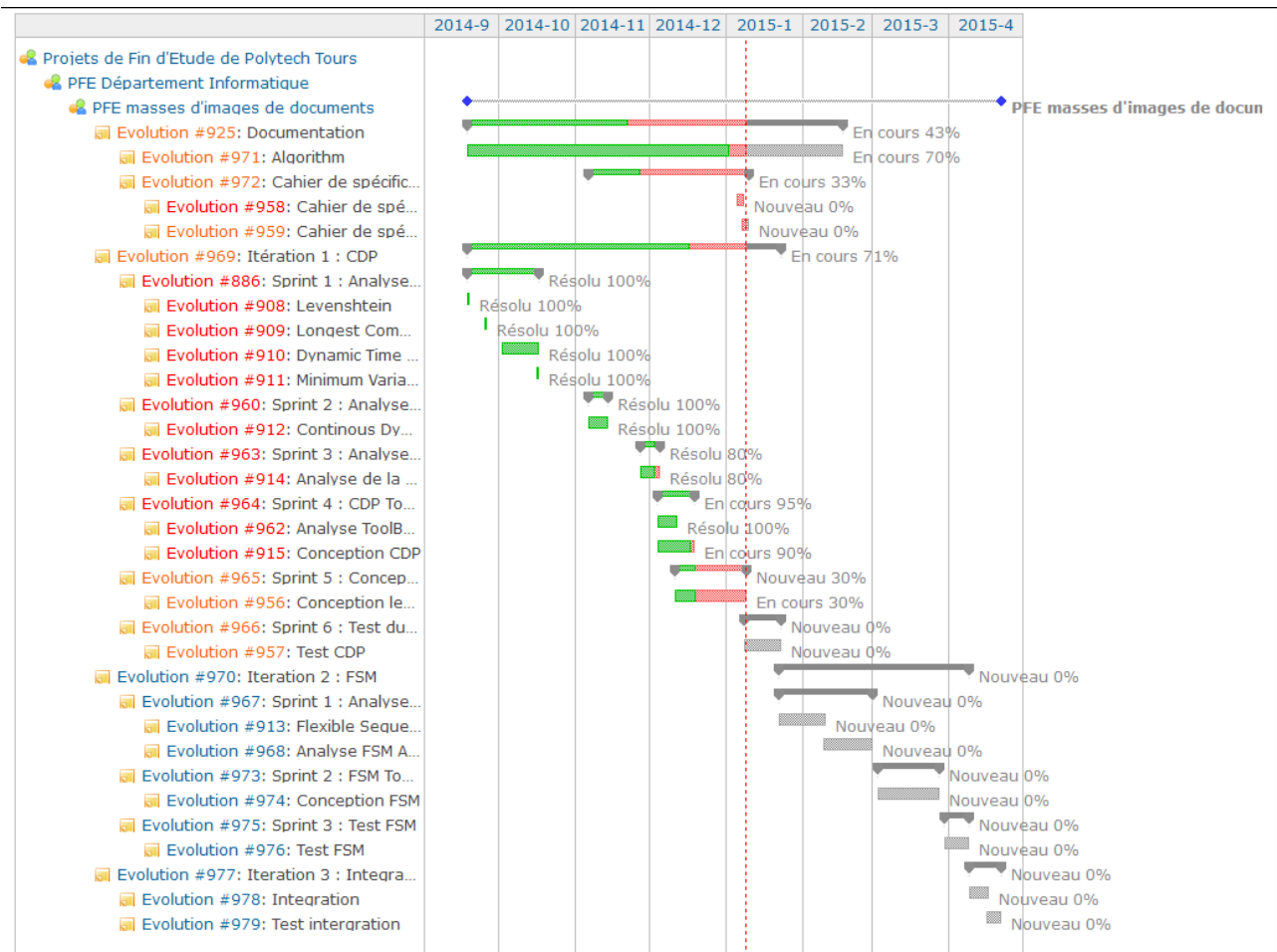


FIGURE 1.1 – Planning prévisionnel

Au final, le projet s'est déroulé similairement au planning. Cependant plusieurs grosses différences sont apparues.

- La première est un allongement du temps de la première itération.
- La seconde est la modification du dernier sprint, l'intégration dans la plateforme a été faite par L. Lambin, ainsi, un troisième algorithme a été rajouté à la place.
- La troisième plusieurs sprints se sont rajoutés lors des différentes phases afin de répondre aux différentes attentes du moment.

Voici le planning final :

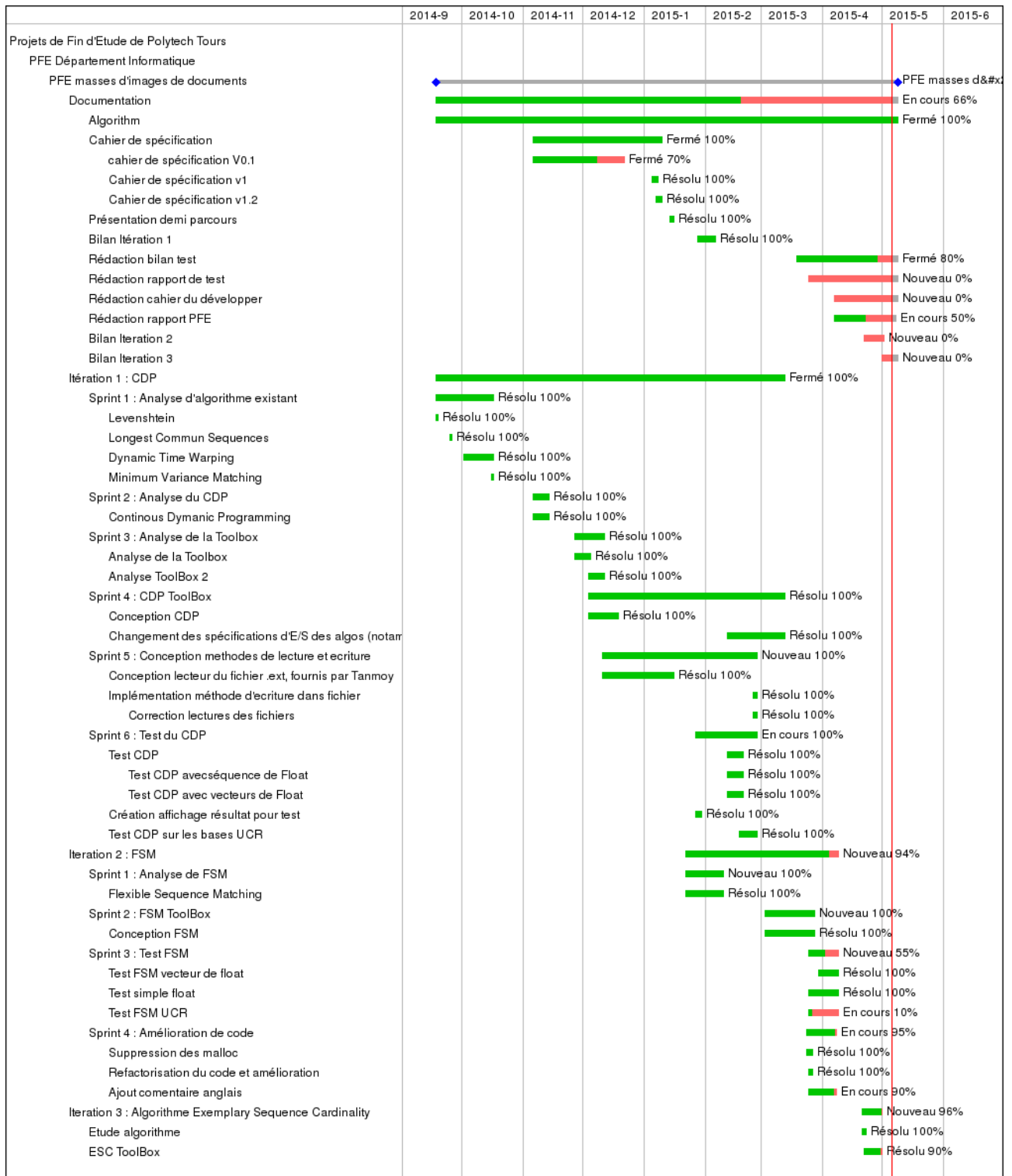


FIGURE 1.2 – Planning final

SEQUENCES

2.1 Définition et distance

Les séquences sont des suites d'éléments représentant un objet. Mettre en correspondance une séquence cible avec une séquence de référence c'est faire correspondre les éléments de la référence avec les éléments de la cible qui lui ressemblent le plus.

La distance entre deux séquences est un calcul qui permet de donner un ordre d'idées sur la ressemblance de deux séquences. Cette distance n'est pas toujours euclidienne et est différente selon l'algorithme de mise en correspondance utilisé.

2.2 Type de séquences

Tous au long du projet, nous avons utilisé deux types de séquences.

2.2.1 Séquences de flottant simple

Le premier type de séquence utilisé est la séquence simple de flottant. Dans ce cas ci, chaque élément qui constitue une séquence est un nombre de type flottant.

Voici un exemple de séquence de ce type :

$A = (1, 2, 8, 16, 16)$

La distance entre deux éléments (i, j) de deux séquences A et B de ce type est :

$$d = |A[i] - B[j]|$$

2.2.2 Séquences de flottant vecteur

Le second type de séquence utilisé est la séquence de vecteur de flottant. Dans ce cas ci, chaque élément qui constitue une séquence est un vecteur de nombres flottant.

Voici un exemple de séquence de ce type : $A = \left(\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}, \begin{pmatrix} 12 \\ 15 \\ 18 \end{pmatrix} \right)$

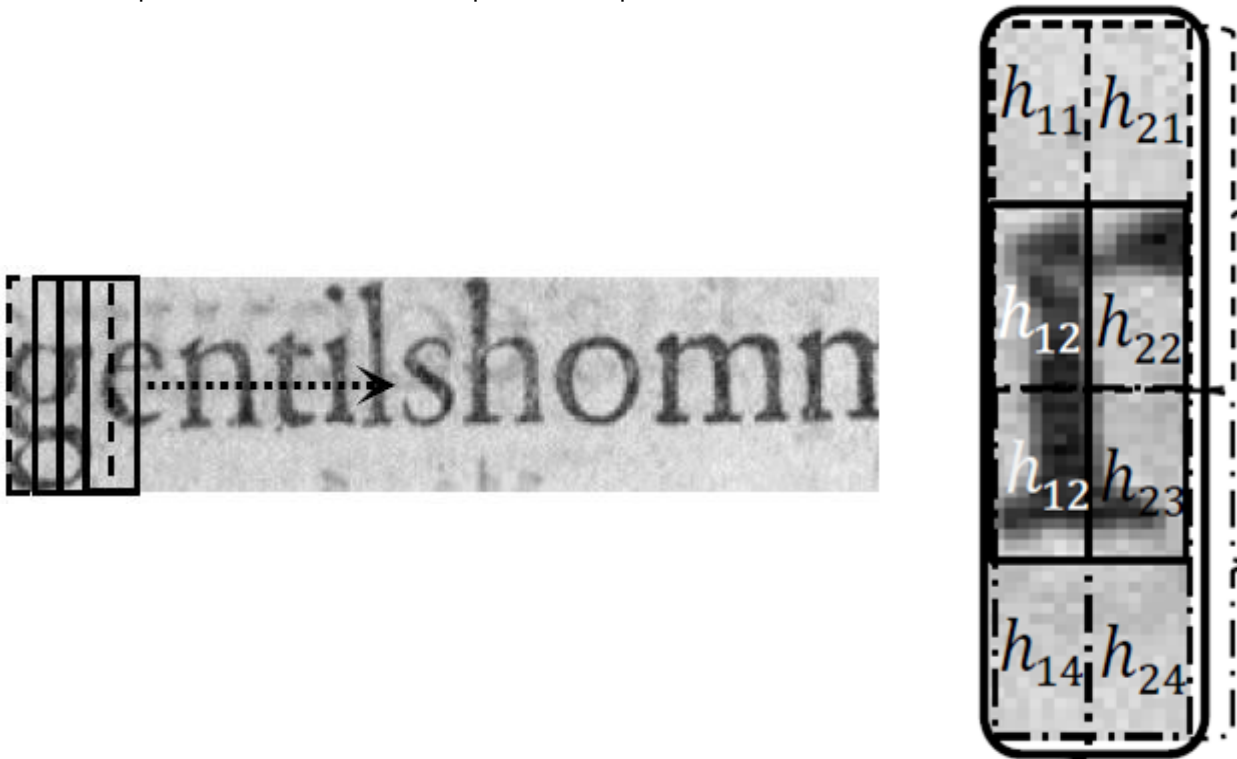
La distance entre deux éléments (i, j) de deux séquences A et B de ce type est :

$$d = \sqrt{\sum_{k=0}^n (A[i, k] - B[j, k])^2}$$

2.3 Obtention des séquences

Le but de notre projet est d'implémenter des méthodes de *wordspotting*. Cependant, afin de pouvoir mettre en correspondance deux images de mots, il faut pouvoir obtenir deux séquences numériques représentant chacune des images. Cela se fait en trois étapes :

- Les deux images sont ramenées à la même hauteur tout en gardant les bonnes proportions sur la longueur.
- chaque image est alors découpée en lamelles verticales homogènes
- sur chaque lamelle, des caractéristiques numériques sont calculées.



Ainsi lors du *wordspotting*, seul le type vecteur de flottant est utilisé.

Dans le projet, le type un n'a été utilisé que pour tester l'algorithme sur de petites séquences.

2.4 Formatage des séquences

Une fois les caractéristiques extraites, il faut les écrire dans un fichier avec un format lisible par la *ToolBox*. Notre *ToolBox* peut lire trois types de format.

2.4.1 Simple

Le premier type de format correspond au premier type de séquences.

Dans ce format, chaque ligne correspond à une séquence. Chaque élément constituant la séquence est séparé par une virgule.

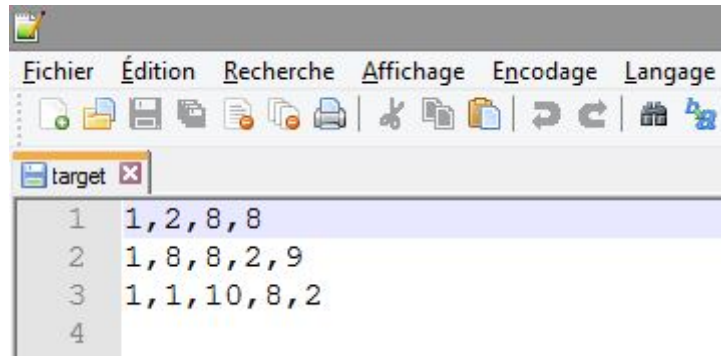


FIGURE 2.1 – premier format texte de séquences

Dans notre exemple, il y aurait donc 3 séquences A, B et C :

A = (1, 2, 8, 8)

B = (1, 8, 8, 2, 9)

C = (1, 1, 10, 8, 2)

2.4.2 Vecteur

Le second type de format correspond au deuxième type de séquence.

Dans ce format, l'ensemble du fichier correspond à une seule séquence. Chaque ligne correspond à un élément de la séquence et chaque élément est un vecteur. Tous les éléments ont la même taille. Les composantes du vecteur sont séparées par une virgule.

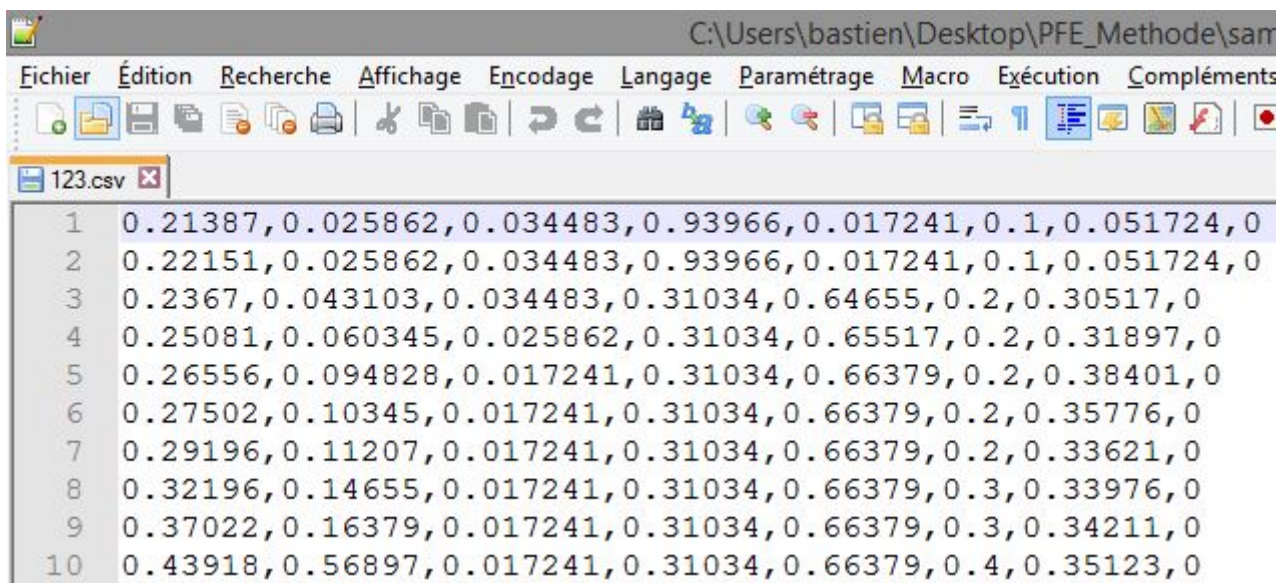


FIGURE 2.2 – second format texte de séquences

Dans l'ensemble de nos fichiers de vecteurs, ces derniers comprennent 8 composantes.

Notre exemple est un extrait d'un fichier représentant une image. La séquence originale comprend 367 éléments. La séquence est donc :

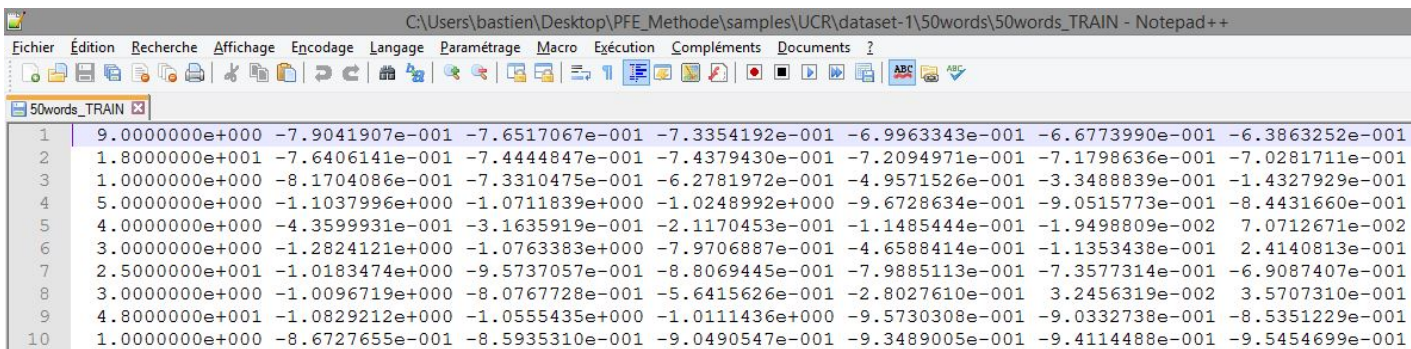
$$A = \left(\begin{pmatrix} 0.21387 \\ 0.025862 \\ 0.034483 \\ 0.93966 \\ 0.017241 \\ 0.1 \\ 0.051724 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.22151 \\ 0.025862 \\ 0.034483 \\ 0.93966 \\ 0.017241 \\ 0.1 \\ 0.051724 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.2367 \\ 0.043103 \\ 0.034483 \\ 0.31034 \\ 0.64655 \\ 0.2 \\ 0.30517 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.25081 \\ 0.060345 \\ 0.025862 \\ 0.31034 \\ 0.65517 \\ 0.2 \\ 0.31897 \\ 0 \end{pmatrix}, \dots \right)$$

2.4.3 UCR

Le dernier format est un format de séquences de flottant simple utilisé pour calculer un taux d'erreur. Ces fichiers sont des séquences temporelles (l'évolution d'un objet au cours du temps) permettant de tester la justesse d'une méthode de correspondance.

Chaque ligne correspond à une séquence. Cependant, contrairement au premier format, nous observons deux différence :

- Le premier nombre de la ligne correspond à un numéro de classe à laquelle appartient la séquence.
- Chaque élément de la séquence est séparé par un ou plusieurs espaces.



Line	Class	Element 1	Element 2	Element 3	Element 4	Element 5	Element 6	Element 7	Element 8	Element 9	Element 10
1	9	0.000000e+000	-7.9041907e-001	-7.6517067e-001	-7.3354192e-001	-6.9963343e-001	-6.6773990e-001	-6.3863252e-001			
2	1	8.000000e+001	-7.6406141e-001	-7.4444847e-001	-7.4379430e-001	-7.2094971e-001	-7.1798636e-001	-7.0281711e-001			
3	1	0.000000e+000	-8.1704086e-001	-7.3310475e-001	-6.2781972e-001	-4.9571526e-001	-3.3488839e-001	-1.4327929e-001			
4	5	0.000000e+000	-1.1037996e+000	-1.0711839e+000	-1.0248992e+000	-9.6728634e-001	-9.0515773e-001	-8.4431660e-001			
5	4	0.000000e+000	-4.3599931e-001	-3.1635919e-001	-2.1170453e-001	-1.1485444e-001	-1.9498809e-002	7.0712671e-002			
6	3	0.000000e+000	-1.2824121e+000	-1.0763383e+000	-7.9706887e-001	-4.6588414e-001	-1.1353438e-001	2.4140813e-001			
7	2	5.000000e+001	-1.0183474e+000	-9.5737057e-001	-8.8069445e-001	-7.9885113e-001	-7.3577314e-001	-6.9087407e-001			
8	3	0.000000e+000	-1.0096719e+000	-8.0767728e-001	-5.6415626e-001	-2.8027610e-001	3.2456319e-002	3.5707310e-001			
9	4	8.000000e+001	-1.0829212e+000	-1.0555435e+000	-1.0111436e+000	-9.5730308e-001	-9.0332738e-001	-8.5351229e-001			
10	1	0.000000e+000	-8.6727655e-001	-8.5935310e-001	-9.0490547e-001	-9.3489005e-001	-9.4114488e-001	-9.5454699e-001			

FIGURE 2.3 – troisième format texte de séquences

L'exemple est un extrait d'un fichier contenant 450 séquences possédant chacune 270 éléments et 50 classes différentes.

ÉTUDE ALGORITHMIQUE

3.1 Introduction

Dans cette partie, nous allons nous intéresser aux trois algorithmes étudiés. Elle n'a pas pour but de d'expliquer les algorithmes mais de les présenter et d'introduire leur pseudo-codes comme je les ai codés. Chaque méthode se découpera en trois parties, une introduction, une implémentation et un exemple.

Plusieurs fonctions et une structure seront utilisées dans les différents algorithmes. Voici les prototypes de ces fonctions et de la structures :

3.1.1 Structure de résultat

struc Resultat :

- *float* distance : la distance entre les séquences
- *List de int* correspondanceR : les indices de la séquences cible qui correspond avec la séquence référence
- *List de int* correspondanceC : les indices de la séquences références qui corresponde avec la séquence cible

Prenons un exemple :

- Reference (R) = (1,2,3)
- Cible (C) = (5,6,8,1,2,3,5,9,7)

Le résultat *res* de la mise en correspondance selon l'une des méthodes (ici CDP) sera :

- *res.distance* = 0
- *res.correspondanceR* = (3,4,5)
- *res.correspondanceC* = (0,1,2)

Ainsi les correspondances seront : R(0)->C(3), R(1)->C(4) et R(2)->C(5) (respectivement le 1 avec le 1, le 2 avec le 2 et le 3 avec le 3).

3.1.2 Fonctions nécessaires

1. calcDistance

Cette fonction calcule la matrice des distances euclidiennes entre chaque élément des deux séquences.

Procedure 1 calcDistance

Input: $R[1..T]$, $C[1..M]$ \leftarrow Séquences : les séquences à comparer, R séquence de référence, C séquence cible

Output: tabDist : \leftarrow tableau de float : tableau des distances

2. distance

Cette fonction récupère la distance euclidienne entre un éléments de chaque séquences et les affecte d'une pondération paramétrable.

Procedure 2 distance

Input:

- tabDist : *tableau de float* : tableau des distances
- indice1, indice2 : *int*, deux indice des séquences

Output: dist : *float*, la distance pondérer par les poids donnés en paramètres

3. getIndiceOfMinColumn

Cette fonction récupère l'indice de la colonne, d'un tableau, ayant la dernière plus petite valeur sur la dernière ligne.

Procedure 3 getIndiceOfMinColumn

Input: A : *tableau de int*, tableau contenant les distances entre deux séquences

Output: j_min : *tableaudeint*, tableau contenant l'indice colonne

4. getIndiceOfMinRow

Cette fonction récupère tous les indices des ligne, d'un tableau, où la valeur sur la dernière colonne est :

- Soit le minimum.
- Soit inférieur à un seuil.

Procedure 4 getIndiceOfMinRow

Input: A : *tableau de int*, tableau contenant les distances entre deux séquences

Output: i_min : *tableaudeint*, un tableau contenant les indice ligne

5. getElasticity

Cette fonction calcule l'élasticité nécessaire à certains des algorithmes pour rendre un bon résultat. Cette valeur correspond à la valeur absolue de la différence entre les tailles des séquences. Elle peut cependant être paramétrable.

Procedure 5 getElasticity

Input:

- M : taille d'une séquences
- N : taille de l'autre séquences

Output: elasticity : *int*, l'élasticité calculée ($|M-N|$) ou donnée par les paramètres

6. getSkipCost

Cette fonction calcule le coût pour éviter un élément lors d'une mise en correspondance. Sa valeur est calculée selon la méthode suivante :

- On trouve le tableau *vmin* : moyenne des X minimums de chaque ligne du tableau des distances
- On calcule : $\text{skipCost} = \text{mean}(\text{vmin}) + Y * \text{std}(\text{vmin})$

où

- X est le nombre de minimums souhaité pour effectuer le calcul, cette valeur est paramétrable. Par défaut $X = 2$.
- Y est le nombre d'écart type voulue, il est paramétrable. Par défaut $Y = 2$.

De plus le skipCost peut être lui même paramétrable.

Procedure 6 getSkipCost

Input:

- tabDist : *tableau de float* : tableau des distances
- M, T : tailles des séquences

Output: skipCost : *int*, le coût pour grand saut calcule ou donné par les paramètres

7. getSmallSkipCost

Cette fonction calcule le coût pour mettre en correspondance deux éléments voisins avec le même élément, cette valeur est paramétrable. Le coût est calculé selon la méthode suivante :

- On trouve le tableau *vmin* : moyenne des X minimums de chaque ligne du tableau des distances
- On calcule : $\text{skipCost} = \text{mean}(\text{vmin})$

Procedure 7 getSmallSkipCost

Input:

- tabDist : *tableau de float* : tableau des distances
- M, T : tailles des séquences

Output: smallSkipCost : *int*, la valeur du coût d'un petit saut calculé ou donné par les paramètres

8. getWeight

Cette fonction permet de donner le poids d'un saut d'élément dans une mise en correspondance, cette valeur est paramétrable.

Procedure 8 getWeight

Input:**Output:** $W : int$, le poids (1 par défaut) du paramètre

3.2 Continuous Dynamic Programming

3.2.1 Présentation

L'algorithme CDP est un algorithme présenté par Ryuichi Oka dans un article Spotting Method for Classification of Real World Data.

Le but du CDP est de mettre en correspondance une séquence de référence $R[1..M]$ avec une séquence cible $C[1..T]$ plus grande; il cherche la référence dans la cible. Pour cela, il calcule les distances entre la séquence de référence et des sous-séquences de la cible. Lorsque la sous-séquence correspond à la cible, il cherche toutes les sous-séquences dont la distance est inférieure à un certain seuil. Les sous-séquences trouvées sont celles qui ressemblent le plus à la référence.

L'avantage du CDP, c'est qu'il permet sauter autant d'éléments nécessaire au début et à la fin afin de trouver la sous-séquence optimale. Cependant le CDP ne fait que de la correspondance un à un, un élément de la référence est forcément mis en correspondance avec un élément de la cible. Certains éléments perturbateurs peuvent donc fausser un résultat. Voici l'équation du CDP :

$$d(t, \tau) = |C[t] - R[\tau]|$$

$$P(t, \tau \geq 3) = \min \begin{cases} P(t-2, \tau-1) + 2.d(t-1, \tau) + d(t, \tau) & \text{si } t \geq 2, \text{ sinon } +\infty \\ P(t-1, \tau-1) + 3.d(t, \tau) \\ P(t-1, \tau-2) + 3.d(t, \tau-1) + 3.d(t, \tau) \end{cases} \quad \text{avec } P(-1, \tau) = P(0, \tau) = +\infty$$

où

$$P(t, 1) = 3.d(t, 1)$$

$$P(t, 2) = \min \begin{cases} P(t-2, 1) + 2.d(t-1, 2) + d(t, 2) & \text{si } t \geq 2, \text{ sinon } +\infty \\ P(t-1, 1) + 3.d(t, 2) \\ P(t, 1) + 3.d(t, 2) \end{cases}$$

Pour pouvoir obtenir le coût minimum il faut normaliser les $P(t, T)$.

$$\text{Le coût } A(t, M) = \frac{P(t, M)}{3.M}$$

3.2.2 Pseudo-code

Input:

- $R[1..M], C[1..T] \leftarrow \text{Séquences}$: les séquences à comparer, R séquence de référence, C séquence cible
- $T, M \leftarrow \text{Entier}$: les tailles respectives des séquences
- **if** $T < M$ **then**
- echanger(R, C);
- echanger(T, M);
- **end if**

Output: resultat $\leftarrow \text{tableau de Resultat}$: tous les résultats du CDP**Begin**

$P[-1..T][1..M], \text{tabDist}[1..T][1..M] := \text{tableau de float};$

```

t,  $\tau$ , k, i := Entier;
resultat := tableau Resultat;
i_min := tableau d'entier;
tabDist  $\leftarrow$  calcDistance(R,C);
for  $\tau = 1$ ;  $\tau \leq M$ ;  $\tau++$  do
    P[1][ $\tau$ ]  $\leftarrow +\infty$ ;
    P[2][ $\tau$ ]  $\leftarrow +\infty$ ;
end for
k  $\leftarrow 3$ ;
for t = 1; t  $\leq T$ ; t++ do
    for  $\tau = 1$ ;  $\tau \leq M$ ;  $\tau++$  do
        if  $\tau == 1$  then
            P[k][2]  $\leftarrow$  3.distance(tabDist,t,1);
        else if  $\tau == 2$  then
            P[k][2]  $\leftarrow$  min(P[k-2][1] + 2.distance(tabDist,t-1,2) + distance(tabDist,t,2), P(k-1,1) + 3.distance(tabDist,t,2), P(k,1) + 3.distance(tabDist,t,2));
        else
            P[k][ $\tau$ ]  $\leftarrow$  min(P[k-2][ $\tau-1$ ] + 2.distance(tabDist,t-1, $\tau$ ) + distance(tabDist,t, $\tau$ ), P(k-1, $\tau-1$ ) + 3.distance(tabDist,t, $\tau$ ), P(k-1, $\tau-2$ ) + 3.distance(tabDist,t, $\tau-1$ ) + 3.distance(tabDist,t, $\tau$ ));
        end if
    end for
end for
i_min  $\leftarrow$  getIndiceOfMinRow(P,T,M);
t  $\leftarrow 0$ ;
for i = 1; i  $\leq$  i_min.size(); i++ do
    res := Resultat;
    for  $\tau = 0$ ;  $\tau < T$ ;  $\tau++$  do
        if  $\tau > i\_min - M$  ET  $\tau \leq i\_min[i]$  then
            res.correspondanceR.push( $\tau$ );
            res.correspondanceC.push(t++);
        end if
    end for
    res.distance  $\leftarrow$  P[i_min[i]][M]/(3*M);
    resultat.push(res);
end for
return resultat;
End

```

▷ Initialisation de la matrice des coût

▷ Début algorithme

▷ Début BackTracking

▷ Récupération indice ligne de la plus petite valeur de la dernière colonne

▷ Calcul de la distance

3.2.3 Exemple

- Séquence référence : (1,2,8,16,16)
- séquence cible : (1,2,9,16,9,25,37)

Déroulons rapidement l'algorithme :

1ère étape : Calculer le tableau des distances

	1	2	8	16	16
1	0	1	7	15	15
2	1	0	6	14	14
9	8	7	1	7	7
16	15	14	8	0	0
9	8	7	1	7	7
25	24	23	17	9	9
37	36	35	29	21	21

FIGURE 3.1 – CDP : Calcule distance

2nd étape : Calculer la matrice de coût

	1	2	8	16	16
	∞	∞	∞	∞	∞
	∞	∞	∞	∞	∞
1	0	3	∞	∞	∞
2	3	0	18	63	∞
9	24	7	3	24	60
16	45	31	10	3	3
9	24	45	24	10	24
25	72	82	50	33	26
37	108	105	108	63	49

FIGURE 3.2 – CDP : Calcule des coûts

3ème étape : Trouver l'indice de correspondance minimum possible.

La valeur minimum à partir de la 5ème position est 24, son indice est 5.

4ème étape : Effectuer un *backtracking* sur la matrice (bleu)

	1	2	8	16	16
	∞	∞	∞	∞	∞
	∞	∞	∞	∞	∞
1	0	3	∞	∞	∞
2	3	0	18	63	∞
9	24	7	3	24	60
16	45	31	10	3	3
9	24	45	24	10	24
25	72	82	50	33	26
37	108	105	108	63	49

FIGURE 3.3 – CDP : Backtrack

5me étape Calculer la distance : $24/(3*5) = 1.6$

3.3 Flexible Sequences Matchnig

3.3.1 Présentation

Le FSM est une méthode découverte par le Laboratoire Informatique de l'Université François Rabelais de Tours en collaboration avec l'entité Computer Vision and Pattern Recognition de l'institut indien de statistique.

Le Flexible Sequences Matching est prévu pour pouvoir agir similairement à plusieurs algorithmes connus, tel que le DTW, le MVM et le CDP. Ainsi, il s'inspire de l'architecture de ces algorithmes pour pouvoir obtenir une correspondance entre deux séquences. Il arrive donc faire des correspondances un à un, un à plusieurs et plusieurs à un.

L'avantage du FSM, c'est qu'il permet d'éliminer l'influence d'éléments glissant (dans le temps) de la séquences cible en ayant la possibilité de sauter ces éléments.

Le principes du FSM est simple à comprendre. Pour chaque élément on lui définit des enfants, plus ou moins loin de lui. Le nombre de ses enfants est défini grâce à une valeur appelée l'élasticité. Le graphe suivant récapitule cette étape.

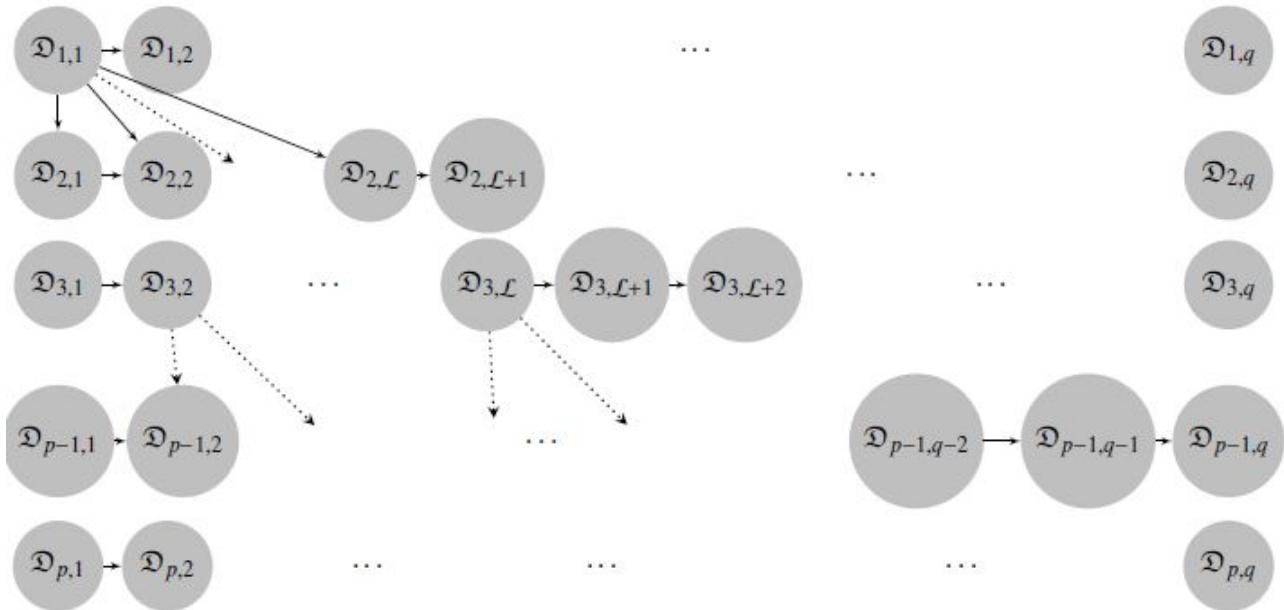


FIGURE 3.4 – FSM DAG

Maintenant, lors du calcul des coûts de correspondance, il faut prendre le parent dont le coût est minimum. L'équation permettant de gérer le coût et les parents est la suivante :

$$P(1,j) = D(i,j) \quad \forall j \in [1, T] :$$

$$\forall i \in [2, M] :$$

$$P(i,j) = \min \left\{ \begin{array}{l} P(i-1,k) + D(i,j) + \text{skipCost} * W * (j-(k+1)) \\ P(i,j-1) + \text{smallSkipCost} + D(i,j) \\ P(i-1,j) + \text{smallSkipCost} + D(i,j) \end{array} \right\}$$

si $1 \leq (i-1 - \text{elasticity}) \leq k \leq (i-1 + \text{elasticity}) \leq T$

et si $k \leq j \leq k+1 + \text{elasticity} + \max(1, i-1 + \text{elasticity}) \leq T$

où $D(i,j)$ est la distance entre les éléments i et j , P la matrice des coûts. SkipCost et smallSkipCost sont des poids pour faire certaines correspondances :

- smallSkipCost est le coût donné si le parent est un voisin direct, c'est à dire l'élément juste au dessus ou l'élément juste à gauche dans le graphe.
- skipCost est le coût donné si le parent n'est pas voisin ; si plus de 1 élément les sépare.

3.3.2 Pseudo-code

Input:

- $R[1..M]$, $C[1..T] \leftarrow \text{Séquences}$: les séquences à comparer, R séquence de référence, C séquence cible
- T , $M \leftarrow \text{Entier}$: les tailles respectives des séquences $M < T$

Output: $\text{resultat} \leftarrow \text{tableauResultat}$: le résultat du FSM

Begin

```

P[1..M][1..T], tabDist[1..M][1..T] := tableau de float
i, j, k, L, M, BcS, tmp, nbJump := Entier ;
skipCost, smallSkipCost, skipValue, elasticity, W := Float ;
resultat := tableau de Resultat ;
pathRow[1..M][1..T], pathCol[1..M][1..T], j_min := tableau d'entier ;

```

▷ Initialisation des distances, poids, élasticité et skipCost

```

tabDist ← calcDistance(R,C) ;
elasticity ← getElasticity(M,T) ;
skipCost ← getSkipCost(tabDist,M,T) ;
smallSkipCost ← getSmallSkipCost(tabDist,M,T) ;
W ← getWeight() ;

```

▷ Initialisation des matrice des coût et des matrices de backtracking

```

for i = 1 ; i ≤ M ; i++ do
  for j = 1 ; j ≤ T ; j++ do
    P[i][j] ← + ∞ ;
    pathRow[i][j] ← + ∞ ;
    pathCol[i][j] ← + ∞ ;
  end for
end for

```

▷ Début de l'algorithme

```

P[1][1] ← distance(tabDist,1,1) ;

```

▷ Verification de correspondance un à plusieurs pour la première ligne

```

for j = 2 ; j ≤ T ; j++ do
  P[1][j] ← distance(tabDist,1,j) ;
  if P[1][j] ≥ P[1][j-1] + distance(tabDist,1,j) then

```

```

    P[1][j] ← P[1][j-1] + distance(tabDist,1,j);
    pathRow[1][j] ← 1;
    pathCol[1][j] ← j-1;
  end if
end for

                                ▷ Calcule des matrices de coût et de chemin

for i = 2; i ≤ M; i++ do
  if i == 2 then
    L ← 1;
    R ← T;
  else
    L ← max(1, i-1 - elasticity);
    R ← min(T, i-1 + elasticity);
  end if
  for k = L; k ≤ R; k++ do
    BcS ← min(T, k + 1 + elasticity - max(1, k - i + 1));
    for j = k; j ≤ BcS; j++ do
      if j = k then
        skipValue ← smallSkipCost;
      else if j = k + 1 then
        skipValue ← 0;
      else
        skipValue ← skipCost * W * |j - k - 1|;
      end if
      if P[i][j] > P[i-1][k] + distance(tabDist,i,j) + skipValue then
        P[i][j] ← P[i-1][k] + distance(tabDist,i,j) + skipValue;
        pathRow[i][j] ← i - 1;
        pathCol[i][j] ← k;
      end if
      if P[i][j] > P[i][j-1] + distance(tabDist,i,j) + smallSkipCost then
        P[i][j] ← P[i][j-1] + distance(tabDist,i,j) + smallSkipCost;
        pathRow[i][j] ← i;
        pathCol[i][j] ← j-1;
      end if
    end for
  end for
end for

                                ▷ Début du BackTracking  ▷ Récupération de l'indice de la plus petite valeur de la dernière ligne
j_min ← getIndexOfMinColumn(P,M,T);
for k = 1; k ≤ j_min.size(); k++ do
  res := Resultat;
  i ← M;
  j ← j_min[k];
  tmp ← j;
  nbJump ← 0;
  while (i ≥ 1) ET (j ≥ 1) ET (j ≤ T) do
    res.correspondanceR.insert(1,i);
    res.correspondanceC.insert(1,j);
    if (i = 1) ET (j = 1) then
      break;
    end if
  end while
end for

```



```

end if
tmp ← pathCol[i][j];
i ← pathRow[i][j];
if |j-tmp| == 0 then
    nbJump ← nbJump + 1;
elseif tmp ≠ ∞
    nbJump ← nbJump + |j-tmp|;
end if
j ← tmp;
end while
res.distance ← (P[M][j_min[k]])/(res.correspondanceR.size());
resultat.push(res);
end for
return resultat;
End

```

▷ Calcul de la distance

3.3.3 Exemple

— Séquence référence : (1,2,8,16,16)

— séquence cible : (1,2,9,16,9,25,37)

Déroulons rapidement l'algorithme :

1ère étape : Calculer le tableau des distances

	1	2	9	16	9	25	37
1	0	1	8	15	8	24	36
2	1	0	7	14	7	23	35
8	7	6	1	8	1	17	29
16	15	14	7	0	7	9	21
16	15	14	7	0	7	9	21

FIGURE 3.5 – FSM : Calcule distance

2nd étape : Calculer les skipCost et autre valeurs paramétrable

— skipCost = 4.6

— smallSkipCost = 1.8

— élasticité = 2

3ème étape : Calculer les matrices de coût et de chemin

	1	2	9	16	9	25	37
1	0	1	8	15	8	24	36
2	2.8	0	8	19.6	∞	∞	∞
8	11.6	7.8	1	10.8	10.2	∞	∞
16	28.4	23.6	9.8	1	9.8	19.2	∞
16	∞	39.4	18.6	2.8	8	14.6	32.2
	1	2	9	16	9	25	37
1	∞	0	∞	∞	∞	∞	∞
2	0	0	0	0	∞	∞	∞
8	1	1	1	2	1	∞	∞
16	2	2	2	2	3	2	∞
16	∞	3	3	3	3	3	3
	1	2	9	16	9	25	37
1	∞	0	∞	∞	∞	∞	∞
2	0	0	1	1	∞	∞	∞
8	0	1	1	2	1	∞	∞
16	0	1	2	2	3	2	∞
16	∞	1	2	3	3	3	3

FIGURE 3.6 – FSM : Calcule des coûts

4ème étape : Effectuer un baktracking sur les matrices (bleu)

	1	2	9	16	9	25	37
1	0	1	8	15	8	24	36
2	2.8	0	8	19.6	∞	∞	∞
8	11.6	7.8	1	10.8	10.2	∞	∞
16	28.4	23.6	9.8	1	9.8	19.2	∞
16	∞	39.4	18.6	2.8	8	14.6	32.2

FIGURE 3.7 – FSM : Backtrack

5me étape Calculer la distance : $2.6/5 = 0.56$

3.4 Exemplary Sequences Cardinality

3.4.1 Présentation

Le ESC est un méthode découverte par le Laboratoire Informatique de l'Université François Rabelais de Tours en collaboration avec l'entité Computer Vision and Pattern Recognition de l'institut indien de statistique.

Le Exemplary Sequences Cardinality est prévu pour pouvoir agir similairement au FSM. Ainsi, il s'inspire fortement de l'architecture de cet algorithme afin d'obtenir une bonne correspondance entre deux séquences. Il peut donc faire de la correspondance un à un, un à plusieurs et plusieurs à un.

Le ESC comprend non seulement les avantages du FSM mais possède également la possibilité de sauter des éléments de la séquence de référence. Cela lui permet d'éliminer certains bruits de la séquence de référence.

Le principe du ESC est semblable à celui de FSM. Cependant, il est autorisé à sauter un élément parent de la référence si la distance entre le parent et l'enfant est supérieure à un coût de saut d'élément.

L'équation de l'ESC est la même que celle du FSM

3.4.2 Pseudo-code

Input:

- $R[1..M]$, $C[1..T] \leftarrow$ *Séquences* : les séquences à comparer, R séquence de référence, C séquence cible
- $T, M \leftarrow$ *Entier* : les tailles respectives des séquences $M < T$

Output: $resultat \leftarrow$ *tableau de Resultat* : le résultat du FSM

Begin

```

P[1..M][1..T], tabDist[1..M][1..T] := tableau de float
i, j, k, L, M, BcSL, BcSR, tmp, nbJump, i_first, j_first := Entier ;
skipCost, smallSkipCost, skipValue, elasticity, W := Float ;
first := bool ;
resultat := tableau de Resultat ;
pathRow[1..M][1..T], pathCol[1..M][1..T], skippable[1..M][1..T], j_min := tableau d'entier ;
                                ▷ Initialisation des distances, poids, élasticité et skipCost

```

```

tabDist  $\leftarrow$  calcDistance(R, C) ;
elasticity  $\leftarrow$  getElasticity(M, T) ;
skipCost  $\leftarrow$  getSkipCost(tabDist, M, T) ;
smallSkipCost  $\leftarrow$  getSmallSkipCost(tabDist, M, T) ;
W  $\leftarrow$  getWeight() ;

```

▷ Initialisation des matrices des coûts et des matrices de backtracking

```

for i = 1 ; i ≤ M ; i++ do
  for j = 1 ; j ≤ T ; j++ do
    P[i][j]  $\leftarrow$  + ∞ ;
    pathRow[i][j]  $\leftarrow$  + ∞ ;
    pathCol[i][j]  $\leftarrow$  + ∞ ;
    skippable[i][j]  $\leftarrow$  + ∞ ;
  end for
end for

```

▷ Début de l'algorithme

▷ Test si l'élément peut être sauté

```

for i = 1 ; i ≤ M ; i++ do
  for j = 1 ; j ≤ T ; j++ do
    skippable[i][j]  $\leftarrow$  1 ;
    if distance(tabDist, i, j) > skipCost then
      tabDist[i][j]  $\leftarrow$  skipCost ;
      skippable[i][j]  $\leftarrow$  -1 ;
    end if
  end for
end for
P[1][1]  $\leftarrow$  distance(tabDist, 1, 1) ;

```

▷ Vérification de correspondance un à plusieurs pour la première ligne

```

for j = 2 ; j ≤ T ; j++ do
    P[1][j] ← distance(tabDist,1,j) ;
    if P[1][j] ≥ P[1][j-1] + distance(tabDist,1,j) ET skippable[1][j-1] == 1 then
        P[1][j] ← P[1][j-1] + distance(tabDist,1,j) ;
        pathRow[1][j] ← 1 ;
        pathCol[1][j] ← j-1 ;
    end if
end for

```

▷ Calcule des matrice de coût et de chemin

```

for i = 2 ; i ≤ M ; i++ do
    if i == 2 then
        L ← 1 ;
        R ← T ;
    else
        L ← max(1, i-1 - elasticity) ;
        R ← min(T, i-1 + elasticity) ;
    end if
    for k = L ; k ≤ R ; j++ do
        BcSL ← max(k,1) ;
        BcSR ← min(T, k + 1 + elasticity - max(1, k - i + 1)) ;
        for j = BcSL ; j ≤ BcS ; j++ do
            if j = k then
                skipValue ← smallSkipCost ;
            else if j = k + 1 then
                skipValue ← 0 ;
            else
                skipValue ← skipCost*W*|j - k - 1| ;
            end if
            if P[i][j] > P[i-1][k] + distance(tabDist,i,j) + skipValue then
                P[i][j] ← P[i-1][k] + distance(tabDist,i,j) + skipValue ;
                if skippable[i-1][k] == 1 then
                    pathRow[i][j] ← i - 1 ;
                    pathCol[i][j] ← k ;
                else
                    pathRow[i][j] ← pathRow[i - 1][k] ;
                    pathCol[i][j] ← pathCol[i - 1][k] ;
                end if
            end if
            if j > 1 then
                if P[i][j] > P[i][j-1] + distance(tabDist,i,j) + smallSkipCost then
                    P[i][j] ← P[i][j-1] + distance(tabDist,i,j) + smallSkipCost ;
                    if skippable[i][j-1] == 1 then
                        pathRow[i][j] ← i ;
                        pathCol[i][j] ← j ;
                    else
                        pathRow[i][j] ← pathRow[i][j-1] ;
                        pathCol[i][j] ← pathCol[i][j-1] ;
                    end if
                end if
            end if
        end for
    end for

```

```

    end for
  end for
end for
  ▷ Début du BackTracking  ▷ Récupération de l'indice de la plus petite valeur de la dernière ligne
j_min ← getIndiceOfMinColumn(P,M,T);
for k=1; k < j_min.size();k++ do
  res := Resultat
  i ← M;
  j ← j_min[k];
  nbJump ← 0;
  tmp ← j;
  i_first = i;
  j_first = j;
  first = true;
  while (i ≥ 1) ET (j ≥ 1) ET (j ≤ T) do
    if first ET skippable[i][j] == -1 then
      tmp ← pathCol[i][j];
      i ← pathRow[i][j];
      j ← tmp;
      i_first ← i;
      j_first ← j;
    else
      res.correspondanceR.insert(1,i);
      res.correspondanceC.insert(1,j);
      if (i = 1) ET (j = 1) then
        break;
      end if
      tmp ← pathCol[i][j];
      i ← pathRow[i][j];
      if |j-tmp| == 0 then
        nbJump ← nbJump + 1;
      elseif tmp ≠ ∞
        nbJump ← nbJump + |j-tmp|;
      end if
      j ← tmp;
    end if
    first ← false;
  end while
  if i_first ≠ ∞ ET j_first ≠ ∞ then
    res.distance ← (P[i_first][j_first])/(res.correspondanceR.size());
  end if
  resultat.push(res)
end for
return resultat;
End

```

▷ Test si le dernier éléments est sauté ou non

▷ Calcul de la distance

3.4.3 Exemple

- Séquence référence : (1,2,3,49,64,34,9,22)
- séquence cible : (73,1,2,3,49,12,9,123,33)

Déroulons rapidement l'algorithme :

1ère étape : Calculer le tableau des distances

2nd étape : Calculer les skipCost et autre valeurs paramétrable

— skipCost = 4.6

— smallSkipCost = 1.8

— élasticité = 2

3ème étape : Déterminer les éléments à sauter

Row	73	1	2	3	49	12	9	123	33
1	-1	1	1	1	-1	1	1	-1	-1
2	-1	1	1	1	-1	1	1	-1	-1
3	-1	1	1	1	-1	1	1	-1	-1
49	-1	-1	-1	-1	1	-1	-1	-1	-1
64	1	-1	-1	-1	-1	-1	-1	-1	-1
34	-1	-1	-1	-1	-1	-1	-1	-1	1
9	-1	1	1	1	-1	1	1	-1	-1
22	-1	-1	-1	-1	-1	1	1	-1	1

FIGURE 3.8 – ESC : Determination des sauts

3ème étape : Calculer les matrices de coût et de chemin

	73	1	2	3	49	12	9	123	33
1	14.4	0	1	2	14.4	11	8	14.4	14.4
2	34	6.2	0	∞	∞	∞	∞	∞	∞
3	53.6	13.4	6.2	0	∞	∞	∞	∞	∞
49	∞	33	25.8	19.6	0	∞	∞	∞	∞
64	∞	∞	45.4	39.2	19.6	14.4	∞	∞	∞
34	∞	∞	∞	58.8	39.2	34	28.8	∞	∞
9	∞	∞	∞	∞	58.8	42.2	34	43.2	∞
22	∞	∞	∞	∞	∞	57.4	52.2	48.4	54.2

	73	1	2	3	49	12	9	123	33
1	∞	∞	0	∞	∞	∞	∞	∞	∞
2	∞	0	0	∞	∞	∞	∞	∞	∞
3	∞	1	1	1	∞	∞	∞	∞	∞
49	∞	2	2	2	2	∞	∞	∞	∞
64	∞	∞	2	2	3	3	∞	∞	∞
34	∞	∞	∞	3	3	3	3	∞	∞
9	∞	∞	∞	∞	3	3	3	3	∞
22	∞	∞	∞	∞	∞	6	6	6	6

Col	73	1	2	3	49	12	9	123	33
1	∞	∞	1	∞	∞	∞	∞	∞	∞
2	∞	1	1	∞	∞	∞	∞	∞	∞
3	∞	1	2	2	∞	∞	∞	∞	∞
49	∞	1	2	3	3	∞	∞	∞	∞
64	∞	∞	2	3	4	4	∞	∞	∞
34	∞	∞	∞	3	4	4	4	∞	∞
9	∞	∞	∞	∞	4	4	4	4	∞
22	∞	∞	∞	∞	∞	5	6	6	4

FIGURE 3.9 – FSM : Calcule des coûts

4ème étape : Effectuer un backtracking sur les matrices (bleu).

Le premier élément (orange) doit être sauté, on prend donc son parent comme premier élément.

	73	1	2	3	49	12	9	123	33
1	14.4	0	1	2	14.4	11	8	14.4	14.4
2	34	6.2	0	∞	∞	∞	∞	∞	∞
3	53.6	13.4	6.2	0	∞	∞	∞	∞	∞
49	∞	33	25.8	19.6	0	∞	∞	∞	∞
64	∞	∞	45.4	39.2	19.6	14.4	∞	∞	∞
34	∞	∞	∞	58.8	39.2	34	28.8	∞	∞
9	∞	∞	∞	∞	58.8	42.2	34	43.2	∞
22	∞	∞	∞	∞	∞	57.4	52.2	48.4	54.2

FIGURE 3.10 – ESC : Backtrack

5me étape Calculer la distance : $34/5 = 6.8$

ÉTUDE DE L'EXISTANT

4.1 Présentation

Au départ du projet, on a mis à ma disposition l'ensemble des sources qu'avait remis Abourahman ADEN HASSAN pour son PFE Matching Elastique et robuste séquences.

Ce dossier comprenait :

- un dossier projet comprenant :
 - une version du projet ligne de commande sous visual studio 2010
 - une version en ligne de commande sous codeblocks
 - une version librairie statique sous visual studio 2010
 - une version librairie static sous codeblock
 - une version Graphique sous QtCreator
- un dossier rapport comprenant :
 - le rapport de PFE
 - le cahier de spécification
 - un cahier de développeur
- un dossier sample comprenant divers fichiers de test
- les executables des projets
- les sources
- les classes de tests unitaires

Son projet était l'implémentation d'une *ToolBox* multiplateforme permettant de mettre en correspondance deux séquences numériques et alphanumériques.

De base, la *ToolBox* avait déjà implémenté 4 méthodes de *matching* :

- l'algorithme du Levenhstein
- l'algorithme de LCS
- l'algorithme du DTW
- l'algorithme du MVM

4.2 Fonctionnalités

Voici le diagrammes des cas d'utilisations original :

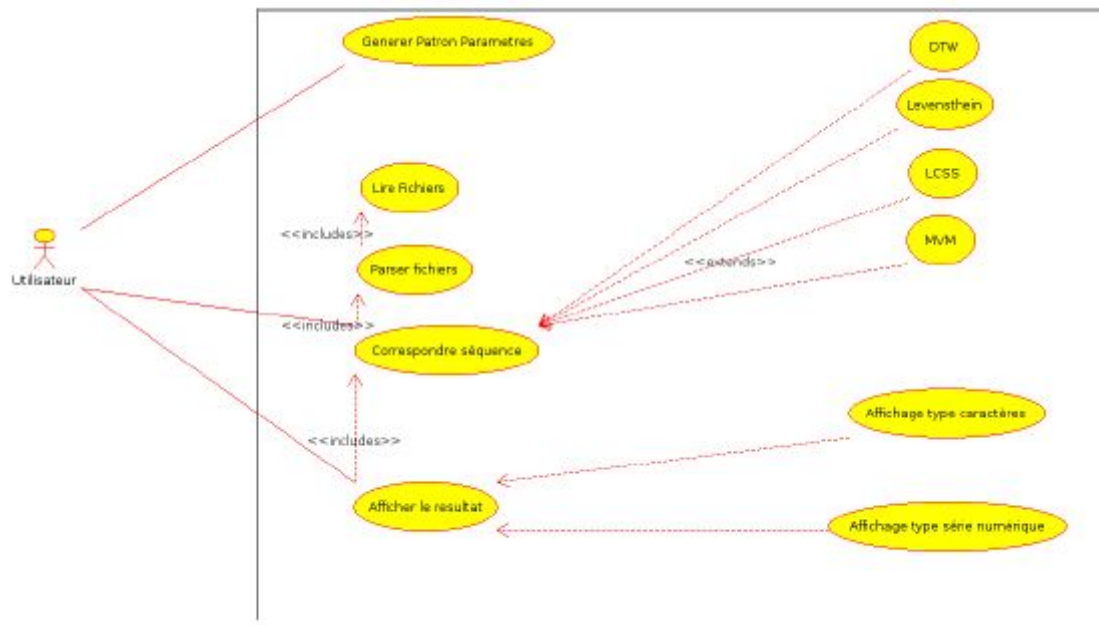


FIGURE 4.1 – Cas d'utilisations original

Comme on peut le voir, la *ToolBox* pouvait :

- lire des fichiers CSV
- parser ces fichiers en séquences
- faire une correspondance entre des séquences
- afficher le résultat en ligne de commande
- créer un patron de paramètre

De plus chaque algorithme était paramétrable. Il était également possible de paramétrer le processus de mise en correspondance en donnant des poids à certains éléments des séquences.

4.3 L'architecture de la *ToolBox*

4.3.1 Architecture générale

La *ToolBox* est découpé en quatre parties :

- la partie model : elle comprend l'ensemble des classes permettant de définir une séquence et les éléments la composant
- la partie calc : elle comprend l'ensemble des classes implémentant les méthodes de *matching* ainsi que la structure du résultat
- la partie inout : elle comprend toutes les classes permettant de lire et parser les fichiers de séquences, les fichiers de paramètres et d'afficher les résultats
- la partie commandLine : elle comprend la classe permettant de lire les instruction passé en ligne de commande et de faire le lien entre toute les parties. c'est cette classe qui appelle toutes les actions nécessaires à la mise en correspondance des séquences.

4.3.2 Diagrammes de classes

Afin de mieux comprendre l'architecture de base de la *ToolBox*, voici les diagrammes de classe fournis par A. ADEN HASSAN

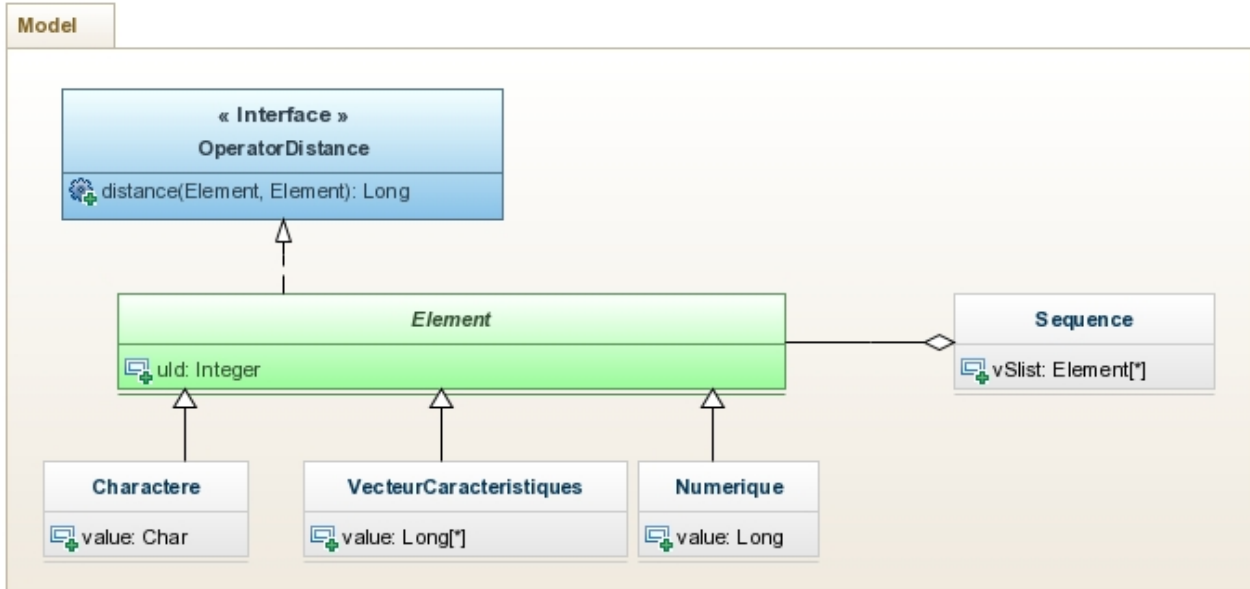


FIGURE 4.2 – Ancien model

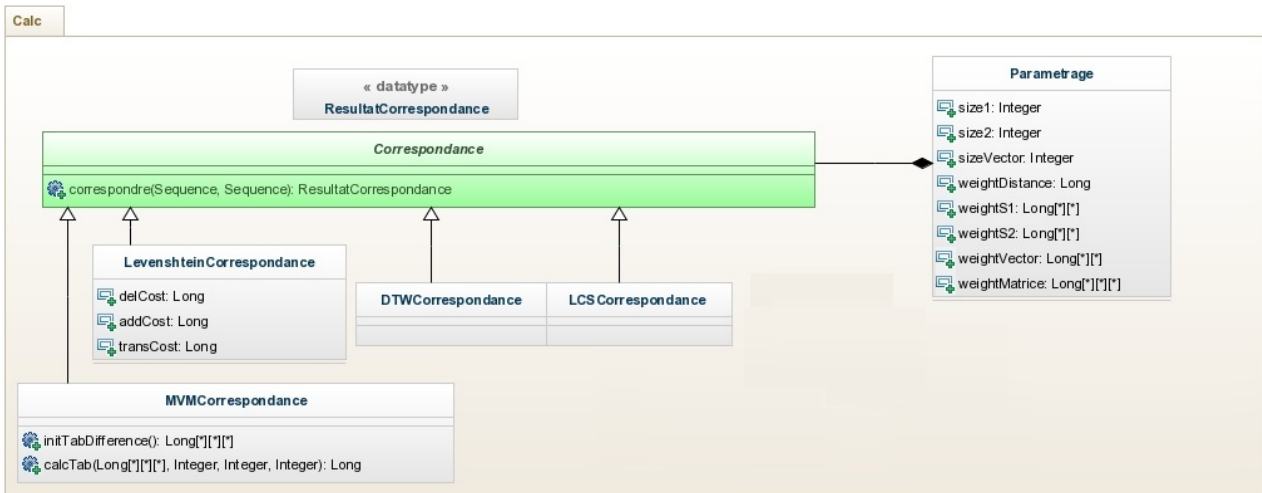


FIGURE 4.3 – Ancien calc

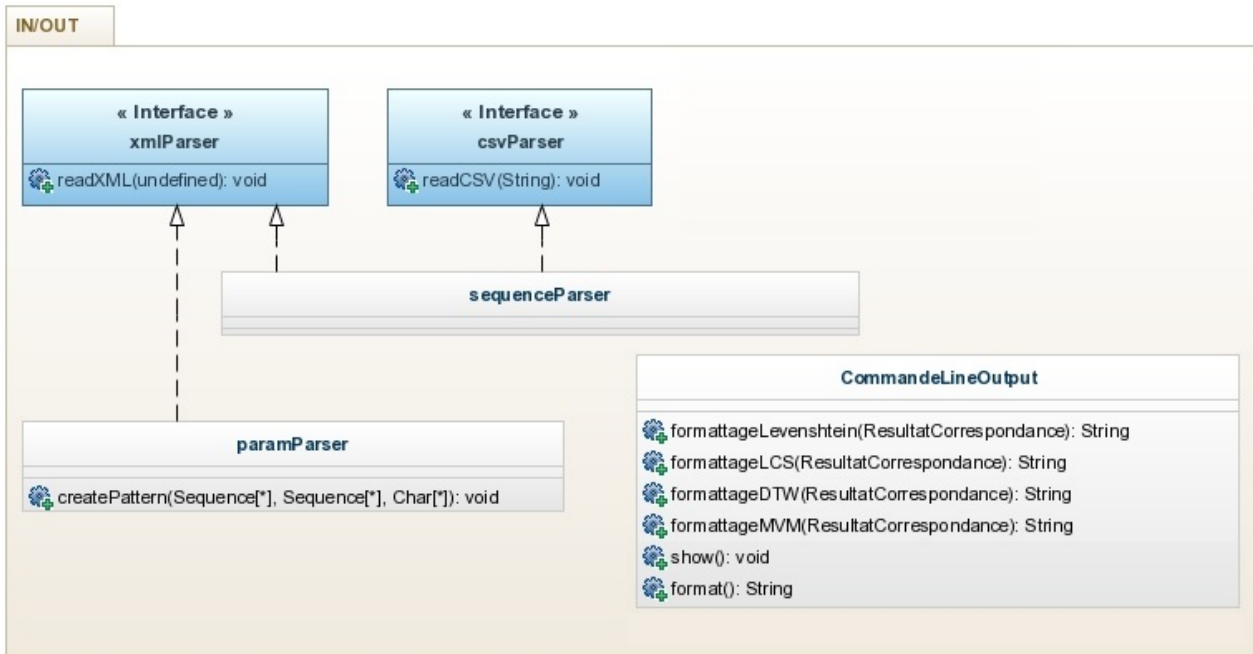


FIGURE 4.4 – Ancien inout

4.3.3 Exécution

Voici un schémas reprenant l'ordre d'appel de de ces classes lors d'une exécution. L'excecution se fait

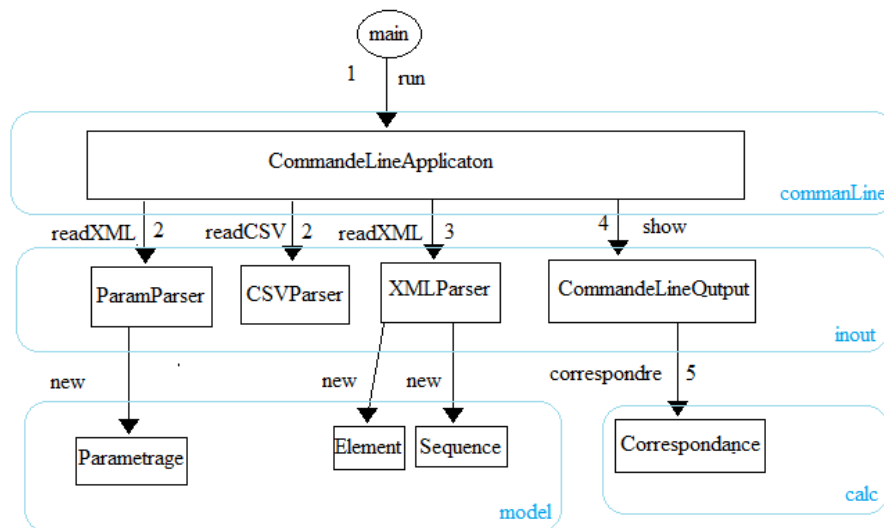


FIGURE 4.5 – Schémas d'excecution

en suivant les numéros (si deux axes ont le même numéros, les deux s'exécute dans dans l'ordre qu'ils veulent)

4.4 Remarques

4.4.1 Remarque général

Bien que la *ToolBox* soit bien codée, respectant toujours les mêmes normes de nommage et faisant une bonne gestion de la mémoire, trois grand points ont été relevés :

- Le premier est sur l'existence d'une seule classe implémentant deux interfaces de lecture dans la package *inout*.
- Le deuxième point concerne le manque d'utilisation de la puissance de l'héritage du code actuel. Le même code est réécrit plusieurs fois plutôt qu'une seule.
- Enfin A. ADEN HASSAN a préféré faire une gestion dynamique de la mémoire en utilisant les fonction *alloc()* et *free()* plutôt que d'utiliser les déclarations propre à C++.

De plus, certains codes étaient parfois maladroits, notamment pour tester les types de certaines instances lors de l'utilisation des classes abstraites. Enfin, certaines fonctions étaient simplifiables.

4.4.2 Remarque spécifique à la reprise

Affichage du résultat

Chaque méthode de *matching* avait sa propre fonction de formatage du résultat. Cependant, parfois c'était le même format. De plus le format utilisé pour l'affichage était une format visuellement compréhensible mais plus dur à réutilisé dans une autre application.

```
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 4.6 – Ancien affichage de résultat

Lecture des fichiers

Dans son implémentation de la *ToolBox*, les fichiers contenant les séquences devaient avoir un format spécial, pour pouvoir être lus.

	A	B
1	SEQUENCE	
2	TYPE	
3	Numeric	
4		
5	BEGIN	
6		1
7		2
8		8
9		6
10		8
11	NEXT	
12		1
13		2
14		8
15		8
16		8
17	NEXT	
18		1
19		2
20		8
21		8
22	NEXT	
23		1
24		2
25		8
26		16
27		16
28	END	
29		

FIGURE 4.7 – Ancien format de séquences

Ce format était facile à lire pour de petite séquences mais plus difficile pour les longues séquences.

Algorithmiquement

Enfin certains algorithmes de *matching* codés ne correspondaient pas toujours à celui donné par leur créateurs, notamment le MVM.

DEVELOPPEMENT

5.1 Modification et ajout

5.1.1 Ajout

Tout au long du projet, plusieurs fonctionnalités ont été intégrées à la *ToolBox*. Certaines étaient moins importantes que d'autres dans l'optique d'une mise en correspondance d'images de mots. Voici les fonctionnalités ajoutées dans leurs ordres d'ajout :

- Intégration de la méthode de CDP
- Intégration d'une méthode de lecture de fichier d'entrée différents à celui déjà créé
- Intégration d'une autre méthode de lecture de fichiers contenant des séquences classifiées
- Intégration d'un calcul de la précision d'une méthode grâce aux classes
- Intégration de la méthode du FSM
- Intégration de la méthode de ESC

La *ToolBox* créée par A. ADEN HASSAN, a été conçue pour pouvoir facilement ajouter d'autre algorithmes de *matching*. Pour cela il ne faut que ajouter une classe pour cette méthode et gérer son accessibilité dans la méthode *run* de la classe *CommandLineApplication*

Ajout du CDP

L'ajout de la méthode CDP s'est fait par l'ajout d'une classe implémentant l'algorithme étudié. La classe ainsi créée qui s'introduit dans le package *calc* se nomme *CDPCorrespondance*.

Intégration des méthodes de lecture

Notre format de fichier d'entrée étant différent de celui prévu par A. ADEN HASSAN. En effet la plupart de tests fournis était automatiquement créés avec Matlab. J'ai dû recréer une classe de lecture de fichier. Cette classe permet l'utilisation des fonctions de parsing déjà créées. Cette nouvelle classe s'introduit dans le package *inout* et s'appelle *EXTParser*.

Pour pouvoir tester l'efficacité des méthodes, on m'a demandé de tester le programme sur des fichiers différents que ceux finaux. Ces fichiers tests, appelés fichiers UCR, ont un format spécial, il a fallu ainsi créer une autre classe de lecture spéciale pour ce type de fichier. Cette classe s'appelle *EXTParserClassMatching* et se met dans le package *inout*. Cette lecture permet de stocker en mémoire la classe correspondante à chaque séquence.

Calcul de la précision

Une fois que les séquences et leur classes sont connues, on peut calculer le taux d'erreur d'une méthode. Pour réaliser ce calcul, une classe a été créée. Cette classe permet d'appeler la méthode de lecture spécifique à cette fonctionnalité.

Le calcul de l'erreur se fait selon le processus suivant :

- On calcule la distance entre chaque séquence référence et chaque séquence cible.
- Pour chaque référence, on compare sa classe avec celle de la cible qui possède la distance avec cette référence la plus petite.
- Si les classe sont les même, on incrémente un compteur.
- A la fin, le taux d'erreur s'obtient grâce au calcul suivant : $1 - \frac{\text{compteur}}{\text{nombre de référence}}$

Plus le taux d'erreur est bas, plus la méthode peut être considérée comme efficace. Cette méthode se compare à un processus de test de classification.

La classe permettant de gérer cette fonctionnalité s'appelle *CommandeLineApplicationClassMatching* et se place dans le package *commandeLine*.

Ajout du FSM

L'ajout de la méthode ESC se fait par l'ajout d'une classe implémentant l'algorithme étudié. La classe ainsi créée s'introduit dans le package *calc*. Elle se nomme *FSMCorrespondance*.

Ajout du ESC

L'ajout de la méthode CDP s'est fait par l'ajout d'une classe implémentant l'algorithme étudié. La classe ainsi créée s'introduit dans le package *calc* et se nomme *ESCCorrespondance*.

5.1.2 Modification

En parallèle des ajouts, plusieurs modifications ont été apportées au code afin de l'améliorer.

- La première modification a été le formatage du résultat. Nous avons simplifié le format de résultat afin qu'il soit plus facilement utilisable par un autre programme. La lecture du résultat sera décrite dans un autre point.
- La seconde modification a été réalisée au niveau du calcul des correspondances. Auparavant, dès qu'une mise en correspondance était réalisée, le résultat s'affichait en console immédiatement. Nous avons modifié le programme pour que dorénavant la *ToolBox* écrive ces résultats dans un fichier texte. Afin de ne pas ralentir le programme, nous avons décidé de tout calculer puis à la fin d'écrire l'ensemble des résultats.
- La troisième modification a été faite dans le package *inout*. Nous avons modifié le sens de l'héritage entre les classes *SequencesParser*, *XMLParser*, *CSVParser* et *ParamParser*. A présent il n'y a plus qu'une classe mère et plusieurs classes filles implémentant la classe mère. Ainsi cette organisation permet d'avoir plusieurs classes de lecture de fichier différent. Il pourra donc être possible d'avoir d'autres formats de séquences en entrée.
- La quatrième modification concerne l'ajout de paramètres. Avec les nouveaux algorithmes, de nouveaux paramètres sont apparus. Il a fallu les ajouter à la classe *Parametrage* et assurer leur lecture à partir des fichiers de paramètres et leur accessibilité par les méthodes de *matchnig*.
- Les dernières modifications effectuées sont de la refactorisation de code et le remplacement des fonctions *alloc*. Nous avons donc pu simplifier le code et réduire le nombre de lignes en supprimant de nombreux doublons.

5.1.3 Diagramme final

Voici les diagrammes de classes finaux de l'application :

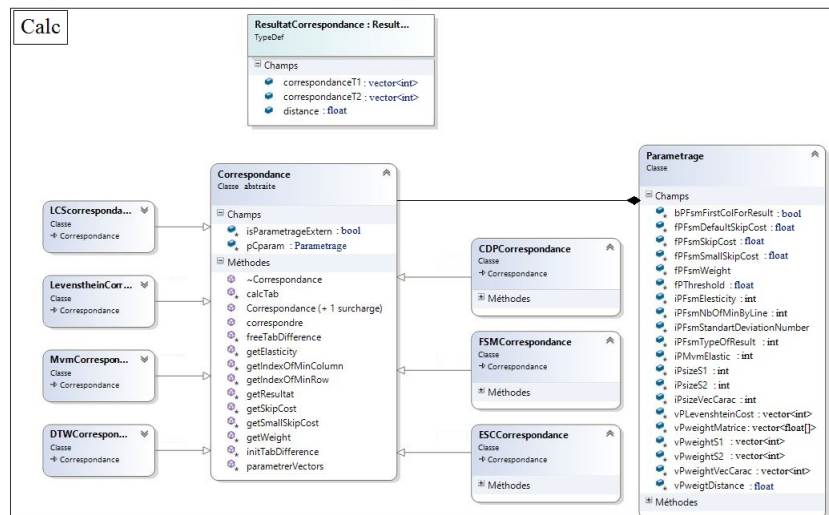
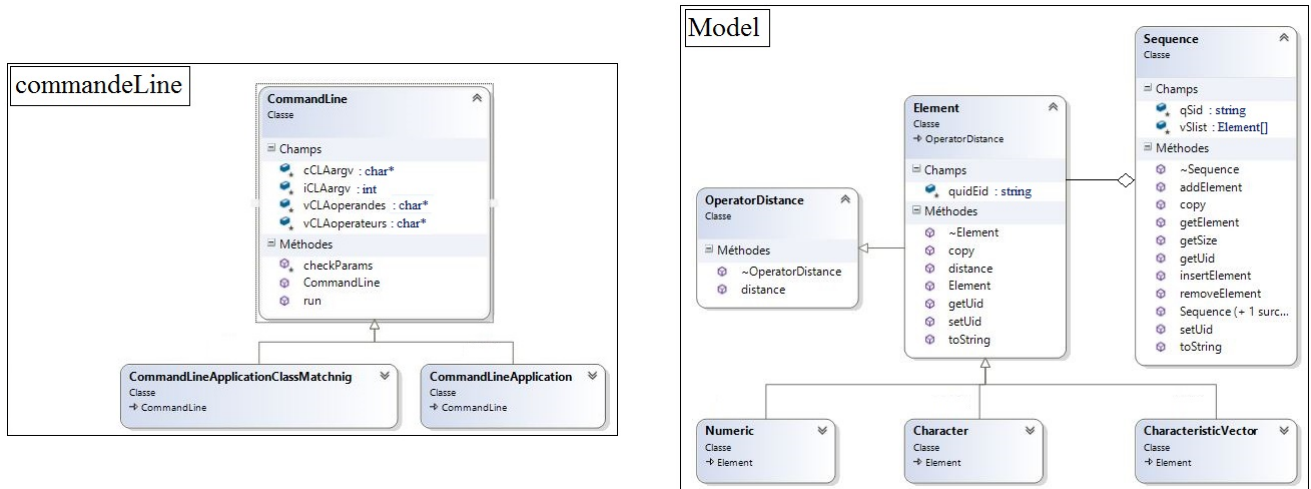


FIGURE 5.1 – Package calc

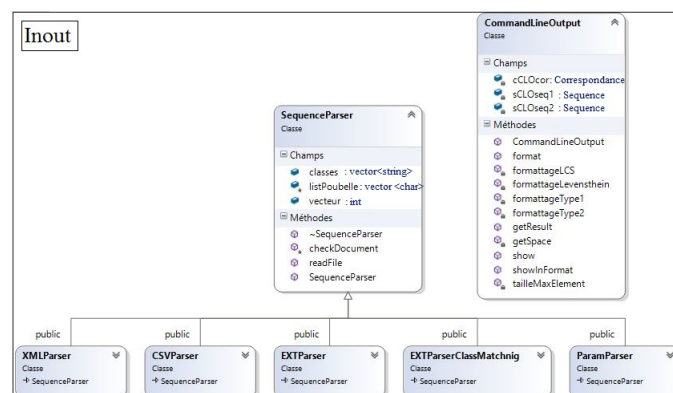


FIGURE 5.2 – Package inout

5.2 Exécution

Le schémas d'exécution final de l'application est similaire à celui d'origine. En effet, les modification apportées n'ont pas modifiées l'ordre d'appel des classes.

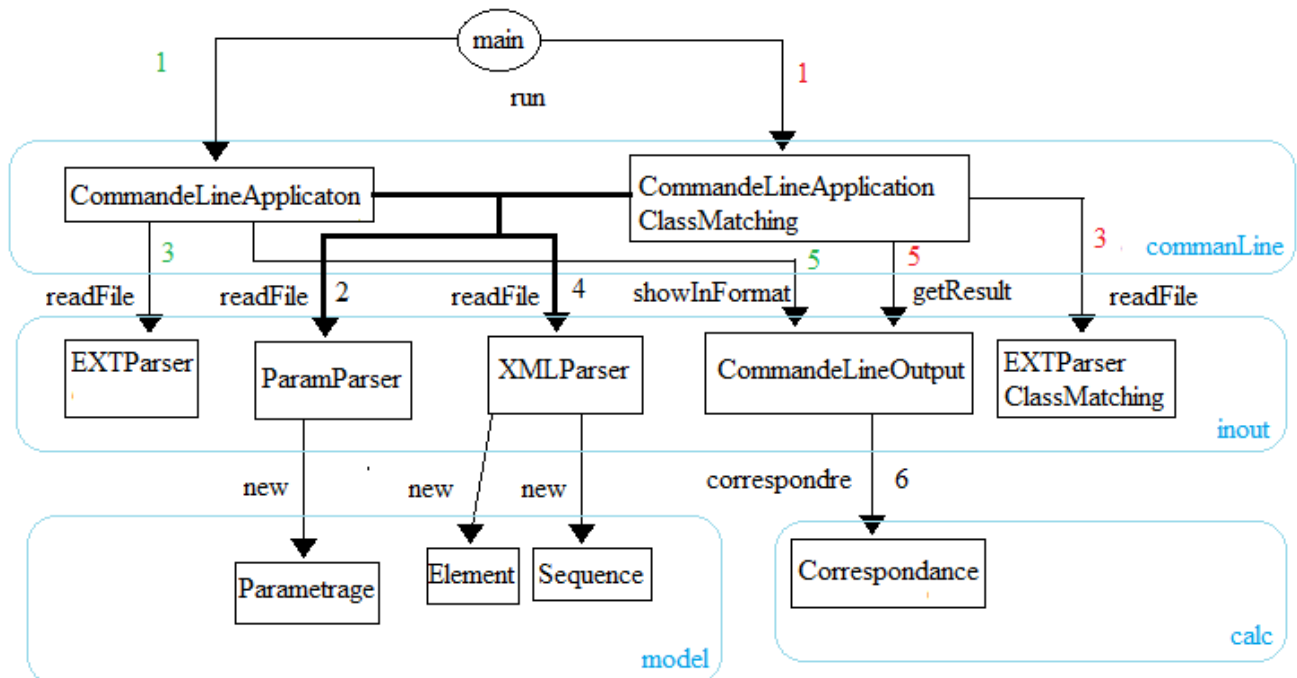


FIGURE 5.3 – Schémas d'excecuton

5.3 Tests

Durant ce projet, les seuls tests effectués étaient de type "fonctionnel". Nous avons procédé à trois types de test :

- des tests sur des séquences de type 1
- des tests sur des séquences de type 2
- des test d'efficacité sur les séquences UCR

Pour chaque type de test, une douzaine de paire références/cibles était testé permettant d'observer si le programme fonctionnait correctement. Le résultat des tests en repris dans un tableur Excel.

UTILISATION

6.1 Commande

Pour pouvoir utiliser la *ToolBox*, il suffit d'appeler l'exécutable avec les bons arguments. Il y a deux grands types de fonctionnalité à utiliser.

- Faire une mise en correspondance entre deux séquences
- Faire une évaluation du taux d'erreur avec des séquences de type UCR

Pour pouvoir utiliser la première fonctionnalité la commande à écrire est :
executable -arguments.

```
C:\Users\bastien\Desktop\PFE_Methode\Exec>MatchingToolBox.exe -sequences ..\samples\TestVectorFloat\Francois\target\402.csv ..\samples\TestVectorFloat\Francois\ref\402.csv -method -fsm -param ..\samples\paramFSMBase.xml -resultat ..\samples\TestVectorFloat\Francois\resultat
Fini avec 16.482 secondes
```

FIGURE 6.1 – commande d'exécution

Pour pouvoir utiliser la seconde fonctionnalité la commande à écrire est :
executable -pourcentage -arguments.

```
C:\Users\bastien\Desktop\PFE_Methode\Exec>MatchingToolBox.exe -pourcentage -sequences ..\samples\UCRTEST3\Beef\Beef_TRAI N ..\samples\UCRTEST3\Beef\Beef_TEST -method -cdp -param ..\samples\paramFSMBase.xml
Beef: 0.466667
Fini avec 100.583 secondes
```

FIGURE 6.2 – commande d'exécution

Voici la liste des arguments obligatoires et ceux possibles avec leurs spécifications :

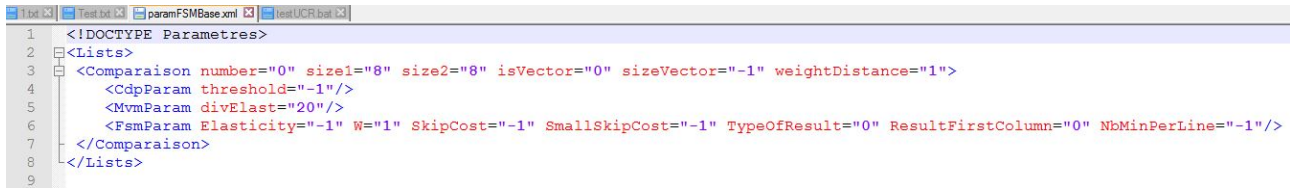
- *-sequence* pathFileCible pathFileReference (obligatoire) : indique les chemins d'accès des fichiers contenant les séquences
- *-method* -lvn [-cost addCost delCost transCost]|-lcs|-dtw|-mvm|-cdp|-fsm|-esc : indique la méthode à utiliser pour le matching, si non donné, l'algorithme utilisé est Levenshtein. Pour le Levenshtein, les coûts de modification des séquences peuvent être fournis directement.
- *-param* pathFileParametre : indique le chemin d'accès du fichier de paramètre si besoins
- *-resultat* pathDirectoryResultat : indique le chemin du répertoire dans lequel vous voulez stoker le résultat (ne fonctionne que pour le la première fonctionnalité), par défaut le repertoire est celui qui contient le fichier cible.

6.2 Paramétrage et Valeur par défaut

Chaque méthode et mise en correspondance peut varier en fonction de plusieurs facteurs. Le paramétrage est un point très important. En effet selon les paramètre, un algorithme va fonctionner de différentes manières. Le *ToolBox* possède un paramétrage par défaut si aucun fichier de paramètre est donné.

Voici un exemple de fichier de paramètre :

Le premier paramètre à faire régler est *isVector* ; ce paramètre décidera si votre fichier contient les séquences de vecteur de flottant ou des séquences de flottant simple. Il a deux valeurs possibles :



```

1 <!DOCTYPE Parametres>
2 <Lists>
3   <Comparison number="0" size1="8" size2="8" isVector="0" sizeVector="-1" weightDistance="1">
4     <CdpParam threshold="-1"/>
5     <MvmParam divElast="20"/>
6     <FsmParam Elasticity="-1" W="1" SkipCost="-1" SmallSkipCost="-1" TypeOfResult="0" ResultFirstColumn="0" NbMinPerLine="-1"/>
7   </Comparison>
8 </Lists>
9

```

FIGURE 6.3 – Fichier de paramètre

- 0 : ce sera des séquences de flottant simple
- 1 : ce sera des vecteurs, c'est la valeur par défaut

Attaché à ce paramètre on retrouve le paramètre *sizeVector* qui doit être modifié selon les cas. Il doit être supérieur à 0 si les séquences contiennent des vecteurs, -1 sinon.

Chaque algorithme possède ses propres valeurs paramétrables. Voici les paramètres pour CDP, FSM et ECS.

6.2.1 CDP

Le CDP ne possède qu'un seul paramètre : *threshold*. Il permet de définir le seuil de recherche pour la distance minimum. Par défaut, la valeur est de -1 ; cela signifie que la distance recherchée est la plus petite

6.2.2 FSM et ESC

Le FSM et le ESC possède les mêmes paramètres, il sont au nombre de 9 :

- *Elasticity* : donne l'élasticité, si < 0 alors l'élasticité sera calculé, -1 par défaut
- *W* : donne le poids d'un grand saut, 1 par défaut
- *SkipCost* : donne le skipCost, si < 0 alors l'élasticité sera calculé, -1 par défaut
- *SmallSkipCost* : donne le smallSkipCost, si < 0 alors l'élasticité sera calculé, -1 par défaut
- *TypeOfResult* : donne le type de calcul pour le résultat, 0 par défaut
- *ResultFirstColumn* : permet de savoir à partir de quelles colonnes la recherche du résultat commence :
 - true : recherche du début, valeur par défaut
 - false : recherche à partir de la fin de la référence
- *NbMinPerLine* : donne le nombre de minimum pris lors du calcul des skipCost (X dans la fonction de calcul), 2 par défaut
- *StandardDeviation* : donne le nombre d'écart type voulu lors du calcul de skipCost (Y dans la fonction de calcul), 2 par défaut
- *DefaultSkipCost* : valeur que prend le skipCost si son calcul retourne 0.

6.3 Interprétation

Une fois la correspondance faite, il suffit maintenant de lire le résultat. Un fichier de résultats se présente comme celui suivant :

Dans un fichier de résultats, chaque ligne correspond à une correspondance. Chaque correspondance est séparée en deux parties. La première partie correspond à la distance entre les deux séquences. La seconde partie donne les correspondances entre les indices des deux séquences. On lit les indices comme ceci :

indiceX->indiceY, cela veut dire que l'élément à la position indiceX de la référence correspond à l'élément d'indiceY de la cible.

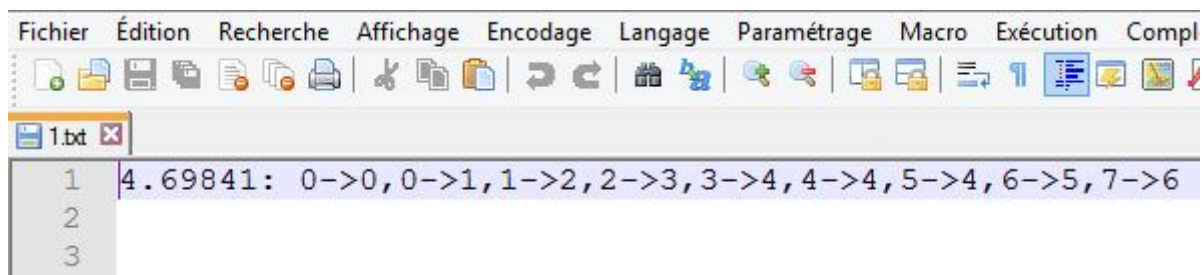


FIGURE 6.4 – Fichier de résultat

RESULTATS

7.1 Exemple de résultat de type 1

Séquence référence : 1,2,8,8
Séquence cible : 1,2,95,95,95,8,8

Séquence référence : 1,2,8,6,8
Séquence cible : 1,2,9,3,3,5,9

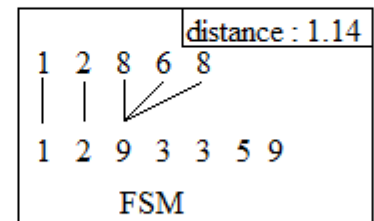
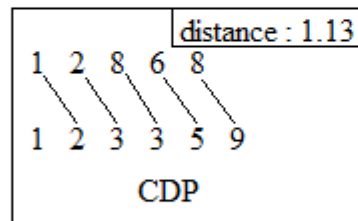
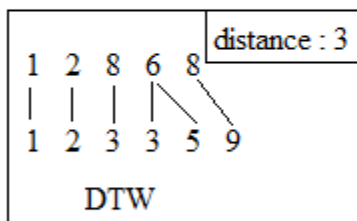
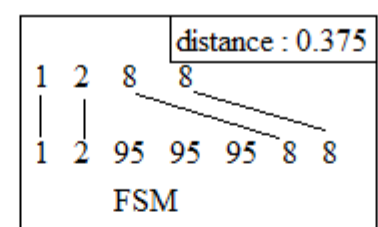
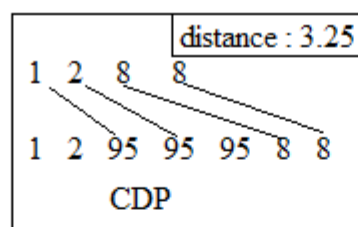
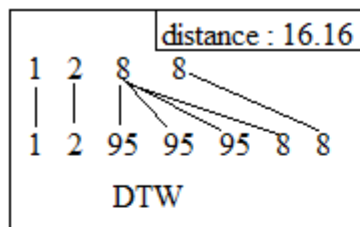


FIGURE 7.1 – resultat sequences simple

Le résultat des autres tests se trouve dans un tableur Excel.

7.2 Exemple de résultat de type 2

L'image de référence est une image avec le mot : François. Malheureusement, l'image de base ne m'a pas été fournie. Seul le fichiers contenant a séquences représentative m'a été donné.



FIGURE 7.2 – resultat sequences simple

On peut donc remarquer grâce au distance que l'image de référence se retrouve plus fidèlement dans la première image cible que dans la deuxième.

En regardant les 12 tests, il est apparu que l'image cible contenait une image proche le référence lorsque sa distance était inférieur à 0.49. Cependant, un test sur un plus grand échantillon serait nécessaire pour obtenir un seuil de correspondance plus adapté.

PROBLÈMES RENCONTRÉS

8.1 Problème de la *ToolBox*

Le premier problème rencontrés a été le temps de calcul, trop long pour certains fichiers de tests, notamment dans des fichiers de tests UCR.. Pour résoudre ce problème, T. MONDAL et moi même avons mis en place une parallélisation de la lecture des fichiers d'entrée ainsi que la parallélisation du calcul. Ainsi, certains fichiers qui ne pouvaient pas être exécutés en une journée de travail, ont pu être exécutés en moins de deux heures.

Afin de favoriser un gain de temps, tous les calculs sont réalisés au préalable de l'affichage (cette phase prenant beaucoup de temps).

Cependant, réaliser tous les calculs oblige l'ordinateur à stocker en mémoire flash tous les résultats. Ainsi il est arrivé pour les plus gros fichiers que l'ordinateur manque de RAM et quitte l'application. Malheureusement, je n'ai pas trouvé immédiatement l'origine du problème

La plupart des autres problèmes qui ont pu me bloquer étaient dus à des fautes d'étourderies. Un autre problème venait du fait que souvent il me fallait modifier les algorithmes implémentés pour cause d'erreur dans les papiers et de modifications de l'algorithme en lui même. Je devais donc sans cesse retester pour avoir des tests à jours

Enfin les deux derniers problèmes concernent l'implémentation des algorithmes. Lors des tests, j'ai remarqué que pour le FSM et le ESC, les relations un à plusieurs ne sont trouvés qu'avec le premier élément. Malheureusement, je n'ai pas réussi à trouver une solution à ce problème.

Le dernier problème viens du manque de temps pour coder le ECS. Cet algorithme m'ayant été donné la dernière semaine, son pseudo-code n'a pas été vérifié. Je n'ai pas eu le temps de le tester, son résultat n'est donc pas garantie.

8.2 Communication avec les acteurs du projet

Durant ce projet plusieurs problèmes de communication se sont posés. Le premier point viens du manque de compréhension entre T. MONDAL et moi du fait de la différence de langage parlé. Ainsi j'ai passé beaucoup du temps à créer une solution pour lire et utiliser les fichiers UCR alors que je n'ai su que bien après que ce n'était pas ce type de fichier que nous allions utiliser au final.

J'ai souvent eu l'impression qu'il me demandait de faire des fonctionnalités qui allais plus dans le sens de ses recherche que celui dans mon PFE.

Cependant, il a bien su m'expliquer comment marchaient les différents algorithmes.

Un autre point qui m'a fait défaut durant ce PFE à été mon manque de communication avec M. RAGOT. En effet je communiquais plus avec T. Mondal. M. RAGOT aurai sans doute plus me recarder sur mon PFE.

8.3 Problème de gestion de projet

Durant le déroulement de ce projet, j'ai eu trois grands problèmes pour la gestion de projet.

Le problème le plus important dans ma gestion de projet a été mon implication trop importante sur le codage, ce qui a entraîné un délaissement de la partie purement gestion du projet. Rétrospectivement, si j'avais passé plus de temps à analyser le projet, certaines modifications auraient été plus rapides à réaliser.

Un autre problème du au de nombreuses modifications des algorithmes et définition des paramètres, notamment pour le FSM au cours du projet. En effet, à chaque modification, les tests devais être refais. Tous les tests effectués n'ont cependant pas été sauvegardés.

De plus, un problème s'est posé lors des tests du FSM et MVM avec les fichiers UCR. En effet, la machine qui me servait pour les tests m'a été retirée. Ne possédant par un ordinateur assez puissant, tous les tests UCR n'ont pas pus été effectués.

Cette machines me servant également à codé avec OS linux, la dernière version du logiciel ne sont pas fonctionnel sur cette plateforme. Cependant, les méthodes implémentés ont été codées de façon à marcher sur les deux plateformes.

Le dernier problème que j'ai rencontré à été le manque de temps ; jou tout du moins un défaut d'organisation de mon temps de travail ; je me suis concentré pour rendre une implémentation de l'algorithme du ESC fonctionnelle au détriment de la rédaction d'un cahier permettant d'expliquer le projet.

CONCLUSION

Dans ce rapport, je vous ai présenté mon travail réalisé lors de ce Projet de Fin d'étude. Grâce à celui-ci projet j'ai pu voir comment on pouvait reprendre un code existant afin de l'améliorer et d'y ajouter des fonctionnalités. Ainsi j'ai pu appréhender la difficulté de lire le code d'une autre personne, d'analyser comment son programme fonctionne et d'étudier les choix qu'il a fait. Cette expérience me sera utile plus tard pour mon stage et mes futurs projets.

Ce travail m'a permis de voir de nouvelle façon de penser en étudiant des algorithmes de *matching* et de voir leurs possibilités dans un processus de correspondance. De plus tous au long du projet, j'ai pu pratiquer le passage d'un algorithme général à un code fonctionnel. L'implémentation des algorithmes et de la *ToolBox* en C++, m'ont aidé à m'améliorer dans ce langage objet puissant que je ne maîtrisais mal.

En travaillant avec M. MONDAL, j'ai pu m'apercevoir de la difficulté de communiquer facilement avec un non francophone à entraîné des pertes de temps et d'efficacité. En effet même si mon anglais est suffisamment bon pour tenir une conversation usuelle, certains mots techniques ou expressions me manquaient. Cette compétence de communication s'acquiert avec la pratique. Cette expérience aura donc été importante. Je sais dorénavant qu'il vaut mieux répéter plusieurs fois les choses de façon différentes pour être assuré de se faire comprendre.

En conclusion, ce projet m'a permis de consolider l'ensemble de mes compétences, tant sur les points techniques, méthodologiques et relationnels. J'ai découvert que même si mes compétences techniques me permettaient de réaliser un projet, j'ai encore du mal à avoir seul une bonne gestion de ce dernier. J'ai cependant tiré beaucoup de profit de cette année et je connais maintenant les points que je dois absolument améliorer.

Bibliographie

- [1] *Flexible Sequence Matching Technique :An Eective Learning-free Approach For Word Spotting*,
<http://sciencedirect.com>.
- [2] *Exemplary Sequences Cardinality : An effective Application of WordSpotting*,
<http://sciencedirect.com>.
- [3] *Classification of Real World Data*,
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.3747&rep=rep1&type=pdf>
- [4] *Rapport de projet, Matchnig Elastique et robuste de sequence*

Recherche par mots-clés "image" dans une masse d'image de documents : méthode

Résumé : Ce PFE a permis d'implémenter trois algorithmes de mise en correspondance de séquences numérique. Le CDP, le FSM et l'ESC ont donc été intégrés dans une boîte à outils permettant de calculer une distance entre deux séquences. Ces séquences seront, en particulier, caractéristiques extraites d'image de mots provenant de livre anciens. Ce rapport nous donne le contexte du projet, son déroulement, ses spécificités et son résultat final. Ainsi il donne l'étude de la boîte à outils créée dans un précédent projet et l'étude des trois différents algorithmes implémentés. Il explique également les modifications apportées à la boîte à outils, les tests effectués sur ces algorithmes, le déroulement du projet et les divers problèmes rencontrés tout au long de l'année. Il présente aussi comment utiliser la boîte à outils et donne des exemples d'utilisation.

Mots clefs : Mise en correspondance, FSM, CDP, ESC, image de mot, UCR, distance

Abstract: This projet is about an implementation of three algorithms for matching numerical sequences. So, the CDP, the FSM and the ESC have been included in a toolbox used for calculate a distance between two sequences. Those sequences will be, in this case, some features extracted from pictures of word from old books. This text give us the project's context, its developpement, its spécification and its result. Thus, it give us a study of the toolbox created afore in another projet and the study of the three algorithms. It explain the modification done to the toolbox, the tests done to the algorithms, the developpement of the projet and all the problems whose been coming across all the year. Finally, it present how to use the toolbox ad show some utilisation exemples.

Keywords: Matching, FSM, CDP, ESC, picture of word, UCR, distance

Laboratoire

Laboratoire Informatique de Tours(EA630)
Équipe Reconnaissance des Formes, Analyse
d'Images (RFAI),
64, avenue Jean Portalis
37200, Tours



Étudiant

Bastien MEUNIER
bastien.meunier@gmail.fr
DI5 2014 - 2015

Encadrant

Nicolas RAGOT
nicolas.ragot@univ-tours.fr