RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Laboratoire de Mathématiques et Sciences Appliquées

Université de Ghardaia

Faculté des Sciences et de Technologie
Département des Mathématiques et Informatique

MEMOIRE

Présenté pour l'obtention du diplôme de ***MASTER***

En : Informatique
Spécialité : Systèmes Intelligents pour l'Extraction de Connaissances (SIEC)

Par : *HEROUINI Slimane & LAGHRAB Ahmed*

Sujet

Vers un noyau d'ensemble de séquences

Soutenu publiquement, le 04 / 07 / 2019, devant le jury composé de :

OULAD NAOUI S.	***MC.***	***Président***
MAHDJOUB Y.	***MA.***	***Examinateur***
BOUHANI A.	***MA.***	***Examinateur***
BELAOUAR S.	***MC.***	***Directeur***

Année Universitaire 2018/2019

Dédicace

Je dédie ce modeste travail à :
Ma très chère Mère,que dieu la protège.
Mon très cher Père.que dieu le protège.
Pour leur patience, leur amour, leur soutien et leurs encouragements.
Mes très chers frères et soeurs.
A toute ma famille.
A tout mes amies et mes camarades.
A tous les enseignants et les enseignantes qui ont contribué a ma formation tout au long de ma vie A tout ceux qui m'ont aidé dans l'élaboration de ce travail.

Herouini Slimane

Dédicace

Je dédie ce modeste travail à :

Mes parents et mes grands-parents vos prières m'ont été d'un grand secoure pour mener a bien mes études , Aucune dédicace ne saurait être assez éloquente pour exprimer ce que vous méritez pour tous les sacrifices que vous n'avez cessés de me donner

Mes chères sœurs, mon frère , mes tantes et mes oncles.

En témoignage de l'attachement ,de l'amour et de l'affection que je porte pour vous, je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de réussite A tous les autres membres de ma famille sans exception.

A mes amis et mes collègues dans et en dehors l'université.

Laghrab Ahmed

Remerciements

En préambule à ce mémoire, la grande louange à **Dieu** qui nous aide et nous donne la bonne santé, la patience et le courage durant l'élaboration de ce modeste travail.

Nous tenons à remercier tous ceux qui nous ont aidés, d'une manière ou d'une autre, pendant ce travail d'étude et de recherche.

Nous tenons d'abord à remercier très chaleureusement Monsieur **SLIMANE BELLAOUAR** qui nous a permis de bénéficier de son encadrement. Les conseils qu'il nous a prodigué, la patience, la confiance qu'il nous a témoignés ont été déterminants dans la réalisation de notre travail de recherche.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre modeste travail Et de l'enrichir par leurs propositions.

Nos remerciements s'étendent également à tous nos enseignants durant les années des études.

Merci à Tous et à Toutes.

Résumé

Les méthodes à noyaux se proposent comme une alternative aux problèmes des données non linéairement séparables.

Grâce à ces méthodes, l'utilisation des algorithmes classiques d'apprentissage automtique devient possible.

Les noyaux de séquences ont attiré l'attention des chercheurs en se focalisant sur des problèmes spécifiques conduisant à une diversité de noyaux de séquences.
Dans la présent mémoire, notre objectif consiste à faire, d'abord, une investigation de la famille de noyaux de séquences.
Ensuite, l'effort c'est concentré sur deux aspects, l'unification des noyaux de séquences ainsi que la généralisation vers un ensemble de séquences.
L'aspect unification est réalisé en étudiant une plate-forme générale pour calculer les noyaux de séquences. Une telle plate-forme mis en jeu trois concepts : noyau, série formelle et automate pondéré.
L'aspect de généralisation se repose sur l'utilisation d'un automate pondéré préfixe.
Finalement, nous nous somme focalisés sur l'implimentation de la plate-forme générale généralisée, où nous décrivons la structure de données liste d'adjacence utilisée ainsi que les différents classes développées.

Mots clé : Apprentissage automatique, méthode à noyaux, noyau de séquence, unification, généralisation, automate pondéré, noyau d'ensemble de séquences.

Abstract

The kernel methods are proposed as an alternative to the problems of non-linearly separable data.
With the help of these methods, the use of classical machine learning algorithms becomes possible.
sequences kernels attracted the attention of researchers by focusing on specific problems leading to a variety of sequences kernels.
In this thesis, our goal is to, first investigate the family of kernels of sequence. Then, the effort is focused on two aspects, the unification of the kernels of sequences as well as the generalization towards a set of sequences.
The unification aspect is realized by studying a general platform to calculate the kernels of sequence. Such a platform brings into play three concepts : kernel, formal series and weighted automata.
The generalization aspect relies on the use of a prefixed weighted automata.
Finally, we focused on the implimentation of the generalized general platform, where we describe the adjacency list data structure used as well as the different classes developed.

Keywords : Machine learning, kernel method, string kernel, unification, generalization, weighted automaton, sequence-set kernel.

ملـخـص

تُقترح طرق النواة كبديل لمشاكل البيانات القابلة للفصل غير الخطي.

بفضل هذه الأساليب ، يصبح استخدام الخوارزميات الكلاسيكية للتعلم الآلي أمرًا ممكنًا.

جذبت نوى التسلسل انتباه الباحثين من خلال التركيز على مشاكل محددة تؤدي إلى مجموعة متنوعة من نوى التسلسل.

في هذه المذكرة، هدفنا هو أولاً التحقيق في عائلة نوى التسلسل.

بعد ذلك ، يركز الجهد على جانبين ، توحيد نوى التسلسل وكذلك التعميم تجاه مجموعة من التسلسلات.

يتحقق جانب التوحيد من خلال دراسة منصة عامة لحساب نوى التسلسل. تعمل مثل هذه المنصة على وضع ثلاثة مفاهيم: النواة والسلسلة الصورية و الآلي المرجح.

يعتمد الجانب التعميمي على استخدام آلية مرجحة مسبوقة.

أخيرًا ، ركزنا على تجسيد النظام الأساسي العام المعمم ، حيث نصف بنية بيانات قائمة الجوار المستخدمة وكذلك الفئات المختلفة التي تم تطويرها.

كلمات مفتاحية : التعلم الألي، طرق النواة، نوى السلاسل، التوحيد، التعميم، اليات المرجحة، نواة مجموعة من السلاسل.

Table des matières

Liste des tableaux **iii**

Table des figures **v**

Liste des Abréviations **vi**

Introduction Générale **1**

1 Méthodes à noyaux et noyaux de séquences **4**
1.1 Méthodes à noyaux 5
1.1.1 Analyse des données non linéaires 5
1.1.2 Noyaux et modularité 6
1.1.3 Validité des fonctions noyaux 7
1.1.4 Types de noyaux 8
1.2 Noyaux pour le texte 8
1.3 Noyaux de séquences 9
1.3.1 Mots et séquences 9
1.3.2 Noyau p-spectre 10
1.3.3 Noyau toutes sous-séquences 12
1.3.4 Noyau sous-séquences de longueur fixe 18
1.3.5 Noyau sous-séquences de mots 20
1.4 Conclusion 23

2 Unification et généralisation des noyaux de séquences **25**
2.1 Préliminaires 26
2.1.1 Semi annaux 26
2.1.2 Automate fini 27
2.1.3 Automate pondéré 27
2.1.4 Série formelle 28
2.2 Modèle à base d'automates pondérés 29

2.2.1 Idée principale 29
2.2.2 Gappy All Subsequence Weighted Automaton 30
2.2.3 Validité et Efficacité de GASWA 32
2.3 Calcul du noyau de séquences à base d'automates pondérés . . . 33
2.3.1 Intersection des automates pondérés 34
2.3.2 Évaluation du noyau 36
2.4 Relations avec d'autres noyaux de séquences 36
2.4.1 Noyau toutes sous-séquences 36
2.4.2 Noyau sous-séquences de mots 37
2.4.3 Autre noyaux de séquences 38
2.5 Généralisation pour un noyau d'ensembles de Séquences 38
2.6 Conclusion 40

3 Implémentation des noyaux de séquences 46
3.1 Environnement de programmation 47
3.2 Représentation des automates 47
3.2.1 Matrice d'adjacence 48
3.2.2 Liste d'adjacence 49
3.3 Choix de la structure de données 50
3.3.1 Complexité d'utilisation la structure liste linéaire chaînée 50
3.3.2 Complexité d'utilisation la structure tableau dynamique 51
3.4 Implémentation de l'automate pondéré préfixe 52
3.4.1 Classe Transition 52
3.4.2 Classe Automate 52
3.4.3 Classe état 53
3.4.4 Classe Noyau 53
3.5 Conclusion 54

Conclusion Générale 54

Bibliographie 56

Liste des tableaux

1.1 Noyau 2- spectres entre les mots "bar ", "bat ", "car" et "cat ". . 11
1.2 Noyau 3- suffixes entre deux mots s="statistics" et t="computation". 12
1.3 Noyau 3- specttres entre deux mots s="statistics" et t="computation". 13
1.4 Les sous-séquences des mots "bar", "baa", "car" et "cat". 13
1.5 Noyau toutes sous-séquences entre les mots "bar", "baa", "car" et "cat". 14
1.6 Les sous-séquences qui se terminent par le symbole "a" dans les mots s = "gatta"et t = "cata". 15
1.7 Les sous-séquences qui se terminent par le symbole "a" dans les mots s = "gatta"et t = "cata". 16
1.8 Table de programmation dynamique pour le calcul d'un noyau. . 16
1.9 Noyau toutes sous-séquences entre les mots "bar", "baa", "car" et "cat". 17
1.10 Table de la programmation dynamique pour le calcul du noyau toutes sous-séquences entre les modt s = "cata" et t="gataa" avec le pré-calcul. 19
1.11 Projection des mots *bar*, *bat*, *car* et *cat* dans un espace de redescription pour des sous-séquences de longueur $p = 2$. 21

2.1 Table représenté les définition des opérations. 29
2.2 Espace de redescription du mot $s = cata$ associé au noyau toutes sous-séquences avec trous. 32

3.1 Matrice d'adjacence représentent l'automate de la Figure 3.1. . . 48
3.2 Comparaison entre liste adjacence et matrice adjacence 50
3.3 Comparaison entre un liste linéaire chaînée et un tableau dynamique en terme de complexité temporelle. 51
3.4 API pour la classe Transition. 52
3.5 API pour la classe Automate. 53
3.6 API pour la classe état. 54

3.7 API pour la classe noyau. 54

Table des figures

1.1 Transformation non linéaire des données d'entrées via une projection ϕ. Extrait de [Alex and Vishwanathan, 2008] 6
1.2 Modularité dans les applications des méthodes à noyaux. Extrait de [Shawe-Taylor and Cristianini, 2004]. 7
1.3 Calcul et stockage des sommes intermédiaires dans un tableau,Extrait de [Bellaouar, 2018] 17
1.4 Remplissage de la ligne suivante dans la table dynamique,Extrait de [Bellaouar, 2018]. 17
1.5 Évaluation du noyau toutes sous-séquences,Extrait de [Bellaouar, 2018]. 18
1.6 algorithme Evaluation du noyau sous-séquences de longueur p. Extrait de [Bellaouar, 2018]. 20

2.1 Automate représentant le langage des mots sur { a,b } préfixés par aa. 27
2.2 Automate pondéré pour toutes les sous-séquences d'un mot $s = s_1 s_2 \dots s_n$. 31
2.3 Automate pondéré pour toutes les sous-séquences du mot $s = cata$. 32
2.4 Les deux automates A_s et A_s associés respectivement aux mots $s =$ "acb" et $t =$ "abc". 35
2.5 Les deux automates A'_s et A'_t associés respectivement aux mots $s =$ "xyz" et $t =$ "xzy". 35
2.6 Automate pondéré $A_{s,t} = A_s \cap A_t$ 41
2.7 Automate filtre pour éliminer les ϵ-chemins. de [Bellaouar et al., 2017] 42
2.8 Automate 2-$gram$ (A_{pgram}), x représente un élément de Σ. Extrait [Bellaouar, 2018] 42
2.9 Automate pondéré pour tous les sous-mots d'un mot $s = s_1 s_2 \dots s_n$. 42
2.10 Automate pondéré préfixe représentant l'ensemble de mots $D = \{xyy, xyz, yzv$
2.11 Le APP renforcé par les états du deuxième niveau. 44
2.12 Le APP final représentant toutes les sous-séquences des mots de l'ensemble $D = \{xyy, xyz, yzv, yzw\}$. 45

3.1 Automate pondéré correspondant à la séquence "123". 48
3.2 Liste d'adjacent de l'automate de la Figure 3.1 par la structure tableau de tableau dynamique (a) et la structure tableau de liste (b). 49

Liste des Abréviations

AA Apprentissage Automatique

ACP Analyse en Composantes Principales

ACC Analyse Canonique des Corrélations

VSM Vector Space Model

SSK String Subsequence Kernel

RST Range Sum Tree

LRT Layered Range Tree

LRST Layered Range Sum Tree

SVM Support Vector Machine

GASWA Gappy All Subsequence Weighted Automaton

APP Automate Pondéré Préfixe

DFS Depth First Search

Introduction générale

Ces dernières années, et à l'ombre de l'évolution scientifique et technologique dans le monde, la numérisation croissante et continue dans différents domaines a conduit à un déluge d'informations. Par conséquent l'exploitation et le traitement manuel de ce volume de données sont devenus couteux et complexes. Une alternative à ce problème consiste à faire appel à des techniques l'apprentissage automatique (AA), qui permettent d'extraire de l'information pertinente à partir des collections de données massives. Dans la littérature, les algorithmes d'AA ont été appliqués pour un large spectre d'applications comme : traitement du langage naturel, bio-informatique, diagnostic médical, reconnaissance de formes, moteurs de recherche, génie-logiciel, robotique, jeux, ...

Cependant, les méthodes classiques d'AA sont des méthodes linéaires et bien adaptées aux données vectorielles. Dans la pratique, de nombreuses applications disposent de données non linéairement séparables qui peuvent être représentées sous forme structurée (arbres,séquences, graphes,...).

Les méthodes à noyaux fournissent une approche efficace pour prendre en charge ce type de données structurées en entrée, Ceci grâce à son astuce qui consiste à projeter les données en entrée dans un espace de redescription de haute dimension. Cette astuce garde les dépendances et les régularités inhérentes aux données, tout en évitant le calcul explicite de cette projection. Les noyaux sont des mesures de similarité qui peuvent être utilisés avec tout algorithme d'apprentissage appliqué sur des données linéairement séparable.

Par ailleurs, parmi les plus importants types de données dans plusieurs domaines, nous trouvons les séquences. Dans la pratique, il existe divers types de noyaux de séquences. Chaque noyau est centré sur un problème individuel donnant naissance à plusieurs approches. Cela conduit à penser à un modèle pour l'unification de ces noyaux de séquences.

Dans ce mémoire, nous nous focalisons sur les méthodes à noyaux et les noyaux de séquences. Ensuite, nous abordons deux aspects. Le premier concerne l'unification, où nous décrivons une plateforme générale pour le calcul des noyau sé-

quences. Puis, nous discutons l'aspect généralisation, dont le but est d'étendre cette plateforme générale afin de prendre en compte un ensemble de mot au lien d'un seul mot.
Dans la littérature, il existe plusieurs méthodes à noyaux de séquence. À titre non exhautif, nous pouvons énumérer, le noyau p-spectre [Leslie et al., 2002], le noyau toutes sous- séquences, le noyau sous-séquences de longueur fixe [Shawe-Taylor an et le noyau sous-séquences de mots (SSK) [Lodhi et al., 2002].

Dans un but d'unification des noyaux de séquences, nous présentons une plate-forme générale proposée pour calculer les noyaux de séquences. Cette plate-forme est basée sur la relation entre « noyau », « séries formelles » et « automates pondérés » [Bellaouar et al., 2017]. Où la projection d'un mot s dans un espace de redescription de haute dimension peut être modélisée par une série formelle qui peut être réalisée par un automate pondéré A_s représentant toutes les sous-séquences du mot s. Alors que, pour l'évaluation du noyau entre deux mots s et t, nous devons construire l'automate résultant de l'intersection entre deux automates qui représentent respectivement toutes les sous-séquences du mot s et toutes les sous-séquences du mot t. Ensuite, nous devons calculer la somme des poids de tous les chemins réussis qui représentent toutes les sous-séquences communes entre les mots s et t.
Ensuite, nous étudions l'aspect généralisation de la plate-forme pour un ensemble de mots au lieu un seul mot, afin de construire un nouveau noyau appelé noyau d'ensembles de séquences. Cette généralisation se base sur la construction d'un automate pondéré préfixe [Bellaouar, 2018] .
Finalement, nous nous sommes focalisés sur l'implémentation du modèle généralisé, où nous avons décrit la structure de donnée utilisée ainsi que les classes développées.
Notre mémoire est organisé comme suit :

- Chapitre 1 : introduit la notion des méthodes à noyaux et décrit les noyaux de séquences largement utilisés.
- Chapitre 2 : présente une plateforme générale pour le calcul des noyau séquences qui unifie tout les noyaux de séquences et la généralisation de cette plateforme pour calculer un noyaux d'ensemble de mots au lieu un seul mot.
- Chapitre 3 : décrit l'implémentation de la plateforme générale de noyaux de séquences avec la généralisation pour calculer un noyaux d'ensemble des

mots.

- Conclusion générale : contient une synthèse de notre travail ainsi que les perspectives envisigées.

Chapitre 1

Méthodes à noyaux et noyaux de séquences

Le présent chapitre introduit les concepts nécessaires de base utiles à la compréhension de la suite de ce chapitre. Nous commençons dans un premier temps par l'introduction des méthodes à noyaux en générale. Ensuite, nous nous focalisons sur les noyaux de séquences, l'objet de notre projet.

1.1 Méthodes à noyaux

Dans cette section, nous relions le concept d'apprentissage automatique aux méthodes du noyau, qui constituent l'un des moyens les plus importants d'analyser des données non linéaires. Ensuite, nous présentons les principes de base nécessaires.

1.1.1 Analyse des données non linéaires

Les méthodes classiques de l'apprentissage automatique, telles que l'analyse en composantes principales (ACP), l'analyse canonique des corrélations (ACC), la régression linéaire, l'analyse discriminante, la régression logistique ... sont utilisées efficacement seulement sur des données linéairement séparables. Mais dans la pratique, de nombreuses applications disposent de données non linéaires, d'où la nécessité de proposer une méthode qui nous permet de faire face à cette situation en exploitant les méthodes linéaires existants dans la littérature. Les méthodes à noyaux ont été proposées [Cristianini and Shawe-Taylor, 2000, Minsky and Papert, 1987] comme une solution efficace à de nombreux problèmes où les données sont non linéairement séparables. L'idée des méthodes à noyaux consiste à changer la représentation des données d'entrées sans oublier de garder les dépendances et les régularités inhérentes aux données (Figure 1.1). Ce changement de représentation de donnée se manifeste par une projection ϕ (Transformation) de puis un espace d'entrée vers un espace de redescription de haute dimension (Figure 1.1). Ainsi, les noyaux sont des mesures de similarité qui peuvent être utilisés avec tout algorithme d'apprentissage appliqué sur des données linéairement séparable. L'utilisation des méthodes à noyaux offre plusieurs avantages, notamment [Alex and Vishwanathan, 2008] :

- L'espace de redescription est suffisemment riche, alors qu'un simple séparateur (hyperplans) est suffisant pour séparées les données dans cet espace.
- Nous n'avons pas à formuler d'hypothèses sur l'espace d'entrée X autrement que pour qu'il s'agisse d'un ensemble.

- Les méthodes à noyaux peuvent être appliquées sur des données d'entrées quelque soit la nature de ces données.

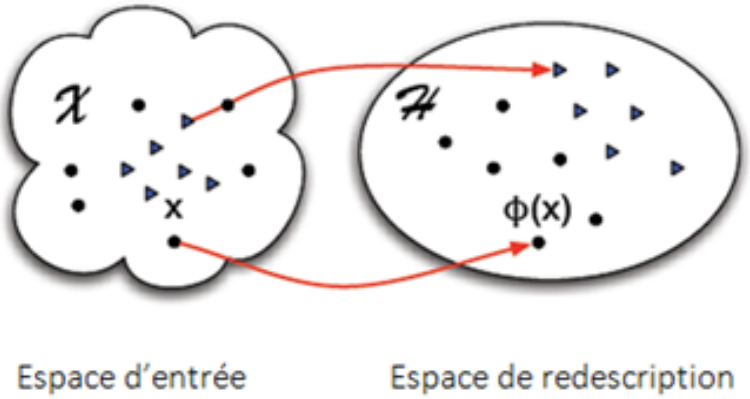

FIGURE 1.1 – Transformation non linéaire des données d'entrées via une projection ϕ. Extrait de [Alex and Vishwanathan, 2008]

1.1.2 Noyaux et modularité

Les méthodes à noyaux offrent une plateforme modulaire. Où ils traitent initialement des données par un noyau pour former une matrice de noyau (une matrice de Gram). Ensuite, ils utilisent plusieurs algorithmes d'apprentissage pour produire une fonction d'apprentissage. En d'autres termes, tous noyau peut être utilisé avec tous algorithme d'apprentissage. Cette propriété de modularité est illustrée par la (Figure 1.2).

Une matrice de Gram peut être définie comme suit : soit n vecteurs dans l'ensemble d'espace d'entrée X,$\mathbf{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, une matrice de Gram G associée à S est la matrice $n \times n$ où chaque entrée $G_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ représente le produit scalaire entre les vecteurs $\mathbf{x}_i$ et $\mathbf{x}_j$ [Bellaouar et al., 2018].

$$G = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \cdots & \langle \mathbf{x}_1, \mathbf{x}_N \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{x}_N, \mathbf{x}_1 \rangle & \cdots & \langle \mathbf{x}_N, \mathbf{x}_N \rangle \end{bmatrix}$$

Une matrice noyau peut être définie comme suit : soit n vecteurs dans l'ensemble d'espace d'entrée X, $\mathbf{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, une matrice noyau peut être définie comme suit : soit n vecteurs dans l'ensemble d'espace d'entrée $n \times n$ où chaque

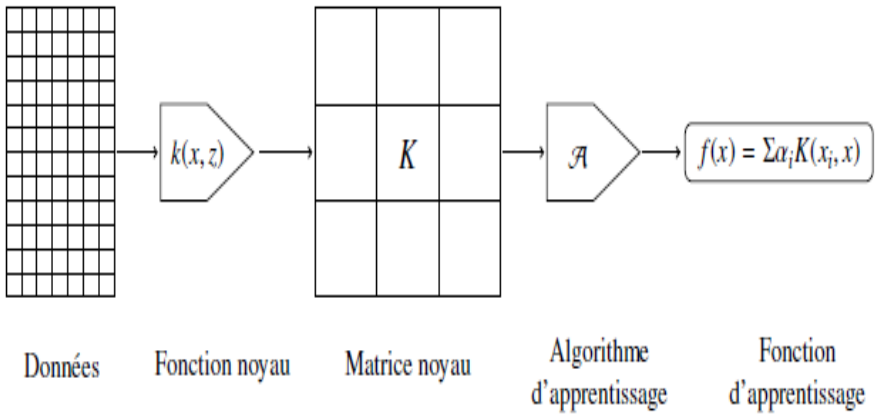

FIGURE 1.2 – Modularité dans les applications des méthodes à noyaux. Extrait de [Shawe-Taylor and Cristianini, 2004].

entrée représente le produit scalaire entre les images des vecteurs $\mathbf{x}_i$ et $\mathbf{x}_j$ par la fonction de projection Φ [Bellaouar, 2018]

$$G = \begin{bmatrix} \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle & \cdots & \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_N) \rangle \\ \vdots & \ddots & \vdots \\ \langle \Phi(\mathbf{x}_N), \Phi(\mathbf{x}_1) \rangle & \cdots & \langle \Phi(\mathbf{x}_N), \Phi(\mathbf{x}_N) \rangle \end{bmatrix} = \begin{bmatrix} \mathrm{K}(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \mathrm{K}(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \mathrm{K}(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \mathrm{K}(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

1.1.3 Validité des fonctions noyaux

Formellement, un noyau est valide, si et seulement si la fonction noyau k est une fonction semi-défini positive [Shawe-Taylor and Cristianini, 2004]. Une fonction est dite semi-définie positive si et seulement si elle est symétrique et la matrice noyau associeé est semi définie positive.
On dit qu'une matrice est semi-définie positive si toutes ses valeurs propres sont non négatives. Mais cette solution n'est pas facile à réaliser, particulièrement pour les matrices de hautes dimensions. Il existe des méthodes très rapides pour connaître si les valeurs propres sont positives. Ceci repose sur la définition d'une matrice semi définie positive.

Définition 1.1 [Strang, 2009, Mohri et al., 2012]. Une matrice $\mathbf{A}$ est semi-définie positive si $\mathbf{v}^T\mathbf{A}\mathbf{v} \geq 0, \mathbf{v} \neq 0$.

Proposition 1.1 Les matrices de Gram et de noyau sont semi-définies positives.

Nous considérons le cas des matrices de noyau :

$$\mathbf{G}_{ij} = \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \text{ , pour } j = 1, \ldots, n.$$

Pour chaque vecteur $\mathbf{v}$, nous avons :

$$\begin{aligned} \mathbf{v}^T \mathbf{G} \mathbf{v} &= \sum_{i,j=1}^{n} v_i v_j \mathbf{G}_{ij} = \sum_{i,j=1}^{n} v_i v_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{n} v_i \Phi(\mathbf{x}_i), \sum_{j=1}^{n} v_j \Phi(\mathbf{x}_j) \right\rangle \\ &= \left\| \sum_{i=1}^{n} v_i \Phi(\mathbf{x}_i) \right\|^2 \geq 0. \end{aligned}$$

1.1.4 Types de noyaux

Dans la littérature, il existe plusieurs types de noyaux, à savoir, le noyau linéaire, polynomial et gaussien ... [Alex and Vishwanathan, 2008] :

— Le noyau linéaire largement utilisé dans le domaine de fouille de texte, où les documents sont représentés par un vecteur contenant la fréquence d'apparition des mots, Les noyaux linéaires sont peut être les plus simples de tous les noyaux, et ils peuvent être définis comme est un produit scalaire simple :

$$k(x, x') =< x, x' >$$

.

— Pour le noyau polynomial, nous calculons tous les produits du $\mathbf{d}^{ième}$ ordre des entrées de x :

$$k(x, x') = (< x, x' >)^d$$

— Le noyau gaussien est un noyau très utilisé dans l'apprentissage automatique, qui s'évalue selon :

$$k(x, x') = \exp(-\frac{||x - x'||^2}{2\sigma^2}).$$

1.2 Noyaux pour le texte

Au cours des dernières décennies, le monde a été témoin du phénomène d'explosion de données, notamment les données textuelles, qui sont considérées

comme l'une des données les plus importantes. Cette quantité importante des données s'est présentée comme un handicape aux opérations d'analyses de classifications manuelles des données. Ceci a motivé la communauté de l'intelligence artificielle au développement de plusieurs algorithmes d'apprentissage automatique pour obtenir des résultats satisfaisants dans l'analyse et la classification de données.
Mais les algorithmes d'apprentissage automatique sont des algorithmes développés pour être appliqués sur des données vectorielles. Le problème que cette représentation n'est pas disponible à l'état brut de plusieurs types de données. Il faut capturer, implicitement ou explicitement, un espace de descripteurs plus approprié. Dans cet espace de haute dimension,Les algorithmes d'apprentissage automatiques ne peuvent pas être appliqués pour des raisons calculatoires. Dans la littérature, les noyaux pour le texte sont introduits selon la représentation des données. Nous pouvons considérer quatre représentations essentielles, à savoir, sacs de mots (VSM, Vector Space Model en anglais), séquences de symboles, vecteurs de concepts et sous forme de transducteurs pour lesquelles nous associons respectivement les noyaux des espaces vectoriels, sémantiques, de séquences et rationnels[Shawe-Taylor and Cristianini, 2004].

1.3 Noyaux de séquences

Noyaux de séquences La représentation de documents sous forme de sac de mots (VSM) ne suffit pas pour obtenir de meilleurs résultats de recherche, car il existe une perte d'informations sur la position des mots ou les séquences de caractères.
Pour pallier à ce problème, les noyaux de séquence ont été introduits. Dans la présente section, nous définissons les concepts de mots, séquences, sous-mots et sous-séquences[Shawe-Taylor and Cristianini, 2004].Par la suite, nous étudions quelques noyaux de séquences fréquemment utilisés. Les exemples d'illustration sont extraits des[Shawe-Taylor and Cristianini, 2004].

1.3.1 Mots et séquences

Un alphabet Σ est un ensemble fini de symboles, le nombre de symboles de Σ est symbolisé par $|\Sigma|$.
Un mot s est une séquence finie de symbole de Σ de longueur $|s|$, $s = s_1 \dots s_{|s|}$ où s_i le i^{e} symbole dans le mot s. Le mot vide est un mot de longeur 0 noté par ϵ.

Nous notons Σ^n l'ensemble de tous les mots de longueur n et Σ^* est l'ensemble de tous les mots qui peuvent être construits sur Σ .

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n.$$

Nous utilisons l'expression booléenne $[s = t]$pour définir la fonction qui renvoie 1 si les mots s et t sont identiques, et 0 sinon.
Nous notons $s \cdot t$ ou st la concaténation de deux mots s et t, où $|st|=|s| + |t|$. Le mot s^n représente la concaténation de s avec lui même n fois. Le mot $s(i : j)$ dénote le sous-mot $s_i s_{i+1} \dots s_j$ de s, où i $\leq$ j.
Autrement dit, le mot t est un sous-mot de s s'il existe u et v telles que $s = utv$ (u et v peuvent être vides).Si $u = \epsilon$, le mot t est un préfixe de s. Par contre, si $v = \epsilon$ t est dit suffixe de s. Le sous-mot ϵ est un préfixe et suffixe de tout mot. Les sous-mots de longueur k sont appelés k-grams ou k-mers.
Nous disons que t est une sous-séquence du mot s, s'il existe une séquence d'indices croissante $I = (i_1, \dots, i_{|t|})$ dans s, $(1 \leq i_1 < \dots < i_{|t|} \leq |s|)$ dans s,où tel que $t_j = s_{i_j}$, pour $j = 1, \dots, |t|$. La longueur de la sous-séquence t est le nombre des indices de I ($|I|$),notée $|t|$. Cependant, $l(I) = i_{|t|} - i_1 + 1$ détermine le nombre de symboles de s couverts par la sous-séquence t.
Par exemple, soit l'alphabet = $\{a, b, \dots, z\}$,et soit s = "informatique" un mot sur Σ. La longueur de s est $|s| = 12$, le mots $s(1 : 4)$ = 'info' est préfixe de s et s(8: 12) ="tique" est un suffixe de s $s(1 : 4)$, $s(8 : 12$) et $s(3 : 6)$="form" sont des sous-mots de s. Les mots $s(2 : 4 : 6)$ = "nom" et " $s(1 : 3 : 5)$ ="ifr" " sont des sous-séquences de longueur 3. Le mot $s(4 : 9)$ ="oi" est une sous-séquence de longueur 2, avec $l(4 : 9) = 6$.

1.3.2 Noyau p-spectre

Le noyau p-spectre ou p-gram [Leslie et al., 2002] est un noyau de séquences de caractères. Nous définissons le spectre d'ordre p (ou p-spectre) d'une séquence s l'histogramme de fréquence de tout sous mots (contiguë) de longueur p. La comparaison des p-spectres de deux mots peut donner une information importante au sujet de leur similarité dans les applications où la contiguïté joue un rôle important. Nous pouvons définir un noyau comme le produit scalaire de leur p-spectre

$$\Phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, \ u \in \Sigma^p.$$

Par le produit scalaire, le noyau p-spectre nous donne le nombre de sous mots de taille p appartenant à chacun des deux mots suggérés.

$$k_p(s,t) = \langle \Phi^p(s), \Phi^p(t) \rangle = \sum_{u \in \Sigma^p} \Phi^p_u(s) \Phi^p_u(t). \tag{1.1}$$

Exemple 1.1 [Noyau à 2-spectres] considérons les mots "bar ", "bat ", "car" et "cat ". Leurs 2 spectres sont donnés par Table 1.1.

TABLE 1.1 – Noyau 2- spectres entre les mots "bar ", "bat ", "car" et "cat ".

ϕ	ar	at	ba	ca
bar	1	0	1	0
bat	0	1	1	0
car	1	0	0	1
cat	0	1	0	1

Exemple 1.2 [Noyau à 3-spectres] nous prenons les deux mots suivantes :

$$s = "statistics"$$
$$t = "computation"$$

De ces deux mots nous pouvons extraire les sous-mots suivants de longueur 3 :
Sous-mots(s) : "sta","tat","ati","tis","ist","sti","tic","ics".
Sous-mots(t) : "com","omp","mpu","put","uta","tat","ati","tio","ion".
Notez que les mots s et t disposent de deux mots communs, "tat" et "ati". Par conséquent, leur produit scalaire sera $\mathbf{K}_3\ (s,t) = 2$. Il est claire que la complexité de calcul explicite du noyau p-specter est $O(|\Sigma^p|)$.
Dans une perspective d'amélioration de la complexité du noyau p-spectre, nous pouvons définir un noyau «auxiliaire» appelé noyau k-suffixe pour aider au calcul du noyau p-spectre. Le noyau k-suffixe $k^S_k(s,t)$ est défini comme suit [Shawe-Taylor and Cristianini, 2004] :

$$k^S_k(s,t) = \begin{cases} 1 & \text{si } s = s_1 u \text{ et } t = t_1 u, \text{ pour } u \in \Sigma^p\,; \\ 0 & \text{sinon.} \end{cases}$$

Le noyau p-spectre peut être exprimé en fonction de sa version K-suffixe comme

suit :

$$k_p(s,t) = \sum_{i=1}^{|s|-p+1} \sum_{j=1}^{|t|-p+1} k_p^S(s(i:i+p-1), t(j:j+p-1)). \quad (1.2)$$

À titre d'illustration, nous présentons l'exemple d'évaluation du noyau 3-spectre pour les mots s ="statistics" et t ="computation" en utilisant la récurrence de l'Equation 1.2.

Les tables Table (1.2) et Table (1.3) montre le processus d'évaluation respectivement le noyau k-suffixes et le noyau p-spectre entre les préfixes correspondants des deux mots.

TABLE 1.2 – Noyau 3- suffixes entre deux mots s="statistics" et t="computation".

$DP : k_3^S$	ε	c	o	m	p	u	t	a	t	i	o	n
ε	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	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
t	0	0	0	0	0	0	0	0	1	0	0	0
i	0	0	0	0	0	0	0	0	0	1	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0
i	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
s	0	0	0	0	0	0	0	0	0	0	0	0

En utilisant l'Equation 1.2 nous pouvons constater que le coût de noyau p-spectre est $O(p|s||t|)$.

1.3.3 Noyau toutes sous-séquences

Le noyau toutes sous-séquences [Shawe-Taylor and Cristianini, 2004] est défini comme : l'espace redescription associé à l'incorporation du noyau toutes

TABLE 1.3 – Noyau 3- specttres entre deux mots s="statistics" et t="computation".

DP : k_3	ε	c	o	m	p	u	t	a	t	i	o	n
ε	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	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
t	0	0	0	0	0	0	0	0	1	1	1	1
i	0	0	0	0	0	0	0	0	1	2	2	2
s	0	0	0	0	0	0	0	0	1	2	2	2
t	0	0	0	0	0	0	0	0	1	2	2	2
i	0	0	0	0	0	0	0	0	1	2	2	2
c	0	0	0	0	0	0	0	0	1	2	2	2
s	0	0	0	0	0	0	0	0	1	2	2	2

sous-séquences est indexé par $I = \Sigma^*$. L'incorporation donnée par :

$$\Phi_u(s) = |\{I : u = s(I)\}|, \ u \in \Sigma^*.$$

Autrement dit, c'est le nombre de fois où le mot u apparaît comme une sous-séquence dans le mot s. Le noyau associé est défini par :

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle = \sum_{u \in \Sigma^*} \Phi_u(s)\Phi_u(t). \tag{1.3}$$

L'exemple de la Table 1.4 présente toutes les sous-séquences des mots "bar","baa","car" et "cat". La Table 1.5 donne la valeur du noyau toutes sous-

TABLE 1.4 – Les sous-séquences des mots "bar", "baa", "car" et "cat".

ϕ	ε	a	b	c	r	t	aa	ar	at	ba	br	bt	ca	cr	ct	bar	baa	car	cat
bar	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0	0
bat	1	2	1	0	0	0	1	0	0	2	0	0	0	0	0	0	1	0	0
car	1	1	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0
cat	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1

séquences entre les mots "bar","baa","car" et "cat".

TABLE 1.5 – Noyau toutes sous-séquences entre les mots "bar", "baa", "car" et "cat".

ϕ	bar	bat	car	cat
bar	8	6	4	1
bat	6	12	3	3
car	4	3	8	4
cat	2	3	4	8

L'espace de redescription est indexé par Σ^*, alors il est claire que le calcul du noyau (produit scalaire) est coûteux, voire impossible. D'où la nécessite de chercher des alternatives.
Les auteurs de [Shawe-Taylor and Cristianini, 2004] ont essayer de faire appel à la récursivité pour aboutir à une méthode plus efficace. Le mécanisme proposé est détaillé par la suite : Nous commençons par la contribution d'une sous-séquence u dans le calcule du noyau :

$$\Phi_u(s)\Phi_u(t) = \sum_{I:u=s(I)} 1 \sum_{J:u=t(J)} 1 = \sum_{(I,J):u=s(I)=t(J)} 1 \tag{1.4}$$

En utilisant le résultat de l'Équation 1.4, Le noyau de l'Équation 1.3 peut être reformulé ainsi :

$$k(s,t) = \sum_{u\in\Sigma^*} \sum_{(I,J):u=s(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1. \tag{1.5}$$

L'idée de la récursion repose sur l'ajout d'un symbole a au mot s. Ainsi, l'Équation 1.5 devient :

$$\sum_{(I,J):sa(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1 + \sum_{u:t=uav} \sum_{(I,J):s(I)=u(J)} 1. \tag{1.6}$$

À partir de l'Équation 1.6, nous pouvons définir la récursion des noyaux toutes sous-séquences :

$$\begin{aligned} k(s,\epsilon) &= 1 \\ k(sa,t) &= k(s,t) + \sum_{j:t_j=a} k(s,t(1:j-1)). \end{aligned} \tag{1.7}$$

La propriété de la symétrie des noyaux permet une récursion analogue :

$$k(\epsilon, t) = 1$$
$$k(s, ta) = k(s, t) + \sum_{j:s_j=a} k(s(1 : j-1), t).$$

Pour montrer la manière de calculer le noyau toutes sous-séquences en faisant appel à la récursivité, nous présentons un exemple réponder dans la littérature. Soit les mots s ="gatt" et t ="cata ", où nous rajoutons le symbole a ="a" au mot "gatt" pour obtenir le mot sa ="gatta". Les lignes de la (Table 1.6) présentent les tuples (I, J) où $s(I) = t(J)$. Elles servent à calculer $k(s, t)$. Alors que les lignes de la(Table 1.7) présentent les sous-séquences commun entre sa et t qui se terminent par le symbole a ($sa(I) = t(J)$). Elles servent à calculer :

$$\sum_{J:t_j=a} k(s, t(1 : j-1)).$$

Les sous-séquences qui ne terminent pas par le symbole "a" entre les mots s= "gatta" et t = "cata".

TABLE 1.6 – Les sous-séquences qui se terminent par le symbole "a" dans les mots s = "gatta"et t = "cata".

g	a	t	a	$sa(i) = t(j)$	c	a	t	a
0	0	0	0	ϵ	0	0	0	0
0	1	1	0	at	0	1	1	0
0	1	1	0	at	0	1	1	0
0	1	1	0	a	0	1	0	0
0	0	1	0	t	0	0	1	0
0	0	1	0	t	0	0	1	0
0	1	1	0	a	0	0	0	1

Le total des lignes dans les (Table 1.6) et la (Table1.7) est 14, c'est à dire k(sa,t)=14 . La (Table 1.6) contient 7 lignes donnant ainsi $k(s, t) = 7$ plus k("gatt","c")= 1 donnée par la première rangée de la (Table 1.7), K("gatt","cat")=6 correspondant aux lignes 2 à 7 de la (Table 1.7).
Il est clair que l'évaluation récursive de ce noyau n'est pas efficace, Par conséquent, recours à une stratégie basée sur la programmation dynamique peut donner un résultat acceptable [Shawe-Taylor and Cristianini, 2004]. Cette tech-

TABLE 1.7 – Les sous-séquences qui se terminent par le symbole "a" dans les mots s = "gatta"et t = "cata".

g	a	t	a	$sa(i) = t(j)$	c	a	t	a
0	0	0	1	ϵ	0	0	0	0
0	1	0	1	a	0	1	0	0
0	1	0	1	aa	0	0	0	1
0	0	1	1	ta	0	1	0	1
0	0	0	1	ta	0	0	1	1
0	1	0	1	ata	0	1	1	1
0	1	1	1	ata	0	1	1	1

nique fait appel à un table de programmation dynamique (Table 1.8) où chaque ligne correspond à un symbole du mot s et chaque colonne correspond à un symbole de mot t. Une entrée $\mathbf{k}_{ij} = k(s(1:i), t(1:j))$ représente le noyau respectivement entre les préfixe $s(1:i)$ et $t(1:j)$ des mots s et t.

TABLE 1.8 – Table de programmation dynamique pour le calcul d'un noyau.

DP	ϵ	t_1	t_2	...	t_m
ϵ	1	1	1	...	1
s_1	1	k_{11}	k_{12}	...	k_{1m}
s_2	1	k_{21}	k_{22}	...	k_{2m}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_n	1	k_{n1}	k_{n2}	...	k_{nm}

La première colonne et la première ligne contiennent la valeur 1 qui indique le cas de base de la récurrence donnée par les Équation 1.7 et 1.8, les entres (i, j)sont calculées comme étant la somme de l'entrée $(i-1, j)$, avec toutes les entrées $(i-1, k-1)$ avec 1 $\leq$k<j dont $\mathbf{t}_k = \mathbf{s}_i$. L'exemple de la (Table 1.9) montre la technique de programmation dynamique pour le calcul du noyau toutes sous-séquences entre les mots s = "gatta" et t = "cata". Cette évaluation nécessite $O(j)$ parce que nous devons vérifier tous les symboles de t jusqu'à la position j. Le renseignement de la table de programmation dynamique nécessite donc $O(|s|\,|t|^2)$. Une solution acceptable mais qui nécessite encore des améliorations.

TABLE 1.9 – Noyau toutes sous-séquences entre les mots "bar", "baa", "car" et "cat".

DP	ϵ	g	a	t	a	a
ϵ	1	1	1	1	1	1
c	1	1	1	1	1	1
a	1	1	2	2	2	3
t	1	1	2	4	6	7
a	1	1	3	5	7	14

Il n'est pas difficile de constater qu'on aura besoin de $O(j)$ lors de remplissage de chaque ligne i de la table de le programmation dynamique :

Une astuce consiste à faire un pré-calcul dans un tableau P pour l'exploiter lorsque nous arrivons à la position j (Algorithme 1)

Algorithm 1: Calcul et stockage des sommes intermédiaires dans un tableau P.

```
m ← length(t)
last ← 0
P[0] ← 0
for k = 1 : m do
    P[k] ← P[last]
    if t_k = s_i then
        P[k] ← P[last] + DP[i − 1, k − 1]
        last ← k
```

FIGURE 1.3 – Calcul et stockage des sommes intermédiaires dans un tableau,Extrait de [Bellaouar, 2018]

En utilisant le tableau P nous pouvons remplir la ligne $i + 1$ en utilisant la boucle simple de l'Algorithme 2.

Algorithm 2: Remplissage de la ligne suivante dans la table dynamique.

```
for k = 1 : m do
    DP[i, k] ← DP[i − 1, k] + P[k]
```

FIGURE 1.4 – Remplissage de la ligne suivante dans la table dynamique,Extrait de [Bellaouar, 2018].

L'évaluation complète du noyau toutes sous-séquences est donnée par l'Algorithme 3.

Algorithm 3: Évaluation du noyau toutes sous-séquences.

Input: Les mots s et t.
Output: Valeur du noyau $k(s,t) = DP\,[n,m]$.

```
1  n ← length(s)
2  m ← length(t)
3  for j = 1 : m do
4      DP[0, j] ← 1
   /* Traitement des lignes                                  */
5  for i = 1 : n do
       /* pré-calcul et remplissage du tableau P             */
6      last ← 0
7      P [0] ← 0
8      for k = 1 : m do
9          P [k] ← P [last]
10         if t_k = s_i then
11             P [k] ← P [last] + DP[i − 1, k − 1]
12             last ← k
       /* renseignement de la ligne suivante                 */
13     for k = 1 : m do
14         DP[i, k] ← DP[i − 1, k] + P[k]
```

FIGURE 1.5 – Évaluation du noyau toutes sous-séquences,Extrait de [Bellaouar, 2018].

L'analyse de l'Algorithme 3 peut être détaillée comme suit :

— le temps d'exécution de la boucle de 3 à 4 est $O(|t|)$.

— la boucle de 5 à 14 qui comporte deux boucles internes de 8 à 12 et de 13 à 14 s'exécute dans un temps $O(|s|\,|t|)$.

— la complexité totale de l'algorithme 3 est $O(|s|\,|t|)$, qui peut être considéré comme une bonne amélioration comparée à la solution brute.

L'amélioration présentée par l'Algorithme 3 peut être expliquée par l'exemple de la Table 1.10.

1.3.4 Noyau sous-séquences de longueur fixe

L'augmentation de la dimension dans l'espace de redescription entraînait des coûts élevés pour le temps d'exécution ; par conséquent, les améliorations incluent la réduction de cette dimension par une longueur fixe p des sous-séquences.
Ainsi, l'espace de redescription est indexé par toutes les sous-séquences de longueur fixe p. L'incorporation associée à un mot est donnée par :

TABLE 1.10 – Table de la programmation dynamique pour le calcul du noyau toutes sous-séquences entre les modt s = "cata" et t="gataa" avec le pré-calcul.

DP	ϵ	g	a	t	a	a
ϵ	1	1	1	1	1	1
p	0	0	0	0	0	0
c	1	1	1	1	1	1
p	0	0	1	1	1	2
a	1	1	2	2	2	3
p	0	0	0	2	4	4
t	1	1	2	4	6	7
p	0	0	1	1	1	7
a	1	1	3	5	7	14

$$\Phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, \ u \in \Sigma^p.$$

où la composante u de longueur P représente le nombre d'occurrences de u dans le mot s.
Le noyau associé est défini comme suit :

$$k_p(s,t) = \langle \Phi^p(s), \Phi^p(t) \rangle = \sum_{u \in \Sigma^p} \Phi_u^p(s) \Phi_u^p(t). \tag{1.8}$$

Pour une évaluation efficace de ce noyau, nous utilisons la même démarche du noyau toutes sous-séquences (Section 1.3.3). Par conséquent, le noyau toutes sous-séquences de longueur p est exprimé par :

$$\sum_{(I,J) \in I_p \times I_p : sa(I) = t(J)} 1 = \sum_{(I,J) \in I_p \times I_p : s(I) = t(J)} 1 + \sum_{u : t = uav} \sum_{(I,J) \in I_{p-1} \times I_{p-1} : s(I) = u(J)} 1. \tag{1.9}$$

Comme nous avons utilisé l'idée de récursion décrite par l'Équation 1.6, nous pouvons obtenir :

$$\begin{aligned} k_0(s,t) &= 1, \\ k_p(s,\epsilon) &= 0, \text{ pour } p > 0, \\ k_p(sa,t) &= k_p(s,t) + \sum_{j : t_j = a} k_{p-1}(s, t(1 : j-1)). \end{aligned}$$

Algorithm 4: Évaluation du noyau sous-séquences de longueur p.

Input: Les mots s et t et la longueur p des sous-séquences.
Output: Valeur du noyau $k_p(s,t) = DP\,[n,m]$.

```
1  n ← length(s)
2  m ← length(t)
3  for j = 0 : m do
4      DP[0, j] ← 1
5  for i = 0 : n do
6      DP[i, 0] ← 1
7  for l = 1 : p do
8      DP_prec ← DP
9      for j = 1 : m do
10         DP[0, j] ← 1
11     for i = 1 : n-p+1 do
12         last ← 0
13         for k = 1 : m do
14             P [k] ← P [last]
15             if t_k = s_i then
16                 P [k] ← P [last] + DP_prec[i − 1, k − 1]
17                 last ← k
           /* renseignement de la ligne suivante          */
18         for k = 1 : m do
19             DP[i, k] ← DP[i − 1, k] + P[k]
```

FIGURE 1.6 – algorithme Evaluation du noyau sous-séquences de longueur p. Extrait de [Bellaouar, 2018].

Il est claire que l'implémentation directe de la récursion de l'Équation 1.9. n'aboutit pas à une solution efficace. Le recours à la technique de la programmation dynamique peut s'avérer nécessaire.L'Algorithme 4 illustre cette idée de programmation dynamique. L'analyse de L'Algorithme (Figure1.6) méne à une complexité $O(p\,|s|\,|t|)$ du noyau toutes sous-séquences de longueur p.

1.3.5 Noyau sous-séquences de mots

Lors de l'étude des noyaux qui traitent les sous-séquences, nous notons qu'ils les traitent de la même manière, sans prendre en considération la distance séparant les éléments non contigües (trous). Cependant, nous pouvons considérer la première occurrence comme plus importante par rapport aux autres occurrences, car elle est plus contigüe. Pour faire fâçe à cette situation, Lodhi et al. proposent le noyau sous-séquence de mots (SSK, String Subsequence Kernel) qui pénalise les sous-séquences selon les trous qu'elle comprennent [Lodhi et al., 2002]. Le noyau SSK utilise un paramétre de pénalisation $\in]0.1]$ pour mesurer la distance des éléments non contiguës des sous-séquences. L'espace de redecsription est indexé par toutes les sous-séquences de longueur p. L'incorporation $\phi_u^p(s)$ d'une sous-séquence $u \in \Sigma^p$ donnée par :

$$\phi_u^p(s) = \sum_{I:u=s(I)} \lambda^{l(I)},\ u \in \Sigma^p.$$

Le noyau associé peut être écrit comme suit :

$$\begin{aligned} K_p(s,t) &= \langle \phi^p(s), \phi^p(t) \rangle \\ &= \sum_{u \in \Sigma^p} \phi_u^p(s).\phi_u^p(t) \\ &= \sum_{u \in \Sigma^p} \sum_{I:u=s(I)} \sum_{J:u=t(J)} \lambda^{l(I)+l(J)}. \end{aligned} \tag{1.10}$$

titre d'exemple illustratif pour l'idée de ce noyaux, nous considérons les mots bar, bat, car et cat ; pour des sous-séquences de longueur p = 2. La Table 1.11 montre la projection de ces mots dans l'espace de redescription.

TABLE 1.11 – Projection des mots *bar*, *bat*, *car* et *cat* dans un espace de redescription pour des sous-séquences de longueur $p = 2$.

ϕ_u^2	ar	at	ba	br	bt	ca	cr	ct
bar	λ^2	0	λ^2	λ^3	0	0	0	0
bat	0	λ^2	λ^2	0	λ^3	0	0	0
car	λ^2	0	0	0	0	λ^2	λ^3	0
cat	0	λ^2	0	0	0	λ^2	0	λ^3

La valeur du noyau entre *bar* et *bat* est $K_2(bar, bat) = \lambda^4$, tandis que, la version normalisée est obtenue par :

$$\widehat{K}_2(bar, bat) = K_2(bar, bat)/\sqrt{K_2(bar, bar).K_2(bat, bat)} = \lambda^4/(2\lambda^4+\lambda^6) = 1/(2+\lambda^2).$$

L'implémentation directe de ce noyau donne une complexité temporelle et spatiale $O(|\Sigma^p|)$,Où $|\Sigma^p|$ est la dimension de l'espace de redescription.
Comme pour les noyaux précédents, on peut utiliser la récursivité pour améliorer l'évaluation du noyau. Ceci est fait en définissant un noyau de suffixe dont la projection est définie par :

$$\phi_u^{p,S}(s) = \sum_{I \in I_p^{|s|}:u=s(I)} \lambda^{l(I)},\ u \in \Sigma^p,$$

Où I_p^k est l'ensemble des p-tuples des indices I avec $i_p = k$ Autrement dit, l'en-

semble des sous-séquences de longueur p dont le dernier symbole est identique au dernier symbole du mot s. Le noyau associé est défini comme :

$$\begin{aligned} K_p^S(s,t) &= \langle \phi^{p,S}(s), \phi^{p,S}(t) \rangle \\ &= \sum_{u \in \Sigma^p} \phi_u^{p,S}(s).\phi_u^{p,S}(t). \end{aligned}$$

Ainsi, le noyau SSK de la version suffixe peut être exprimé comme suit :

$$K_p(s,t) = \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} K_p^S(s(1:i), t(1:j)), \tag{1.11}$$

Avec pour $p = 1$, $K_1^S(s,t) = [s_{|s|} = t_{|t|}]\ \lambda^2$. Maintenant, nous pouvons introduire un noyau récursif pour la version suffixe afin de calculer la similarité entre deux mots (sa et tb), si leurs symboles finaux $a = b$, comme suit :

$$K_p^S(sa,tb) = [a=b] \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} \lambda^{2+|s|-i+|t|-j} K_{p-1}^S(s(1:i), t(1:j)). \tag{1.12}$$

L'implémentation naïve de cette récursivité du noyau SSK conduit à une complexité $O(p(|s|^2|t|^2))$ qui est considérée insatisfaisante. Alors, il est intéréssant de considérer des implémentations plus efficaces.
Dans la littérature, il existe plusieurs approches qui calculent efficacement le noyau SSK. Parmi ces méthodes : les approches de la programmation dynamique, la programmation dynamique éparse, à base de trie et à base géométrique.
Nous décrivons brièvement les approches sus-citées :
L'approche de la programmation dynamique [Lodhi et al., 2002] utilise une table de programmation dynamique et donnant une complexité de calcul SSK en $O(p\,|s||t|)$.
À cause du fait que la plupart des entrées de la table de la programmation dynamique sont nulles, donc ne contribuent pas au résultat, l'approche de la programmation dynamique éparse propose une solution basée sur la technique de la programmation dynamique éparse [Rousu and Shawe-Taylor, 2005]. Les auteurs utilisent deux structures de données. La première est un arbre de somme d'intervalles (RST , Range Sum Tree) pour remplacer la table de la programmation dynamique, utilisée pour renvoyer efficacement la somme de n valeurs

dans un intervalle. La seconde structure de donnée est un ensemble de listes de correspondances l_i pour calculer le noyau des suffixes. La complexité de calcul du noyau est ,$O(p|l_1|logn)$, avec $|l_1|$ est la taille de la liste la plus longue et $n = min(|s|, |t|)$.
L'approche à base de trie proposée par E. Fredkin est basés sur des arbres de recherche. L'idée générale de cette approche est que les feuilles de l'arbre représentent l'espace de redescription qui indexé par l'ensemble des mots de longueur p sur un alphabet Σ. L'évaluation de cette approche dans le pire des cas est $O(\binom{p+g_{max}}{g_{max}} (|s| + |t|))$, où g_{max} est le gap (trous) maximum autorisé.
L'approche à base géométrique [Bellaouar et al., 2014] est proposée pour contourner les inconvénients de l'approche de la programmation dynamique. Pour se faire, elle utilise une liste linéaire chaînée de correspandance conjointement avec un arbre d'intervalles en couches (LRT , Layered Range Tree) menant à une complexité de calcul du noyau SSK en $O(|L|log|L| + pK)$,où $|L|$ est la taille de la liste de K est le total des points rapportés par le LRT pour toutes les entrées de la liste. Ensuite,[Bellaouar et al., 2018] ont amélioré l'approe à base géométrique en étandant le LRT en un LRST (Layered Range Sum Tree) qui calcul efficacement la somme de n point dans un intervalle rectangulaire. Ainsi, il ont atteint un comlexité de $O(p|L|Log|L|)$.
Les noyaux sous-séquences de mots ont été utilisées dans de nombreuses applications. À titre d'exemple :

- la Classification de texte [Lodhi et al., 2002, Joachims, 1998] ; [Cancedda et al., 2003] , [Nehar et al., 2014] , [Althubaity et al., 2008] .
- Détection de malwares [Pfoh et al., 2013].
- Classification des séquences protéiques [Zaki et al., 2005],
- Analyse sémantique [Kate and Mooney, 2006]
- Lambda pruning pour la classification et le clustering SVM[Seewald and Kleedorfer, 2007]
- Extraction de l'information relationnelle [Mooney and Bunescu, 2006]

1.4 Conclusion

Dans ce chapitre nous avons commencé par l'introduction du concept noyau, notament l'aspect prise en charge de traitement des données non linéairement séparable.
avons commencé par l'introduction du concept noyau, notament l'aspect prise

en charge de traitement des données non linéairement séparable.
Par la suite nous avons décrit une catégorie des noyaux largement utilisée, les noyaux de séquences (p-spectre, toutes sous-séquences, sous-séquences de longeur fixe et sous-séquences de mots).
Pour chaque noyau, nous nous somme focalisés sur l'aspect efficacite de l'implémentation qui passe généralement par la programmation dynamique.

Chapitre 2

Unification et généralisation des noyaux de séquences

Les chercheurs travaillent depuis des milliers d'années sur le principe de l'unification universelle. Ce travail continu pour la théorie de l'unification se justifie par un souhait de découvrir le secret de l'univers naturel [Hoare, 1996]. Cependant, l'idée d'unifier les théories de l'informatique en général [Hoare, 1996] et de la fouille de données en particulier [Yang and Wu, 2006] est très difficile. Nous avons découvert (chapitre 01), qu'il existe une diversite de noyaux de séquences. À savoir, le noyau p-spectre, le noyau toutes sous-séquences, le noyau sous-séquences de longueur fixe et le noyau sous séquences de mots (SSK). Chaque noyau s'intéresse à un problème particulier donnant naissance à plusieurs approches. Dans la théorie d'unification de l'apprentissage automatique, [Bellaouar et al., 2017] proposent une plate-forme générale pour calculer les noyaux de séquence, qui peut être aussi utile pour évaluer des noyaux d'ensembles de séquences.
Dans le présent chapitre, nous commençons par partie préliminaires qui consiste à introduire des concepts d'automates pondérés ainsi que les séries formelles. Ensuite, nous abordons deux aspects. Le premier concerne l'unification, où nous décrivons le modèle GASWA (Gappy All Subsequence Weighted Automaton) qui est une plate-forme générale pour le calcul des noyaux de séquence.
En outre, nous discutons l'aspect généralisations, dans le but est d'étendre le modèle GASWA afin de prendre en compte un ensemble de mots au lieu d'un mot.

2.1 Préliminaires

Cette section est dédiée pour introduire les informations préliminaires dont nous avons besoin dans la suite de ce chapitre.

2.1.1 Semi annaux

Le semi annaux [Kuich and Salomaa, 1986] est une structure algébrique $(\mathbb{S}, +, \cdot, 0, 1)$ possédant les propriétés suivantes :

1. $(\mathbb{S}, +, 0)$ est un monoïde commutatif ;
2. $(\mathbb{S}, \cdot, 1)$ forme un monoïde ;
3. l'opération $(.)$ est distributive par rapport à $(+)$;
4. pour tout $a \in \mathbb{S} : 0 \cdot a = a \cdot 0 = 0$.

2.1.2 Automate fini

Un automate fini A sur Σ est défini par un quintuplet (Σ, Q, E, I, F). Où Σ est un alphabet fini, Q est un ensemble fini d'états, $E \subseteq Q \times \Sigma \times Q$ est un ensemble de transitions, I, $F \subseteq Q$ sont respectivement un ensemble fini d'états initiaux et un ensemble fini d'états finaux (ou acceptant) [Kuich and Salomaa, 1986].
Pour chaque transition $e = (p, a, s, q) \in E$, il y a son état origine ou source $o(e) = p$, son étiquette $l(e) = a$ et son état destination ou arrivé $d(e) = q$.
Un chemin π dans un automate fini A est une suite finie de transitions consécutives $e_1 e_2 \dots e_n$. C'est-à-dire, l'origine de chaque transition est la destination de la précédente. On dit qu'un chemin π dans l'automate A est réussi, s'il commence par un état $i \in I$ et se termine par un état $f \in F$ [Hopcroft and Ullman, 1979].
Un automate fini est représenté comme un graphe orienté étiqueté. Les états sont représentées par des cercles, à l'intérieur de chaque état son nom. Un état initial est signalé par une petite flèche qui pointe sur l'état et un états final est représenté par Deux cercles. Les transitions sont représentées par des flèches qui partent d'un état origine vers un état destination, l'étiquette de la transition est indiquée au milieu de la flèche (La figure 2.1).
Un automate fini acyclique est un automate fini ne contenant aucun cycle.

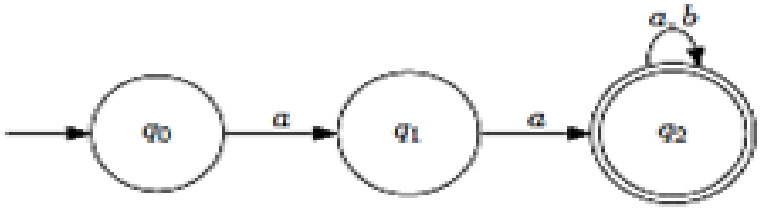

FIGURE 2.1 – Automate représentant le langage des mots sur { a,b } préfixés par aa.

2.1.3 Automate pondéré

Un automate pondéré est un automate fini basé sur la notion de semi-anneau ainsi que dont les arêtes ont un poids. Plus précisément, un automate pondéré A sur $\mathbb{S}$ est un 7-uplet $(\Sigma, Q, E, I, F, \lambda, \rho)$, où $\mathbb{S}$ est un semi-anneau, Σ est un alphabet fini, Q est un ensemble fini d'états, $E \subseteq Q \times \Sigma \times S \times Q$ est un ensemble de transitions, I et $F \subseteq Q$ sont respectivement un ensemble fini d'états initiaux et un ensemble fini d'états finaux (ou acceptant), $\lambda : I \to S$

et $\rho : F \to S$ sont respectivement de pondération pour les états initiaux et de pondération pour les états finaux.
Pour chaque transition $e = (p, a, s, q) \in E$, il y a son état origine ou source $o(e) = p$, son étiquette $l(e) = a$, son poids ou coefficient $wt(e) = s$ et son état destination ou arrivé $d(e) = q$. Nous considérons si le coefficient s est égal 0 qu'il n'existe pas une transition entre les états p et q [Sakarovitch and Thomas, 2009].
Un chemin dans l'automate A est une suite finie de transitions $\pi = e_1 e_2 \dots e_n$. Où l'état origine $o(\pi) = o(e_1)$, l'étiquette $l(\pi) = l(e_1)l(e_2)\dots l(e_n)$ est le produit des étiquettes des transitions, le poids $wt(\pi) = wt(e_1) \cdot wt(e_2) \dots \cdot wt(e_n)$ est le produit des poids de ses transitions et l'état destination $d(\pi) = e_n$. On dit qu'un chemin π dans l'automate A est réussi (calcul), s'il commence par un état $i \in I$ et se termine par un état $f \in F$ [Hopcroft and Ullman, 1979].
Le poids d'un calcul est le poids du chemin $wt(\pi)$ multiplié respectivement à gauche et à droite par le poids initial de l'état de départ et le poids final de l'état d'arrivée.

2.1.4 Série formelle

Pour un alphabet Σ et un semi-anneau $\mathbb{S}$, une série formelle r est une application de $\Sigma^* \to \mathbb{S}$. On dénoté par (r, w) l'image d'un mot $w \in \Sigma^*$ par l'application r, qui appelée le coefficient de w dans r. La série r, elle-même, est exprimée comme une somme formelle :

$$r = \sum_{w \in \Sigma^*} (r, w)w. \tag{2.1}$$

Le langage $supp(r) = \{w \in \Sigma^* | (r, w) \neq 0\}$, c'est le support de la série r. On dénote par $\mathbb{S}\langle\langle\Sigma\rangle\rangle$ l'ensemble de séries formelles sur un alphabet Σ à coefficients dans un semi-anneau $\mathbb{S}$. Un polynôme est défini comme une série avec un support fini. L'ensemble des polynômes est dénotée par $\mathbb{S}\langle\Sigma\rangle$.
Soit une structure d'un semi-anneau sur $\mathbb{S}\langle\langle\Sigma\rangle\rangle$, où pour tout $r_i \in \mathbb{S}\langle\langle\Sigma\rangle\rangle$et $s \in \mathbb{S}$, les opérations suivantes sont définies comme suit (Table 2.1) [Kuich and Salomaa, 1986] :

L'ensemble des polynômes $\mathbb{S}\langle\Sigma\rangle$ est un sous semi-anneau de $\mathbb{S}\langle\langle\Sigma\rangle\rangle$.

Opération	**Définition**
la somme	$(r_1 + r_2, w) = (r_1, w) + (r_2, w)$
le produit scalaire	$(sr_1, w) = s(r_1, w)$ et $(r_1 s, w) = (r_1, w)s$
le produit de Cauchy	$(r_1 \cdot r_2, w) = \sum_{w_1 w_2 = w} (r_1, w_1)(r_2, w_2)$
le produit d'Hadamard	$(r_1 \odot r_2, w) = (r_1, w)(r_2, w)$

TABLE 2.1 – Table représenté les définition des opérations.

2.2 Modèle à base d'automates pondérés

Dans cette section, nous nous focalisons sur la présentation de la plate-forme générale dédiée pour l'unification des noyaux de séquences [Bellaouar et al., 2017] . Cette plate-forme générale peut être utile pour le traitement des noyaux d'ensembles de séquences. Nous discutons aussi l'aspect efficacité de calcul du noyau et la robustesse du modèle proposé.

2.2.1 Idée principale

L'idée générale de la plate-forme générale proposée par [Bellaouar et al., 2017] est basée sur la relation entre les trois concepts suivants : « noyau », « séries formelles » et « automates pondérés ». En effet, les auteurs utilisent le noyau toutes sous-séquences avec trous comme un cas générale. L'espace de redescription de ce noyau est indexé par toutes les sous-séquences de Σ^*. La fonction de projection $\Phi_w(s)$ pour une sous-séquence w dans un mot s est représentée comme pour la méthode du noyau SSK par :

$$\Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)},\ w \in \Sigma^*. \tag{2.2}$$

Notons que L'Équation (2.2) est une application de Σ^* dans $\mathbb{R}$, donc elle définit une série formelle r_s, où le coefficient de la série $(r_s, w) = \sum_{I:w=s(I)} \lambda^{l(I)}$. Par conséquence, L'Équation 2.3 définissant cette série formelle est considérée comme une nouvelle représentation du mot s dans l'espace de redescription.

$$r_s = \sum_{w \in \Sigma^*} \sum_{I:w=s(I)} \lambda^{l(I)} w \tag{2.3}$$

Ainsi, la série formelle de L'Équation 2.3 peut être vue comme le compor-

tement $\|A\|$ d'un automate pondéré réalisant la série formelle. Autrement dit, le poids de toutes les sous séquence w dans le mot s. Avec la supposition que les fonctions de pondérations initiale et finale λ et ρ sont égaux à 1.

$$\|A\| : \Sigma^* \to \mathbb{S},$$

avec

$$(\|A\|, w) = \sum_{\pi : l(\pi) = w} wt(\pi) = \sum_{I : w = s(I)} \lambda^{l(I)} = \Phi_w(s). \tag{2.4}$$

De ce qui précède, nous concluons que la projection des mots dans un espace de redescription liée au noyau toutes sous-séquences avec trous, peut être représenté par une série formelle qui peut être réalisée par un automate pondéré. Ainsi pour une évaluation efficace des noyaux de séquences, il est nécessaire de procéder à une réalisation compacte des automates pondérés correspondants.

2.2.2 Gappy All Subsequence Weighted Automaton

Dans cette section, nous montrons la réalisation de la série formelle qui représente la projection associée au noyau toutes sous-séquences avec trous (Équation 2.3) par une construction d'un automate pondéré compact et non déterministe. Cette construction est dite GASWA, pour Gappy All Subsequence Weighted Automaton [Bellaouar et al., 2017].

Soit un mot $s = s_1 s_2 ... s_n$ sur Σ^*. L'automate GASWA est un automate pondéré associée au noyau toutes sous-séquences avec trous (figure 2.2), défini par un 7-uplet $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$, où :

— Σ est un alphabet fini ;

— $Q_s = \{q_0\} \cup \{q_i, q'_i \mid 1 \leq i \leq n\} \cup \{q''_i \mid 1 \leq i \leq n-1\}$;

— $E_s = \{(q_0, \lambda, s_1, q_1), (q_0, \lambda, s_1, q'_1), (q_0, 1, \epsilon, q''_1)\} \cup \{(q'_i, \lambda, s_{i+1}, q_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(q'_i, \lambda, s_{i+1}, q'_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(q'_i, \lambda, \epsilon, q'_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(q''_i, \lambda, s_{i+1}, q_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(q''_i, 1, \epsilon, q''_{i+1}) \mid 1 \leq i < n-2\} \cup \{(q''_i, \lambda, s_{i+1}, q'_{i+1}) \mid 1 \leq i \leq n-1\}$;

— $I_s = \{q_0\}$;

— $F_s = \{q_i \mid 0 \leq i \leq n\}$;

— Les fonctions de pondération initiale et finale (λ et ρ) sont des scalaires égaux à 1.

L'automate GASWA est défini sur le semi-anneau des probabilités $\langle \mathbb{R}_+, +, \times, 0, 1 \rangle$. On peut distinguer trois niveaux d'états $q_0, q'_1, \ldots, q'_n$; $q_0, \ldots, q_n$ et $q_0, q''_1, \ldots, q''_{n-1}$. Où Le premier niveau représente le mot s et la valeur λ^1 est le poids de la transition pondérant la distance entre deux symboles consécutifs dans le mot s. Mais pour autoriser les trous, le premier niveau est étendu par des ϵ-transitions : $e_i = (q'_i, \epsilon, \lambda, q'_{i+1})$, pour $1 \leq i < n-1$ Avec une pénalité λ pour chaque trou. Pour mettre fin à une sous-séquence à un symbole donné, nous devons passer à partir du premier niveau au second par l'ajout des transitions : $e_i = (q'_i, s_{i+1}, \lambda, q_{i+1}),\ i = 0..n-1$, pour $1 \leq i < n-1$ avec $q'_0 = q_0$.
D'autre part, pour commencer par n'importe quel symbole de le mot s, nous devons créer les transitions de troisième niveau : $e_i = (q''_i, \epsilon, \lambda^0, q''_{i+1}),\ i = 0 \ldots n-2$, pour $0 \leq i < n-2$ avec $q''_0 = q_0$. Où l'absence d'un trou ou d'une distance entre deux symboles consécutifs est indiquée par le poids λ^0.
Puis, de l'état q''_i, pour $1 \leq i < n-1$ nous pouvons décider soit de terminer une sous-séquence avec la transition $e_i = (q''_i, s_{i+1}, \lambda, q_{i+1})$, ou bien continuer la sous-séquence avec la transition $e_i = (q''_i, s_{i+1}, \lambda, q'_{i+1})$. La Figure 2.2 illustre la construction de GASWA.

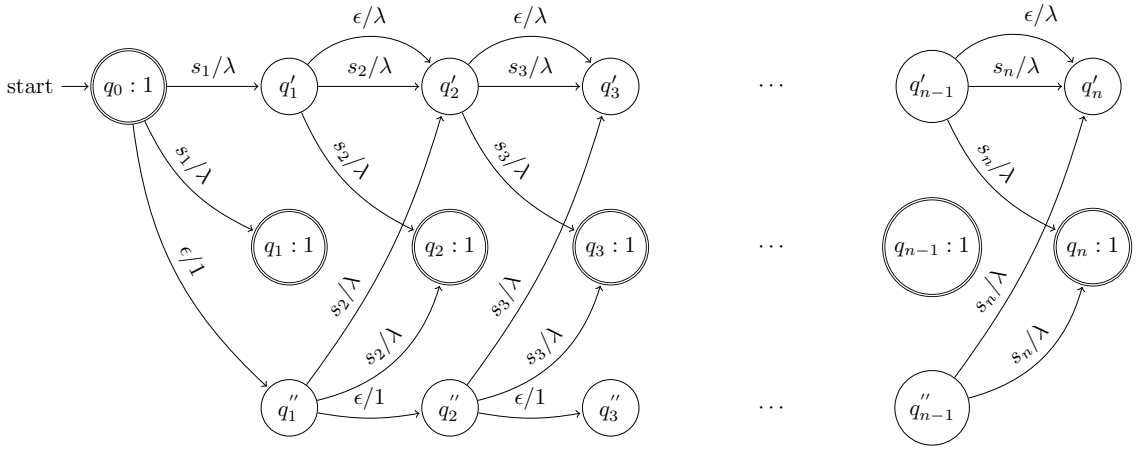

FIGURE 2.2 – Automate pondéré pour toutes les sous-séquences d'un mot $s = s_1 s_2 \ldots s_n$.

À titre exemple, nous construisons un GASWA pour le mot $s = cata$. On peut représenté l'espace de redescription et la projection associé à s respectivement par la Table 2.2 et la série formelle de l'Équation 2.5.

$$r_s = 1\epsilon + \lambda c + 2\lambda a + \lambda t + (\lambda^2 + \lambda^4)ca + \lambda^2 at + \lambda^3 aa + \lambda^3 cat + \lambda^4 caa + \lambda^3 ata + \lambda^4 cata. \tag{2.5}$$

La série formelle de l'Équation 2.5 peut être réalisée par l'automate GASWA

TABLE 2.2 – Espace de redescription du mot $s = cata$ associé au noyau toutes sous-séquences avec trous.

Sous-séquence	ϵ	c	a	t	ca	ct	at	aa	cat	caa	ata	cata
coefficient	1	λ	2λ	λ	$\lambda^2 + \lambda^4$	λ^3	λ^2	λ^3	λ^3	λ^4	λ^3	λ^4

de la Figure 2.3.

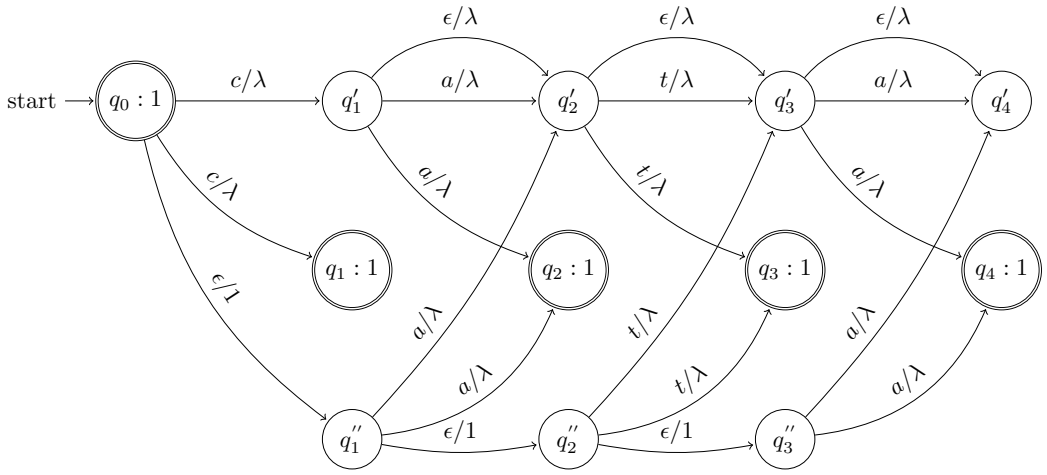

FIGURE 2.3 – Automate pondéré pour toutes les sous-séquences du mot $s = cata$.

2.2.3 Validité et Efficacité de GASWA

Pour démontrer la validité et l'efficacité de la construction GASWA, les auteurs qui ont proposé la construction ont commencé par la caractérisation des séries formelles réalises au niveau des états finales de l'automate GASWA, par le biais de la Proposition 2.1.

Proposition 2.1 [Bellaouar et al., 2018] Soit $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ un automate pondéré GASWA associé au mot $s = s_1 s_2 \dots s_n$. Nous dénotons par $\mathfrak{S}_k$, $k = 0..n$ la série formelle réalisée dans l'état q_k qui peut être formalisée comme suit :

$$\mathfrak{S}_k = \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w, \tag{2.6}$$

I^k dénote les indices des tuples I où $i_k = s_k$.

Le résultat de la proposition est utilisé pour démontrer la validité de le construc-

tion GASWA (Lemme 2.1).

Lemme 2.1 [Bellaouar et al., 2018] Soit $s = s_1 s_2 \dots s_n$ un mot sur Σ^*. L'automate GASWA $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ réalise la série formelle associée à l'espace de redescription de la projection correspondante au noyau toutes sous-séquences avec trous donnée par :

$$\Phi : s \mapsto \Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)}, w \in \Sigma^*.$$

En suit, ils ont démontré que leurs construction est compacte (Corollaire 2.1).

Corollaire 2.1 [Bellaouar et al., 2018] Soit $s = s_1 s_2 \dots s_n$ un mot sur Σ^*. L'automate GASWA $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ représentant toutes les sous-séquence du mot s avec trous est compact.

Le résultat combiné de Lemme 2.1 et du Corollaire 2.1 mènent au Théorème 2.1.

Théorème 2.1 [Bellaouar et al., 2018] Soit $s = s_1 s_2 \dots s_n$ un mot sur Σ^*. L'automate GASWA $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ réalise la série formelle associée à l'espace de redescription de la projection correspondante au noyau toutes sous-séquences avec trous donnée par :

$$\Phi : s \mapsto \Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)}, w \in \Sigma^*.$$

Et la construction GASWA est compacte.

2.3 Calcul du noyau de séquences à base d'automates pondérés

nous pouvons calculer le noyau toutes sous-séquences avec trous entre deux mots s et t ($k(s,k)$) à base d'automates pondérés en deux étapes. Dans la première étape, nous devons construire l'automate $A_{s,t} = A_s \cap A_t$ résultant de l'intersection de deux automates pondérés $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ et $A_t = (\Sigma, Q_t, E_t, I_t, F_t, \lambda, \rho)$ qui réalisent respectivement toutes les sous-séquences des mots s et t. Ensuit, dans la deuxième étape, nous évaluons tous les calculs de

$A_{s,t}$ (chemins réussis de toutes les sous-séquences communes entre les mots s et t) [Bellaouar et al., 2017].

2.3.1 Intersection des automates pondérés

L'intersection des automates pondérés peut être considérée comme une généralisation de l'intersection des automates finis [Hopcroft and Ullman, 1979]. Aussi, elle peut être vue comme un cas particulier de la composition des transducteurs pondérés [Pereira and Riley, 1997, Cortes et al., 2004, Mohri et al., 1996]. L'automate pondéré $A_{s,t} = A_s \cap A_t$ qui réalise l'intersection des deux automates pondérés $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ et $A_t = (\Sigma, Q_t, E_t, I_t, F_t, \lambda, \rho)$ est défini par les composantes suivantes :

- Les états de l'automate $A_{s,t}$ sont les paires des états à partir de l'automate A_s et les états à partir de l'automate A_t.
- Une transition $e_{s,t} = ((q_s, q_t), l(e_{s,t}), wt(e_{s,t}), (q'_s, q'_t))$ de l'automate $A_{s,t}$ est conçue à partir d'une transition $e_s = (q_s, l(e_s), wt(e_s), q'_s)$ de l'automate A_s et une transition $e_t = (q_t, l(e_t), wt(e_t), q'_t)$ de l'automate A_t, où $l(e_{s,t}) = l(e_s) = l(e_t)$ et $wt(e_{s,t}) = wt(e_s) \cdot wt(e_t)$.
- Les états initiaux de $A_{s,t}$ sont les paires des états initiaux de A_s et de A_t.
- Les états finaux de $A_{s,t}$ sont les paires des états finaux de A_s et de A_t.
- Les fonctions de pondération initiale λ et finale ρ sont des scalaires égaux à 1.

La complexité spatiale et temporelles de l'intersection dans le pire des cas (toutes les transitions partant de l'état q_s correspondent à toutes les transitions partant de l'etat q_t) est $O(|A_s||A_t|)$.
Cependant, un problème se pose lors de l'application de cette intersection dans le cas des ϵ-transitions. Ceci peut générer des ϵ-chemins redondants. Ainsi, conduisent à un résultat incorrect. Pour faire face à ce problème, [Mohri et al., 1996, Pereira and Riley, 1997] ont étendu leur algorithme par le mécanisme de filtrage-epsilon portant sur les transducteurs pondérés. Bellaouar et al. [Bellaouar et al., 2017] ont adaptés la solution de filtrage-epsilon pour les automates pondérés.
Afin d'illustrer le problème cité en haut et la solution adaptée, nous discutons l'exemple d'intersection de deux automates A_s et A_t associés respectivement aux mots $s =$ "xyz" et $t =$ "xzy".
La Figures 2.4 montre Les deux automates GASWA associés aux mot s et t.

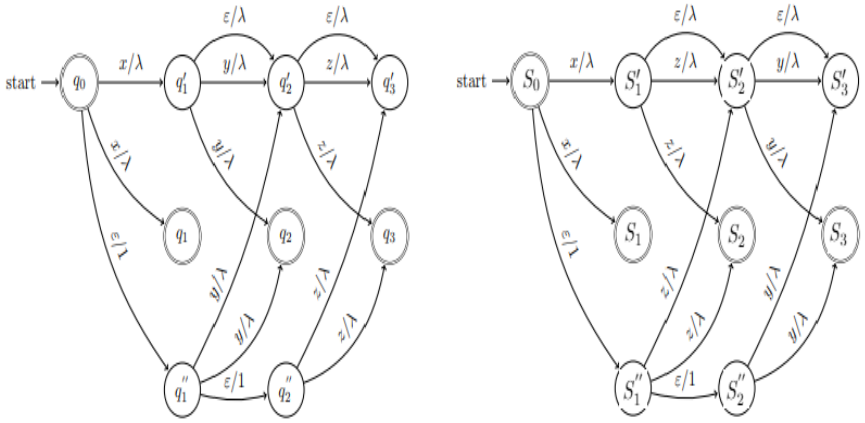

FIGURE 2.4 – Les deux automates A_s et A_s associés respectivement aux mots $s =$ "acb" et $t =$ "abc".

Pour comprendre comment ces -transitions fonctionnent, nous étendons A_s et A_t pour obtenir respectivement A'_s et A'_t comme suit : les ϵ-transitions dans A'_s (A'_t) sont étiquetés $\epsilon_1(\epsilon_2)$ et les ϵ-transitions correspondant dans $A'_s(A'_t)$ sont respectivement marquées par des boucles étiquetées ϵ_2 (ϵ_1), voir la Figure 2.5.

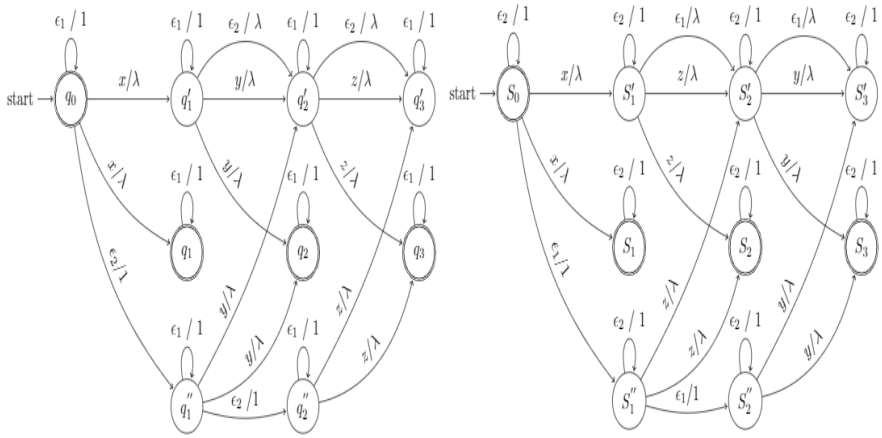

FIGURE 2.5 – Les deux automates A'_s et A'_t associés respectivement aux mots $s =$ "xyz" et $t =$ "xzy".

Ainsi, nous obtenons deux types de ϵ-transitions dans les automates : (wait, move) et (move, wait) (La Figure 2.5). La Figure 2.6 illustrer les transitions possibles résultant de l'intersection des deux automates A'_s et A'_t.

Nous pouvons observer les chemins redondants, qui conduit à un résultat incorrect pour le cas pondéré. Pour être correct, il faut garder un seul chemin. Donc pour résoudre ce problème, on utilise le mécanisme de filtrage-epsilon qui

peut être représenté par un automate pondéré (Figure 2.7) [Bellaouar et al., 2017]. Plus de détails pour la preuve et des discussions intuitives pour le filtrage epsilons peuvent se trouver dans [Mohri et al., 1996, Pereira and Riley, 1997].

2.3.2 Évaluation du noyau

Pour évaluer le noyau toutes sous-séquences avec trous entre deux mots s et t, il faut passer par les étapes suivantes :
Premièrement, nous devons construire l'automate pondéré $A_{s,t} = A_s \cap A_t \cap A_f$ sur un semi-anneau $\mathbb{S}$, où A_s et A_t représentent respectivement l'automate du mot s, l'automate du mot t et l'automate filtre (Figure 2.7) pour éliminer les ϵ-chemins redondants.
Deuxièmet, nous devons calculer la somme des poids de tous les calculs de $A_{s,t}$. Sous forme formelle, nous devons calculer la distance comme suit :

$$dist(q_0) = \sum_{\pi \in P(q_0,F)} \lambda(o(\pi)) \cdot wt(\pi) \cdot \rho(d(\pi)). \tag{2.7}$$

Où $P(q_0, F)$ dénote l'ensemble des chemins à partir de l'état initiale q_0 jusqu'aux tous les états finaux F.
En effet, est un algorithme de plus court chemin généralisé [Mohri, 2002]. La complexité de l'évaluation du noyaux est $o(|A_{s,t}|)$, du moment que l'automate $A_{s,t}$ est acyclique.

2.4 Relations avec d'autres noyaux de séquences

Dans cette section, nous nous concentrons sur la relation entre certains noyaux de séquences et le noyau de séquence à base d'automates pondérés pour illustrer l'aspect unification du modèle GASWA.

2.4.1 Noyau toutes sous-séquences

Nous avons décrit le noyau toutes sous-séquences dans la section 1.3.3.
Il a été prouvé qu'il peut être défini comme un noyau de sous-séquences à base d'automates pondérés par la Proposition 2.2.

Proposition 2.2 [Bellaouar et al., 2018] Soient s et t deux mots dans Σ^*, le noyau toutes sous-séquences $K(s,t)$ peut être défini comme un noyau de sous-séquences à base d'automates pondérés $dist(A_{s,t})$, $A_{s,t} = A_s \cap A_t$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t.

2.4.2 Noyau sous-séquences de mots

Le noyau sous-séquences de mots [Lodhi et al., 2002] est décrit à la section 1.3.5.
La preuve de Proposition 2.3 montre que nous trouvons une relation entre le Noyau sous-séquences et un noyau de sous-séquences à base d'automates GASWA.

Proposition 2.3 [Bellaouar et al., 2018] Soient s et t deux mots dans Σ^* et p la longueur des sous-séquences. Le noyau sous-séquences de mots $k_p(s,t)$ peut être défini comme un noyau de sous-séquences à base d'automates GASWA $dist(A)$, $A = A_s \cap A_t \cap A_{pgram}$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t et A_{pgram} est l'automate pondéré correspondant à tous les p-grams sur Σ^*.

À titre d'exemple, la Figure 2.8 illustre un automate 2-gram.

Noyau sous-séquences de longueur fixe

Le noyau sous-séquences de longueur fixe est décrit à la section 1.3.4 .
La relation entre le noyau sous-séquences de longueur fixe et un noyau de sous-séquences à base d'automates GASWA est prouvée dans la Proposition 2.4 .

Proposition 2.1 [Bellaouar et al., 2018] Soient s et t deux mots dans Σ^* et p la longueur des sous-séquences. Le noyau sous-séquences de longueur fixe $K_p(s,t)$ peut être défini comme un noyau de sous-séquences (sans pénalisation) à base GASWA $dist(A)$, $A = A_s \cap A_t \cap A_{pgram}$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t et A_{pgram} est l'automate pondéré correspondant à tous les p-grams sur Σ^*.

2.4.3 Autre noyaux de séquences

Grâce au modèle GASWA, il est possible de calculer d'autres types de noyaux de séquences. Ceci est réalisé en modifiant le paramètre de pénalité λ, ou en utilisant d'autres automates comme l'automate des p-gram pour spécifier le calcul du noyau toutes sous-séquences avec trous [Bellaouar, 2018].
En outre, nous pouvons calculer un nouveau noyau d'un motif spécifique par la conception d'un automate spécifique $A_{pattern}$ sur un alphabet Σ). En suit, nous réalisons l'intersection $A_s \cap A_t \cap A_{pattern}$.

De plus, cette approche est valide même si on s'intéresse aux sous-mots, Il suffit de construire un automate pondéré pour tous les sous-mots ayant la même construction que celui de l'automate toutes sous-séquences en tenant de compte l'élimination les ϵ-transitions $e_i = (q_{i-1}, \epsilon, \lambda, q_i)$ $i = 1..n-1$ à partir du premier niveau. La Figure 2.9 illustre cette idée :

2.5 Généralisation pour un noyau d'ensembles de Séquences

Nous avons vu dans la section 2.2, que le point important de la plate-forme générale qui proposée par [Bellaouar et al., 2017, Bellaouar, 2018] est la représentation de toutes les sous-séquence d'un mot s par un l'automate pondéré compacte (GASWA).
Dans la présente section, nous étudions la généralisons de cette idée pour un ensemble de mots afin de construire un nouveau noyau appelé noyau d'ensembles de séquences. L'idée principale pour cette généralisation consiste à construire un automate pondéré préfixe (APP), qui représente toutes les sous-séquences d'un ensemble de mots D.
Pour illustrer ce processus de construction, nous considérons l'ensemble de mots $D = \{xyy, xyz, yzv, yzw\}$

— pour crée le premiér niveau de l'automnate APP constitué des états $q'_0, \ldots, q'_n$ ($q'_0 = q_0$), nous construirons un APP compacte representant tous les mots de l'ensemble D. où chaque mot dans D est representé par un chemin dans cet APP. Dans le pire des cas , $n = \sum_{s \in D} |s|$ (n 'existe aucune deux mots dans D ayant le même préfixe). Les transitions de ce niveau sont pondérée

par le facteur de penalisation λ, qui pénalise la distance entre deux symboles consécutifs dans le mots s. Ensuite, pour autoriser les trous, nous devons étendre le premiér niveau par des ϵ-transitions, avec la pénalité de chaque ϵ-transitions par le fecteur λ pour chaque trou, En plus , Nous ajoutons à chaque état le nombre de passes dans cet état. Cette valeur represente le nombre d'occurrences du symbole dans la transition incidente. Dans la Figure 2.10, nous trouvons une illustration de cette construction (le premiér niveau de l'automate APP).

— Ensuite, pour générer les sous-séquences des mots de l'ensemble D, nous devons passer à partir du premier niveau au second qui comporte les états finaux $q_0, \ldots, q_n$ de l'APP en rajoutant des transitions $e_i = (q_i, l(e_i), \lambda, q'_{i+1}),\ i = 0..n-1$.Pour prendre en compte la multiplicité des sous-séquences, nous devons copier la valeur stockée dans chaque état du premier niveau $q'_i,\ i = 1..n$, dans l'état final q_i du deuxième niveau comme des poids finaux. La Figure 2.11 illustre cette étape.

— Finalement, nous créons les états du troisième niveau $q''_0, \ldots, q''_k$, où k est la longueur des plus long mots dans l'ensemble D moins un. Pour commencer par n'importe quel symbole du mot s dans l'ensemble D, nous devons construire les transitions de troisième niveau : $e_i = (q''_i, \epsilon, 1, q''_{i+1}),\ i = 0..k-1$. Puis, de l'état q"$_i, pour$1=$1..k$, nous pouvons décider soit terminer une sous-séquence avec la transition $e_i = (q''_i, l(e_i), \lambda, q_j)$, où $l(e_i)$ est une étiquette de la position $i+1$ dans un mot s de l'ensemble D et les états q_j sont les états finaux correspondants de l'APP. Dans notre exemple, nous pouvons créer quatre transitions à partir de q''_2 aux états finaux pour terminer une sous-séquence, (q''_2, y, λ, q_3), (q''_2, z, λ, q_4), (q''_2, v, λ, q_7) et (q''_2, w, λ, q_8). Pour continuer les sous-séquences à partir des états q'_j avec les transitions $e_i = (q''_i, l(e_i), \lambda, q'_j)$, où $l(e_i)$ est l'étiquette de la la position $i+1$ dans un mot s de l'ensemble D et les états q'_j sont les états associés du premier niveau de l'APP. Dans notre exemple, nous pouvons créer quatre transitions à partir de l'état q''_2 pour continuer les sous-séquences, $(q''_2, y, \lambda, q'_3)$, $(q''_2, z, \lambda, q'_4)$, $(q''_2, v, \lambda, q'_7)$ et $(q''_2, w, \lambda, q'_8)$. Le processus complet est illustré par la Figure 2.12.

Il est clair de voir que le processus de construction de l'APP est similaire à celui de construction de l'automate GASWA. Par consiquent, l'évaluation d'un

noyau pour un ensemble de mots D_1 et D_2 reste identique à celui d'évaluation d'un noyau de séquence pour deux mots.
Par ailleurs, nous pouvons considérer ce nouveau noyau comme un noyau d'arbres, où les sous structures sont des sous-chemins de l'automate pondéré préfixe.
L'utilisation de ce nouveau noyau pourrait être très utile dans de nombreuses applications de l'apprentissage automatique, particulièrement quand nous devons comparer deux ensembles d'objets.

2.6 Conclusion

Dans ce chapitre, nous avons commencé par l'introduction des informations préliminaires concernant les automates pondérés et les séries formelles qui sont la base du modèle d'unification des noyaux de séquences. Ensuite, nous avons vu une présentation du développement d'une plate-forme générale pour le calcul des noyaux de séquences efficacement. Où nous avons introduit l'idée générale et le processus de construction de modèle GASWA [Bellaouar, 2018], qui est valide, et efficace.
Par la suite nous avons montré l'aspect unification de le plate-forme proposée.
Finalement, nous avons étudie l'aspect généralisation du modèle de mots à un modèle sur des ensembles de mots donnait naissance à un nouveau noyau d'ensemble de mots.

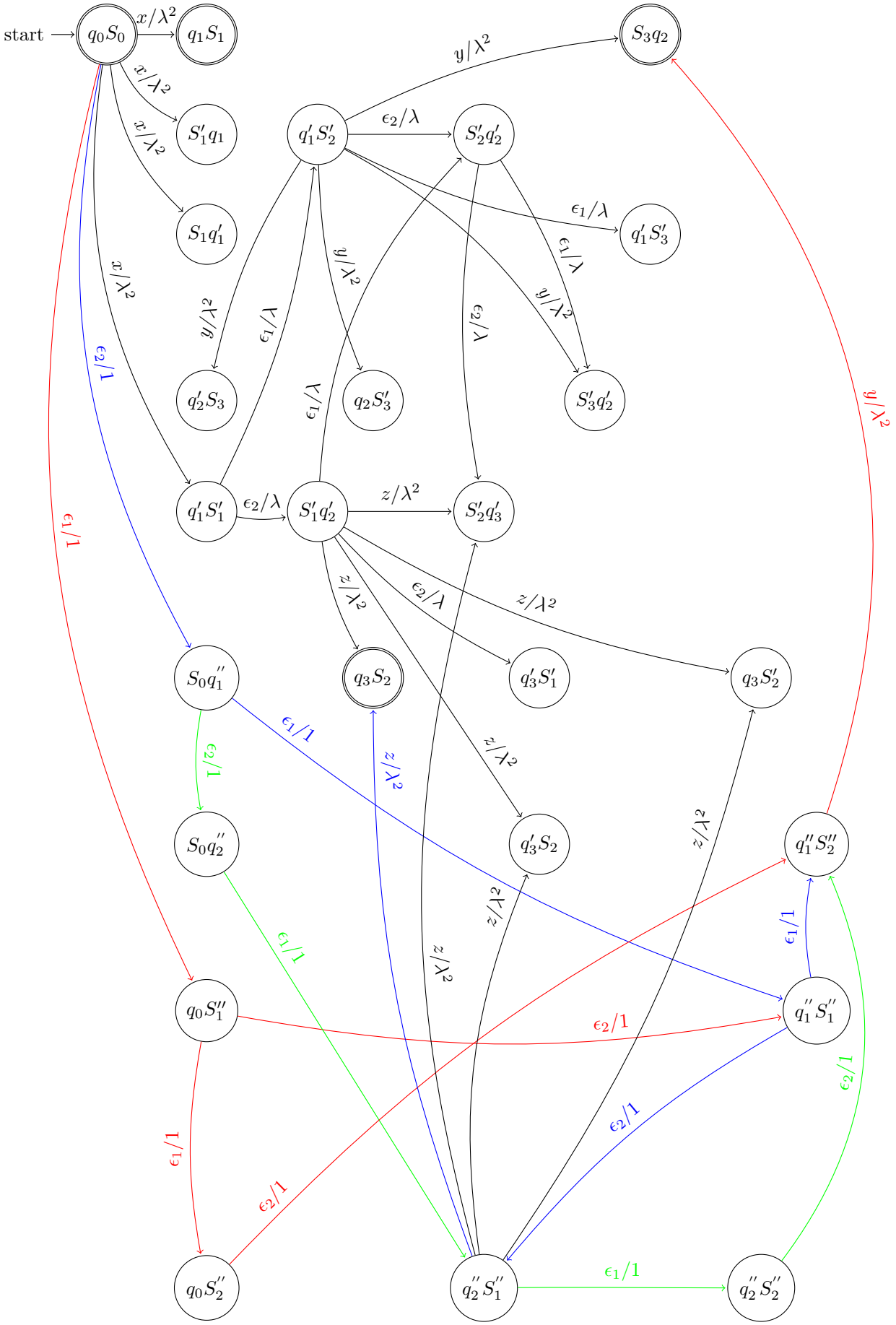

FIGURE 2.6 – Automate pondéré $A_{s,t} = A_s \cap A_t$

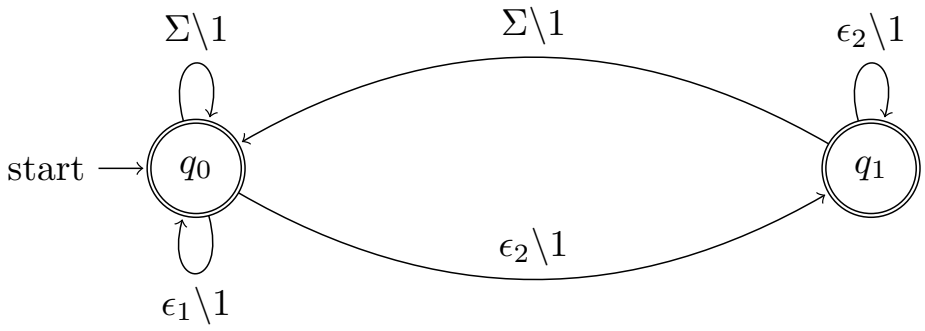

FIGURE 2.7 – Automate filtre pour éliminer les ϵ-chemins. de [Bellaouar et al., 2017]

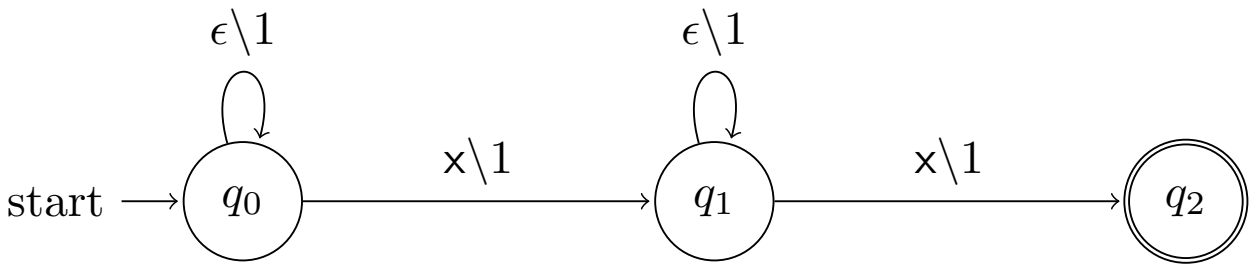

FIGURE 2.8 – Automate 2-*gram* (A_{pgram}), x représente un élément de Σ. Extrait [Bellaouar, 2018]

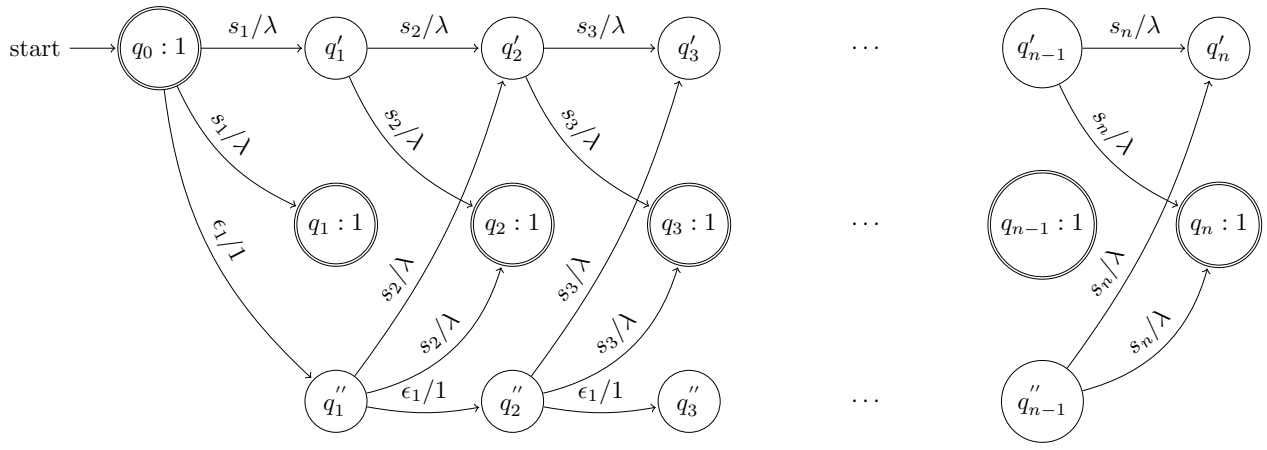

FIGURE 2.9 – Automate pondéré pour tous les sous-mots d'un mot $s = s_1 s_2 \dots s_n$.

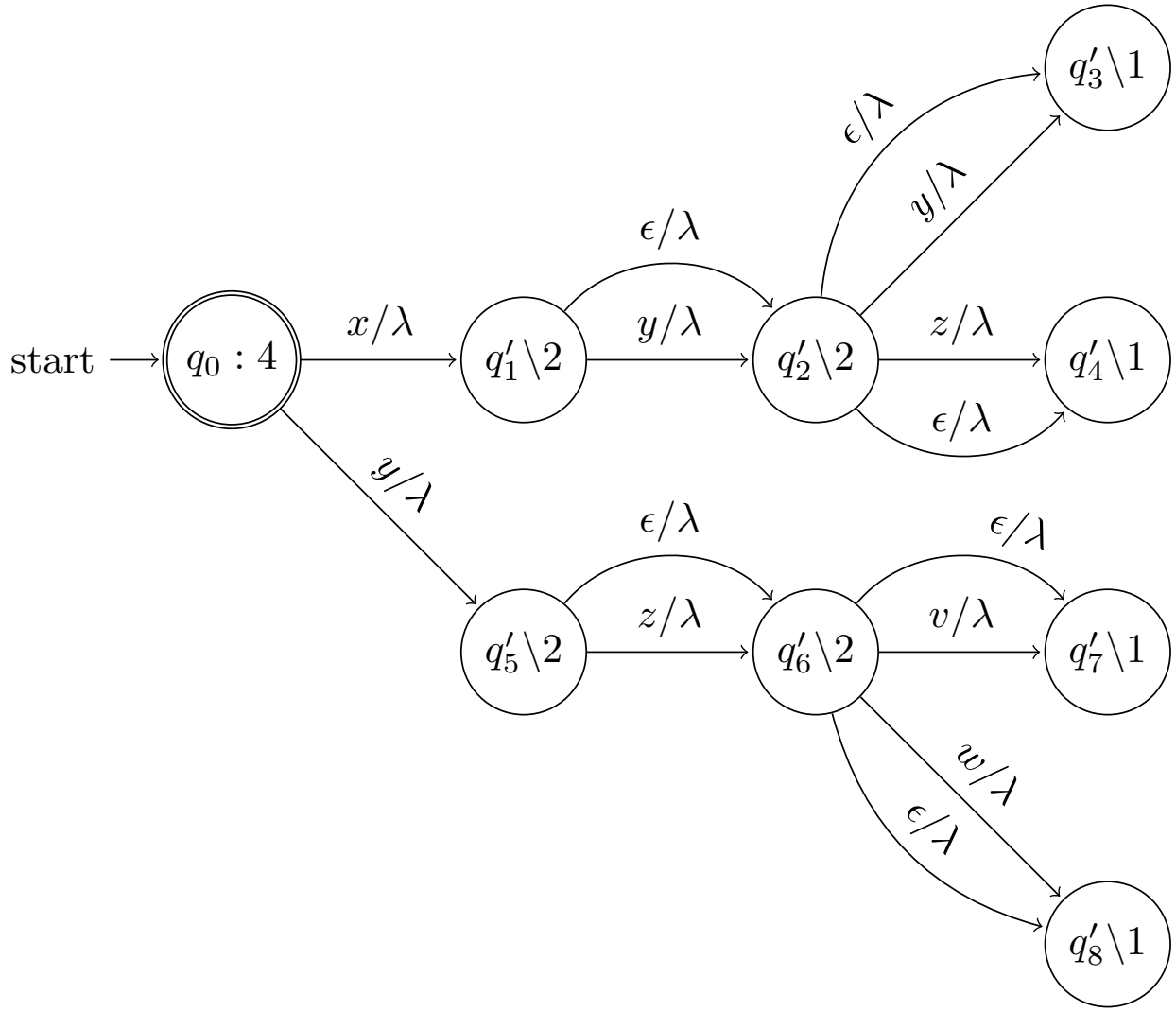

FIGURE 2.10 – Automate pondéré préfixe représentant l'ensemble de mots $D = \{xyy, xyz, yzv, yzw\}$.

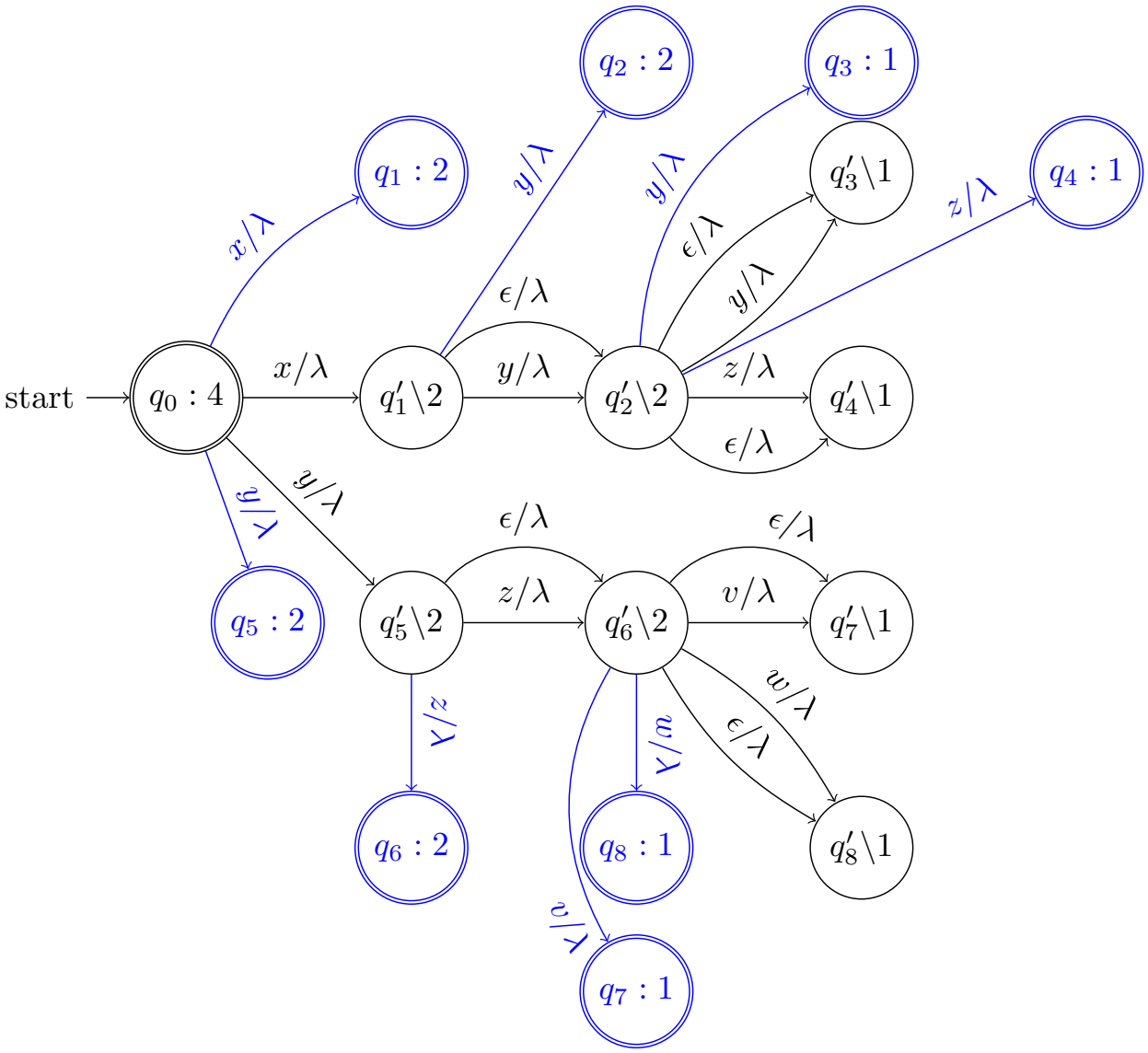

FIGURE 2.11 – Le APP renforcé par les états du deuxième niveau.

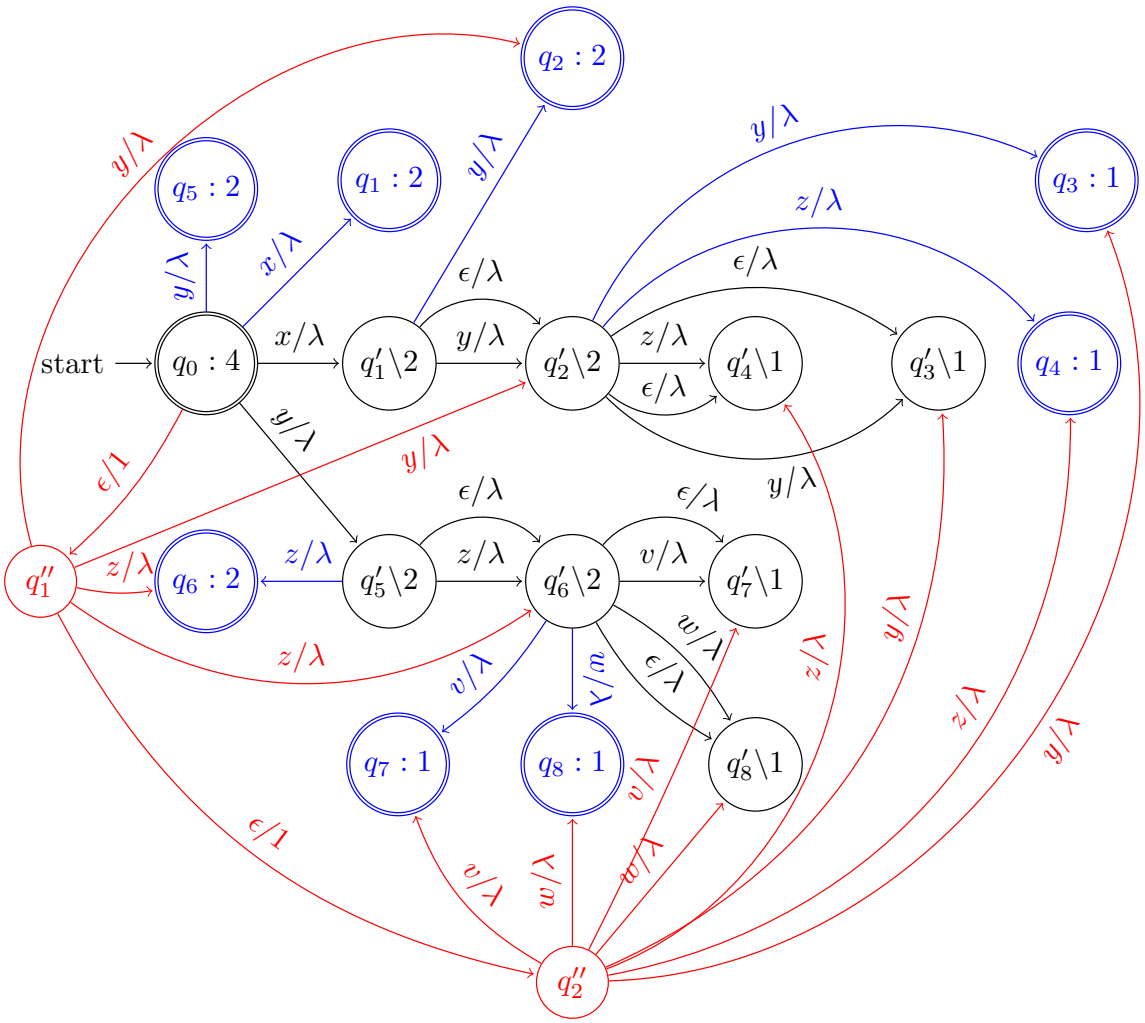

FIGURE 2.12 – Le APP final représentant toutes les sous-séquences des mots de l'ensemble $D = \{xyy, xyz, yzv, yzw\}$.

Chapitre 3

Implémentation des noyaux de séquences

Dans ce chapitre nous nous focalisons sur l'implémentation des noyaux de séquences en utilisant notre plat-forme générale étendu pour un ensemble de séquences. Autrement dit, l'implémentation dans un premier temps de l'automate pondéré préfixe représentant toutes les sous-séquences d'un ensemble de séquences. Ensuite le calcul de noyau se manifeste par l'implémentation de l'intersection de deux automates pondérés préfixe associés à deux ensembles de séquences. Pour se faire nous avons utilisé le langage de programmation java.

3.1 Environnement de programmation

Notre implémentation est réalisée sur un PC Intel Core i3 avec un processeur de 1.7 GHz et 4 GB de mémoire centrale fonctionnant sous le système d'exploitation Windows 8.1. Toutes les classes sont développées avec le langage Java en utilisant l'environnement de développement elipse 2017.

3.2 Représentation des automates

Dans littérature, il existe plusieurs représentations des automates. Dans cette section, nous nous focalisons sur les représentations les plus répandues, à savoir, matrice d'adjacence et liste d'adjacence.
D'abord un automate en générale et un automate pondéré en particulier, peut être considéré comme un graphe orienté (pondéré) qui peut être défini comme suit :
Définition 3.1 [Robert and Kevin, 2011] Un graphe orienté (ou digraphe) est un ensemble de sommets et une collection d'arcs (d'arêtes dirigées). Chaque arc connecte une paire ordonnée de sommets.

Définition 3.2 [Robert and Kevin, 2011] Le graphe pondéré est un modèle de graphe dans lequel nous associons des poids ou des coûts à chaque arc.

Dans la théorie des automates, les sommets représentent les états et les arcs représentent les transitions.

3.2.1 Matrice d'adjacence

La matrice d'adjacence est un tableau de deux dimensions de taille $V \times V$ où V est le nombre des états de l'automate. Soit $adj[][]$ un tableau de deux dimensions, $adj[i][j] = (a, \lambda)$, indique qu'il existe une transition de l'état i vers l'état j portant une étiquette a et une pondération λ.
À titre d'illustration, nous essayons de représenter l'automate pondéré de la Figure 3.1 par la matrice d'adjacence de la Table 3.1.
La pondération 0 indique l'absence d'une transition.

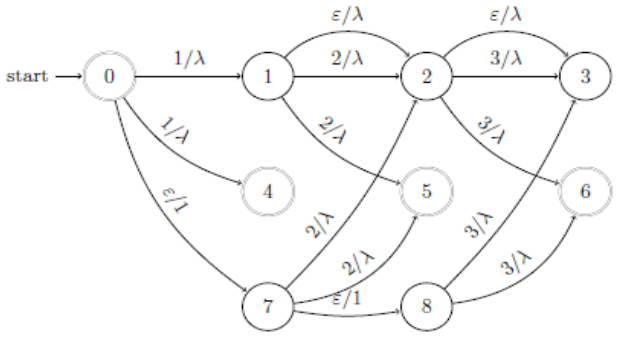

FIGURE 3.1 – Automate pondéré correspondant à la séquence "123".

TABLE 3.1 – Matrice d'adjacence représentent l'automate de la Figure 3.1.

	0	1	2	3	4	5	6	7	8
0	0	(1,λ)	0	0	(1,λ)	0	0	(ϵ, 1)	0
1	0	0	(2,λ), (ϵ, λ)	0	0	(2,λ)	0	0	0
2	0	0	0	(3,λ), (ϵ, λ)	0	0	(3,λ)	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	(2,λ)	0	0	(2,λ)	0	0	(ϵ, 1)
8	0	0	0	(3,λ)	0	0	(3,λ)	0	0

Cette représentation est facile à implémenter. Elle est, aussi, rapide dans la recherche et dans la vérification de la présence ou l'absence d'une transition entre deux états. La matrice d'adjacence fournit un accès à temps constant $O(1)$ pour un couple d'états (i, j).

Parmi les inconvénients de cette structure :

— Complexité $O(V^2)$ en matière d'espace mémoire. Ceci même si le nombre des transitions est petit. L'espace mémoire est lié au nombre des états et non pas au nombre de transitions.

— L'ajout d'un seul état nécessite $O(V^2)$ opérations.

3.2.2 Liste d'adjacence

La structure de donnée liste d'adjacence utilise un tableau de listes ou un tableau de tableau dynamique. La taille du tableau est égale au nombre des états.
Si on dispose d'un tableau $adj[]$, un tableau d'entrée $adj[i]$ représente la liste des états adjacents à l'état i. La figure 3.2 montre la structure de données liste d'adjacence associée à l'exemple de Figure 3.1.

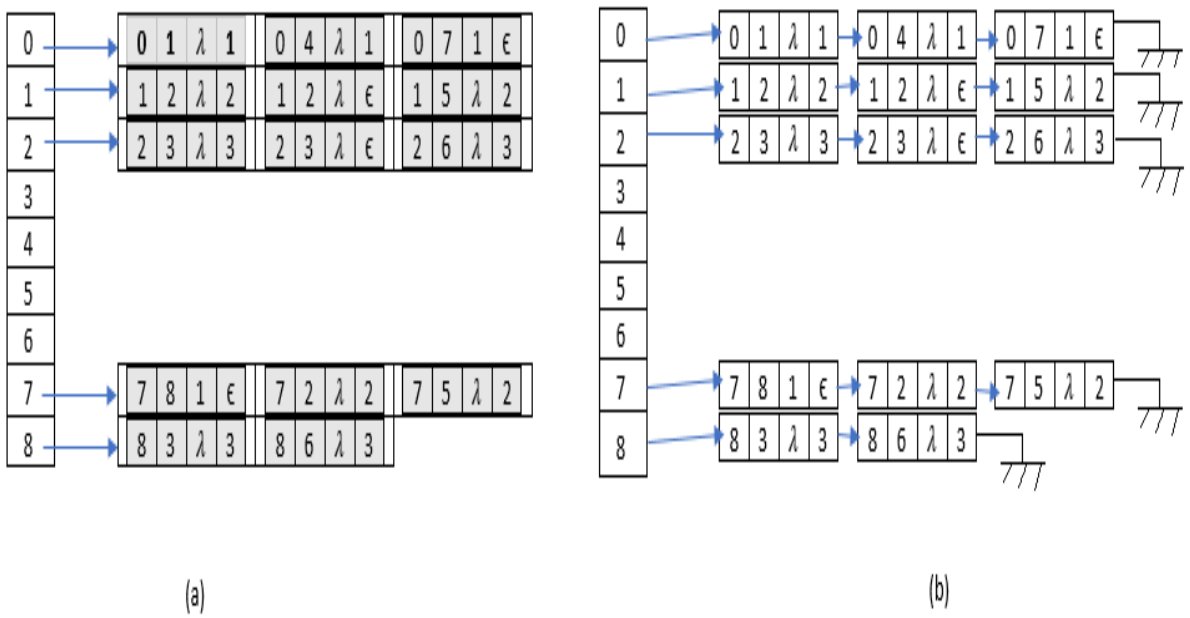

FIGURE 3.2 – Liste d'adjacent de l'automate de la Figure 3.1 par la structure tableau de tableau dynamique (a) et la structure tableau de liste (b).

Dans cette structure, la taille de la mémoire réservée est liée au nombre des états et au nombre des transitions. Ainsi, la complexité de l'espace mémoire est $O(V + E)$, tel que E est le nombre des transitions.

De plus, l'ajout d'un état dans un automate représenté à l'aide d'une liste d'adjacence peut être effectué en $O(1)$.
Dans la Table 3.2, nous comparons la complexité de l'implémentation liste adjacence matrice adjacence.

TABLE 3.2 – Comparaison entre liste adjacence et matrice adjacence

Implémentation	Complexité temporelle		Espace mémoire
	Inserstion	Recherche	
Liste d'adjacence	$O(1)$	$O(K)$	$O(V+E)$
Matrice d'adjacence	$O(V_2)$	$O(1)$	$O(V_2)$

K est la taille de voisinage d'un noeud.

3.3 Choix de la structure de données

Le choix de la structure de données appropriée est lié au nombre des états et au nombre des transitions.
Le nombre des états et le nombre des transitions d'un automate pondéré, dans notre cas, varie en fonction du nombre de mots ayant un préfixe commun. Autrement dit, le nombre des états et de transitions diminue dans le cas où le nombre de mots avec un préfixe commun augmente. Dans le cas contraire, la taille de l'automate augmente. Par ailleurs, et suite à la construction compacte de l'automate pondéré préfixe le nombre de transition est toujours très inférieure au nombre des états.
Il n'est pas difficile de conclure que la représentation basée sur une matrice d'adjacence de l'automate pondéré préfixe mène à une matrice creuse . Une alternative consiste à utiliser une représentation à base de liste d'adjacence.
Du point de vue implémentation de la liste d'adjacence, il est clair que notre structure de donnée doit être purement dynamique, du moment qu'elle est utilisée pour représenter un ensemble de mots.
Nous discutons, par la suite le choix entre une liste linéaire chainée et un tableau dynamique (de taille modifiable).

3.3.1 Complexité d'utilisation la structure liste linéaire chaînée

Le meilleur cas pour ajouter un élément dans un liste linéaire chaînée est de l'ajouter en tête de liste. Le coût de cette opération est $O(1)$.

Dans le pire de cas, la recherche d'un élément dans un liste linéaire chaînée est $O(N)$, où N est la taille de la liste.

3.3.2 Complexité d'utilisation la structure tableau dynamique

La recherche d'un élément dans la table dynamique de taille N nécessite dans le pire de cas $O(N)$ opération.
La structure tableau dynamique nous assure que la modification de la taille se produira rarement. Ceci est réalisé en doublant la taille du tableau (repeated doubling) à chaque fois que le tableau est plein.
Nous calculons le coût des opérations de copie de données lors de l'ajout d'un nouvel élément dans le tableau dynamique à travers de cette illustration :
Au départ de l'insertion, la table est vide. La $i^{éme}$ opération déclenche une expansion uniquement lorsque $i-1$ est une puissance de $2 \cdot 2^x \rightarrow 2^{x+1}$ (le coût d'expansion est 2^x). Par conséquent, le coût total de n opérations d'insertion est :

$$C_i = n + \sum_{i=1}^{|log_2 n|} 2^i < 3n = O(n).$$

Donc, le coût amorti $= (O(n))/n = O(1)$. Par conséquent, le coût de chaque insertion dans la table dynamique est amorti à $O(1)$.
La Table 3.3 résume la comparaison entre une liste linéaire chaînée et un tableau dynamique en terme de complexité temporelle

TABLE 3.3 – Comparaison entre un liste linéaire chaînée et un tableau dynamique en terme de complexité temporelle.

Implémentation	Complexité temporelle			
	Pire de cas		Meilleure de cas	
	Inserstion	Recherche	Inserstion	Recherche
LLC	$O(N)$	$O(1)$	$O(1)$	$O(1)$
Tableau dynamique	$O(N)$	$O(N)$	$O(N)$	$O(1)$

De plus, du point de vue espace mémoire, il faut considérer l'espace supplémentaire utilisé par un liste linéaire chaînée pour gérer les liens.
De ce fait, nous optons pour un tableau dynamique de tableau dynamique pour implémenter notre liste d'adjacence représentant l'automate pondéré préfixe.

3.4 Implémentation de l'automate pondéré préfixe

Pour implémenter l'automate pondéré préfixe nous avons développé quatre classes, à savoir, la classe Transition, Automate, état et la classe noyau.

3.4.1 Classe Transition

Une transition est représentée par une classe qui contient l'état source, l'état destination, l'étiquette et la pondération.
Cette transition fait partie d'un ensemble de transitions incidentes à un état. Dans notre cas, cet ensemble est représenté par ArrayList.
L'API de la classe Transition se montre dans la Table 3.4.

TABLE 3.4 – API pour la classe Transition.

public class transition		
	Transition (int s, int d, double l, char e)	crée un Transition entre l'état source s et l'état d
String	ToString ()	afficher une Transition
int	getsrc ()	retourne la source d'une transition
int	getdes ()	retourne la destination d'une transition
char	getetiq ()	retourne l'étiquette d'une transition
double	getpoid ()	retourne la pondération d'une transition
int	setsrc ()	mèttre à jour l'état source d'une transition
int	setdes ()	mèttre à jour l'état destination d'une transition
char	setetiq ()	mèttre à jour l'étiquette d'une transition
double	setpoid ()	mèttre à jour le pondération d'une transition

3.4.2 Classe Automate

L'automate pondéré préfixe est représenté par une liste d'adjacence. Plus précisment, dans notre cas nous avons choisi un tableau dynamique de tableau dynamique.
Ainsi, la classe Automate contient un tableau dynamique pour représenter les états.
Chaque entrée de ce tableau contient un tableau dynamique pour représenter les transitions incidentes à un tel état.

D'autre informations supplémentaire concernant les états, tels que le statut finale, non final et le nombre de passes par chaque état sont emmagasinées dans des tableaux dynamiques séparés.
Dans cette classe il existe les méthodes principales pour construire un automate pondéré (la Table 3.5).

TABLE 3.5 – API pour la classe Automate.

public class Automate		
	Automate ()	construire un automate filtre
	Automate (int p)	construire un automate p-grame
	Automate (Sting s, double lambda)	construire un automate GASWA pour le mot s avec facteur de pénalité lambda
	Automate (String [] s, double lambda)	construire un automate pondéré préfixe pour un ensemble des mots s[] avec facteur de pénalité lambda
void	Addstate (int s)	rajouter un état s à l'automate
void	AddTransition (int s, Transition t)	rajouter une transition t à l'état s
Automate	Automate_Etendu_APP (Automate A)	exentension de l'automate A par des epsilon-transitions
booléen	estFinale (int s)	test l'état s est finale ou non
int	get_state_size ()	retourne nombre de state (de l'automate)
int	get_transition_size ()	retourne le nombre de transitions incidentes dans un état

3.4.3 Classe état

La classe état est une classe interne de la classe Automate. l'API de la classe état illustrée dans La Table 3.6.

3.4.4 Classe Noyau

La classe noyau est développée pour calculer un noyau de séquence d'un ensemble de mots. l'API de la classe est représentée dans la Table 3.7.

Le constructeur de la classe noyau est un automate intersection entre trois automates.
Nous avons choisi trois paramètre pour la classe noyau pour être le plus générale

TABLE 3.6 – API pour la classe état.

Public class état		
	état (int s, boolean Final)	crée un état s qui a une statut Final
	état (int s)	crée un état s
void	set_statut (int s, boolean Final)	mèttre à jour la statut de l'état s
boolean	get_statut (int s)	returne la statut de l'état s
void	set_pass (int s, int pass)	mèttre à jour la valeur de pass d'un état s
int	get_pass (int s)	returne la valeur de pass associer à l'état s

TABLE 3.7 – API pour la classe noyau.

public class noyau		
public Automate	noyau (Automate A ,Automate B, Automate P)	Créée un noyau (Automate Intersection) entre trois Automate.
int	calculeNoy (Automate A)	évaluation du noyau d'un automate A

possible.
En générale, nous aurons besoin de deux automates pondéré préfixes représentant deux ensembles de mots, et éventuellement, nous aurons, aussi besoin d'un automate filtre ou d'un automate p-gram.
L'évaluation d'un noyau fait appel à un parcours DFS avec l'application d'un algorithme du plus court chemin généralisé.

3.5 Conclusion

Ce chapitre est dédié à l'implémentation des noyaux des séquences.
Dans premier temps, nous avons discuté les différentes représentations possibles d'un automate pondéré préfixe. Nous avons choisi la représentation sous forme d'une liste d'adjacence.
Par la suite , nous avons décrit les différentes classes Java utilises par implémentation les noyaux de séquences, à savoir, les classes état, Transition, Automate, et noyau.

Conclusion générale

Dans la littérature, il existe une diversit de noyaux de séquences marquant l'intérêt croissant occtroyé par les chercheurs à ce sujet de recherche.
Le but du présent memoire est quadruple :

- Investigation de la famille de noyaux de séquences.
- Étude de l'aspect unification des noyaux de séquences qui se repose sur la relation entre les concepts, noyau, série formelle et automate pondéré.
- Étude de l'aspect unification des noyaux de séquences qui se repose sur la représentation d'un automate pondéré préfixe.
- Implémentation du modèle généralisée qui utilise une structure de données liste d'adjacence et le paradigmes orienté objet du langage Java.

Au terme de ce mémoire, et vue la limite du temps, notre implémentation requiert encore plus de raffinements afin que le modéle généralisé pourra être utilisé dans des tâches réels d'apprentissage automatique.

Bibliographie

[Alex and Vishwanathan, 2008] Alex, S. and Vishwanathan, S. (2008). *Introduction to Machine Learning*. United Kingdom.

[Althubaity et al., 2008] Althubaity, A., Almuhareb, A., Alharbi, S., Al-Rajeh, A., and Khorsheed, M. (2008). KACST Arabic Text Classification Project : Overview and Preliminary Results. In *Proceedings of The 9th IBIMA conference on Information Management in Modern Organizations*.

[Bellaouar, 2018] Bellaouar, S. (2018). *Noyaux de mots et d'arbres : efficacité et unification*. PhD thesis, these de doctorat. University of Laghouat, Algeria.

[Bellaouar et al., 2017] Bellaouar, S., Cherroun, H., Nehar, A., and Ziadi, D. (2017). Weighted automata sequence kernel. In *Proceedings of the 9th International Conference on Machine Learning and Computing*. ACM.

[Bellaouar et al., 2014] Bellaouar, S., Cherroun, H., and Ziadi, D. (2014). Efficient list-based computation of the string subsequence kernel. In *LATA'14*, pages 138–148.

[Bellaouar et al., 2018] Bellaouar, S., Cherroun, H., and Ziadi, D. (2018). Efficient geometric-based computation of the string subsequence kernel. *Data Mining and Knowledge Discovery*, 32(2) :532–559.

[Cancedda et al., 2003] Cancedda, N., Gaussier, E., Goutte, C., and Renders, J. M. (2003). Word sequence kernels. *J. Mach. Learn. Res.*, 3 :1059–1082.

[Cortes et al., 2004] Cortes, C., Haffner, P., Mohri, M., Bennett, K., and Cesabianchi, N. (2004). Rational kernels : Theory and algorithms. *Journal of Machine Learning Research*, 5 :1035–1062.

[Cristianini and Shawe-Taylor, 2000] Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support Vector Machines : and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA.

[Hoare, 1996] Hoare, T. (1996). Unification of theories : A challenge for computing science. In *Recent Trends in Data Type Specification*, pages 49–57. Springer.

[Hopcroft and Ullman, 1979] Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

[Joachims, 1998] Joachims, T. (1998). Text categorization with suport vector machines : Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK. Springer-Verlag.

[Kate and Mooney, 2006] Kate, R. J. and Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 913–920, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Kuich and Salomaa, 1986] Kuich, W. and Salomaa, A. (1986). *Semirings, automata, languages.* EATCS monographs on theoretical computer science. Springer-Verlag, Berlin, New York, Heidelberg.

[Leslie et al., 2002] Leslie, C., Eskin, E., and Noble, W. S. (2002). The spectrum kernel : a string kernel for svm protein classification. *Pac Symp Biocomput*, pages 564–575.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *J. Mach. Learn. Res.*, 2 :419–444.

[Minsky and Papert, 1987] Minsky, M. and Papert, S. (1987). *Perceptrons - an introduction to computational geometry.* MIT Press.

[Mohri, 2002] Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3) :321–350.

[Mohri et al., 1996] Mohri, M., Pereira, F. C. N., and Riley, M. (1996). Weighted automata in text and speech processing. In *Proceedings of ECAI-96, Workshop on Extended finite state models of language, Budapest, Hungary. John Wiley and Sons.*

[Mohri et al., 2012] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning.* Adaptive computation and machine learning series. MIT Press.

[Mooney and Bunescu, 2006] Mooney, R. J. and Bunescu, R. C. (2006). Subsequence kernels for relation extraction. In Weiss, Y., Schölkopf, P. B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 171–178. MIT Press.

[Nehar et al., 2014] Nehar, A., Benmessaoud, A., Cherroun, H., and Ziadi, D. (2014). Subsequence kernels-based arabic text classification. In *11th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2014, Doha, Qatar, November 10-13, 2014*, pages 206–213.

[Pereira and Riley, 1997] Pereira, F. and Riley, M. (1997). Finite state language processing. In *chapter Speech Recognition by Composition of Weighted Finite Automata*. The MIT Press.

[Pfoh et al., 2013] Pfoh, J., Schneider, C., and Eckert, C. (2013). Leveraging string kernels for malware detection. In *Network and System Security : 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings*, Lecture Notes in Computer Science. Springer.

[Robert and Kevin, 2011] Robert, W. and Kevin, W. (2011). *Algorithms FOURTH EDITION*. EATCS monographs on theoretical computer science. Princeton University.

[Rousu and Shawe-Taylor, 2005] Rousu, J. and Shawe-Taylor, J. (2005). Efficient computation of gapped substring kernels on large alphabets. *J. Mach. Learn. Res.*, 6 :1323–1344.

[Sakarovitch and Thomas, 2009] Sakarovitch, J. and Thomas, R. t. (2009). *Elements of automata theory*. New York, N.Y. Cambridge University Press.

[Seewald and Kleedorfer, 2007] Seewald, A. K. and Kleedorfer, F. (2007). Lambda pruning : an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering. *Adv. Data Analysis and Classification*, 1(3) :221–239.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.

[Strang, 2009] Strang, G. (2009). *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fourth edition.

[Yang and Wu, 2006] Yang, Q. and Wu, X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology and amp ; Decision Making*, 05(04) :597–604.

[Zaki et al., 2005] Zaki, N. M., Deris, S., and Illias, R. (2005). Application of string kernels in protein sequence classification. *Applied Bioinformatics*, 4(1) :45–52.