Matrices score-position, algorithmes et propriétés

Aude Liefooghe

Matrices score-position, algorithmes et propriétés

THÈSE

présentée et soutenue publiquement le 4 juillet 2008

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Aude LIEFOOGHE

Composition du jury

Présidents :	Mireille CLERBOUT, Professeur	LIFL, Université de Lille I
Rapporteurs :	Mireille RÉGNIER, Directeur de recherche INRIA	ALGO, Centre de Roquencourt
	Mathieu RAFFINOT, Chargé de recherche CNRS HDR	LIAFA, Université Paris Diderot
Examinateurs :	Sophie SCHBATH, Directeur de recherche INRA	MIG, Jouy-en-Josas
Directeurs :	Hélène TOUZET, Chargé de recherche CNRS HDR	INRIA, CNRS, LIFL, USTL
	Jean-Stéphane VARRÉ, Maître de conférence	INRIA, CNRS, LIFL, USTL

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UPRESA 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : `direction@lifl.fr`

Table des matières

Introduction

L'objet de cette thèse est la recherche de motifs dans un texte. Les mots peuvent représenter une infinité de notions, ils sont le plus grand vecteur de l'information, du savoir. Un motif est une structure plus complexe qu'un mot, il représente un ensemble de mots et lui donne sens, afin, par exemple, de le retrouver dans un autre mot. La recherche d'un motif dans un autre mot, dans un texte, est un problème fondamental en informatique en général et en algorithmique en particulier. Ce problème a de nombreuses applications dans la gestion et le traitement de données textuelles. Les exemples les plus directs sont la recherche documentaire, la compression de texte, la détection de plagiat, les moteurs de recherche, la reconnaissance vocale, la correction d'erreurs, l'encodage de l'information, le traitement de texte, l'analyse musicale, etc.

***

Les motifs auxquels on s'intéresse ici sont pondérés, ils ont la particularité d'être modélisés de manière probabiliste. Il ne s'agit plus d'un simple mot mais d'un ensemble de mots représenté par une matrice. A chaque position correspond un ensemble de lettres et à chacune d'elles est associé un poids en rapport avec sa probabilité d'apparition. Une telle matrice est appelée matrice score-position. Elle est dérivée de la matrice de fréquence qui associe à chaque position dans le motif et à chaque lettre de l'alphabet une probabilité d'apparition. Les lettres plus fréquentes que la moyenne ont un score positif les autres un score négatif.

Notre problématique est la localisation de matrices score-position dans un texte, c'est-à-dire la recherche de toutes ses occurrences. L'objectif de la thèse est de concevoir des algorithmes efficaces de résolution de ce problème. Afin de réduire la complexité en temps de calcul des algorithmes de recherche, nous nous appuyons sur les propriétés combinatoires et statistiques des matrices score-position fournies.

Notre travail s'inscrit donc, dès la définition du sujet, dans un contexte à la fois algorithmique et combinatoire. Nous avons pris le parti de traiter le problème de manière analogue à celui de la recherche de motifs exacts. Ces travaux de recherche s'intègrent dans le champ de la théorie de l'information en général et de l'algorithmique discrète en particulier.

***

Le choix de s'intéresser à ce problème n'est pas purement théorique mais étroitement lié à l'une de ses principales applications, la bio-informatique.

Depuis toujours l'homme tente de comprendre le mystère de la vie. La biologie est une des approches physique de ce problème. En effet chacune de nos cellules est porteuse de l'information génétique. Celle-ci détermine nos caractères ainsi que le fonctionnement de la machinerie biologique selon le contexte dans lequel se trouve la cellule.

Ces dernières années, grâce aux efforts portés sur les projets de séquençage de génomes, les données récoltées ont crû de manière exponentielle. L'informatique intervient dans le stockage, la visualisation et l'analyse de cette masse de données. La bio-informatique est une thématique pluridisciplinaire qui lie l'informatique et les mathématiques dans le but de résoudre des problèmes biologiques. L'informatique permet, en effet, d'améliorer la compréhension des systèmes décrits formellement. La participation de l'informatique à la résolution de problèmes biologiques passe par la modélisation mathématique des problèmes. Ainsi, à partir de la modélisation mathématique d'un problème biologique, on peut implémenter de nouveaux algorithmes permettant sa résolution.

On peut voir l'ADN comme un texte formé par un alphabet de quatre lettres : A, C, G et T. Cette vision très simpliste de la réalité biologique, induite par le séquençage du génome, permet néanmoins une certaine compréhension des phénomènes biologiques. Elle apporte le matériel nécessaire à l'étude de certaines problématiques, permettant ainsi de fournir des éléments de réponse. Les multiples représentations d'objets biologiques par des objets textuels ont ouvert un champ de recherche en bio-informatique basé sur l'algorithmique de texte [70, 71, 34, 35].

***

Comprendre le fonctionnement de la cellule est essentiel notamment dans l'étude de certaines maladies comme le cancer. Cela est en partie régie par la concentration de la cellule en protéines. Leur synthèse dépend du code génétique mais aussi de l'environnement, du contexte. Le génome est composé de parties dites "non codantes" et de parties "codantes" appelées gènes. Ce sont ces derniers qui sont synthétisés en protéines. Cette transformation du gène en protéine est constitué de deux étapes : l'ADN est d'abord transcrit en ARN puis l'ARN est traduit en protéine. Ainsi "le plan de construction" de la protéine est donné par le code génétique mais sa synthèse dépend du contexte : la fonction de la cellule, son environnement, son état, sa température, etc.

Nous nous intéressons, dans le cadre de cette thèse, à l'adaptation de la cellule à son environnement et plus particulièrement à la régulation des gènes, c'est-à-dire à leur synthèse ou non en protéines selon le contexte. Analyser la régulation des gènes est un objectif majeur de la biologie moléculaire. Certains mécanismes comme la fixation de protéines sur l'ADN influent sur le "comportement" de la cellule. Par exemple, pour que la transcription puisse s'initier il est nécessaire que certaines molécules soient fixées en amont du gène. Cette région s'appelle le promoteur. Mieux comprendre le mécanisme de liaison de ces protéines à certains loci spécifiques de l'ADN demande de répondre à certaines questions biologiques. Quelles sont ces protéines ? Quelles sont les sites où elles se fixent ? Notre problématique biologique concerne la localisation de *sites de fixations de facteurs de transcription* (SFFT) dans une séquence d'ADN.

Afin de résoudre cette problématique, on peut modéliser l'ADN par un texte et les sites de fixation par des motifs textuels. Différents types de motifs permettent une telle modélisation. Mais le modèle le plus utilisé par les biologistes est la matrice score-position.

***

Ce mémoire de thèse est divisé en quatre chapitre.

Le premier chapitre pose le contexte. Dans un premier temps nous définissons un ensemble d'objets utiles à la compréhension de la thèse. Puis nous présentons différents types de motifs, du mot unique à un ensemble de mots. Pour le cas d'un ensemble de mots, nous présentons différentes modélisations : alphabet étendu, expressions rationnelles, modèles de Markov. Dans une troisième partie, nous exposons la problématique biologique à laquelle on s'intéresse : la localisation de SFFT. Nous décrivons les objets biologiques permettant d'appréhender la problématique biologique. Le problème est finalement formellement posé en détaillant la modélisation d'un SFFT par une matrice score-position et la notion d'occurrence.

Le deuxième chapitre expose les solutions proposées par la communauté d'abord pour la recherche d'un simple mot puis pour la recherche de matrices score-position. Nous présentons trois approches ainsi que des algorithmes classiques les illustrant : l'accélération des opérations de comparaison, le pré-traitement du texte et le pré-traitement du motif. Nous détaillons un peu plus la partie pré-traitement du motif en expliquant l'approche par préfixe, l'approche par suffixe et l'approche par facteur. S'agissant de la recherche de matrices score-position seules des méthodes d'accélération des opérations et de pré-traitement du texte ont été proposées. Ce qui a motivé le choix de l'approche de pré-traitement du motif développée dans la thèse. Enfin, nous discutons du sens à donner aux occurrences d'un motif, et ce grâce au calcul de statistiques et probabilités soit pour une occurrence, soit pour le nombre trouvé dans le texte, et pour un motif représentant un mot, un ensemble de mots ou une matrices score-position .

Le troisième chapitre présente notre adaptation de l'algorithme de Knuth, Morris et Pratt [18] à la localisation de matrices score-position dans un texte. Grâce au pré-traitement du motif on calcule une table de décalages qui permet de ne pas tester toutes les positions du texte. Cette approche du problème semble intéressante du point de vue de son application bio-informatique aux SFFTs. En effet, les banques de matrices sont stables dans le temps, il semble donc rentable de pré-calculer des informations sur ces données pérennes. Nous présentons une extension de Morris-Pratt [39] et de Knuth-Morris-Pratt ainsi qu'un troisième algorithme inspiré des précédents, et plus efficace. Nous formalisons le problème en étendant la notion de bord connue pour un seul mot. Nous proposons à chaque fois les algorithmes permettant de réaliser le pré-traitement et le calcul des occurrences dans le cas des matrices score-position. Nous présentons les résultats obtenus qui montrent des gains sensibles.

Le quatrième et dernier chapitre s'intéresse au problème de la localisation multiple de matrices score-position dans un texte. Les facteurs de transcription qui se fixent en amont du gène ne sont pas indépendants les uns des autres mais s'organisent en modules et interagissent. Ainsi, il est fréquent de procéder simultanément à la recherche des occurrences de plusieurs matrices. La solution que nous proposons est à nouveau de pré-traiter les motifs mais, cette fois, tous en même temps. Toujours pour les mêmes raisons de stabilité des données biologiques modélisées par des matrices score-position, le pré-calcul est justifié. Nous proposons une structure d'index calculée de manière optimale selon l'espace mémoire disponible pour

le stockage. Cela nous a conduit à une solution originale de découpage de matrices en sous-matrices qui réduit significativement les temps de recherche. Nous proposons également des extensions pour le cas où le jeu de matrices est constitué de matrice similaires. Nous montrons également que cette structure d'index peut s'appliquer à une matrice et se coupler à notre extension de Knuth-Morris-Pratt.

Le développement de ces algorithmes a fait l'objet d'un logiciel développé en C++ et disponible en ligne, TFM-scan.

Chapitre 1

Des mots aux matrices score-positions, Problèmes

Sommaire

Ce premier chapitre est consacré à la caractérisation d'un ensemble de mots. La compression d'un ensemble de mots dans un motif permet à la fois de lui donner du sens et le localiser dans un autre mot. Nous commençons par la définition du problème de la recherche de motifs.

Les mots peuvent représenter une infinité de notions, ils sont le plus grand vecteur de l'information, du savoir. La recherche de motifs dans un texte est un problème fondamental en informatique. Ce problème a de nombreuses applications dans la gestion et le traitement des données textuelles, comme la recherche documentaire, les moteurs de recherche, la compression de texte, la détection de plagiat, la reconnaissance vocale, la correction d'erreurs, l'encodage de l'information, le traitement de texte, l'analyse musicale, etc.

Différents types de motifs permettent une telle modélisation. La conception de motifs a été introduite par la théorie de l'apprentissage [4, 29, 30]. Nous poursuivons par une présentation non exhaustive des motifs exacts, des expressions rationnelles, des motifs à distance et des modèles probabilistes.

La modélisation de l'ADN par un texte de quatre lettres : A, T, G et C est une représentation très simpliste de la réalité biologique. Cependant elle permet une certaine compréhension des phénomènes biologiques et apporte des éléments de réponses aux problématiques. Les multiples représentations d'objets biologiques par des objets textuels ont ouvert un champ de recherche en bio-informatique basé sur l'algorithmique de texte [70, 71, 34, 35].
Comprendre la régulation des gènes est un objectif majeur de la biologie moléculaire. Cette régulation s'opère en partie par la fixation d'objets biologiques, comme les facteurs de transcriptions, sur l'ADN. Ainsi, une problématique biologique importante est la recherche de *sites de fixations de facteurs de transcription* (SFFT) dans une séquence d'ADN. Les biologistes décrivent l'ensemble des séquences correspondants aux sites d'un facteur par une matrice score-position. Cette problématique et sa modélisation bio-informatique seront détaillées dans la troisième section de ce chapitre.
Le modèle des matrices score-position, sur lequel porte ce travail de thèse, et sa localisation dans un texte clôtureront ce chapitre.

1.1 Définitions et notations

Dans cette section nous introduisons quelques définitions et notations, d'abord sur les mots et leurs recherches dans un texte, puis sur les motifs.

1.1.1 Notion de motifs

Définition 1.1 (Mot) *Soit Σ un alphabet fini de lettres. Un* mot *sur l'alphabet Σ est une suite finie de lettres de Σ.*

Définition 1.2 (Mot vide) *Le* mot vide *est une suite de zéro lettre et est noté ϵ.*

Définition 1.3 (Ensemble des mots d'un alphabet) *Soit Σ un alphabet fini de lettres. L'ensemble de tous les mots sur l'alphabet Σ est noté Σ^*.*

Exemple 1.1 *a, b, ϵ, et aba sont des mots de l'alphabet $\{a, b\}$ et appartiennent à l'ensemble $\{a, b\}^*$.*

Définition 1.4 (Longueur) *Soit u un mot. La* longueur *d'un mot u est définie par la longueur de sa suite de lettres associées et est notée $|u|$.*

Exemple 1.2 $|aba| = 3$.

Définition 1.5 (Position) *Soit u un mot et $i = 0, 1, \ldots |u| - 1$. Pour une numérotation des indices commençant à 0, une* position *i de u est notée u_i et est la $i + 1^{ème}$ lettre de u.*

Exemple 1.3 *Soit $u = aba$, $u_1 = b$.*

Définition 1.6 (Concaténation) *Soit u et v deux mots. La* concaténation *de ces deux mots u et v est le mot composé des lettres de u puis de celles de v et est notée uv.*

Exemple 1.4 *Soit $u = aba$ et $v = ba$, $uv = ababa$.*

Définition 1.7 (Facteur - Préfixe - Suffixe) *Le mot v est un facteur d'un mot u si il existe deux mots x et y tel que $u = xvy$. Si $x = \epsilon$ alors v est un* préfixe *de u et si $y = \epsilon$ alors v est un* suffixe *de u.*

Exemple 1.5 *Soit $u = ababa$, ab est un préfixe de u : **ab**aba, ba est un suffixe de u : aba**ba** et bab est un facteur de u : a**bab**a.*

Définition 1.8 (Occurrence d'un mot) *Soit u et v deux mots. Le mot v contient une* occurrence *de u si v est un facteur de u.*

Définition 1.9 (Occurrence d'un mot) *Soit u et v deux mots. Une* occurrence *de v dans u peut se caractériser par une position dans u. Il y a une occurrence de v dans u à la position i si et seulement si $u_i \ldots u_{i+|v|-1} = v$.*

Définition 1.10 (Recherche d'un mot) *Soit u et v deux mots. La recherche du mot u dans le mot v est la localisation des occurrences de u dans v.*

Exemple 1.6 *Soient les mots $u = aba$ et $v = a$**aba**bb**aba**. Les occurrences de u dans v apparaissent aux positions 1 et 6 du de v.*

Définition 1.11 (Langage) *Soit Σ un alphabet. Un langage sur l'alphabet Σ est un sous-ensemble de Σ^*.*

Un langage peut être défini par un mot ou par un ensemble de mots.

Exemple 1.7 *Soit $\{a, aba, ababa\}$ un langage sur l'alphabet $\{a, b\}$, $\{a, aba, ababa\} \in \{a, b\}^*$*

Définition 1.12 (Langage vide) *Le* langage vide *est un ensemble qui contient zéro mot et est noté $\emptyset$.*

Définition 1.13 (Motif) *Un* motif *représente un langage non vide ne contenant pas le mot vide.*

Un langage peut être représenté par un motif décrit par un mot, une expression régulière, une distance à un mot, un modèle probabiliste, etc. Ces différents motifs sont présentés dans les sections 1.2.1 1.2.3 et 1.2.4.

Il existe deux termes anglais : *motif* et *pattern* pour le terme français motif. Le mot *motif* définit des motifs courts, exacts (ou peu dégénérés) et continus alors que le mot *pattern* définit des motifs plus complexes et dégénérés, voire composés d'un ensemble de motifs. Nous employons ici le terme générique français motif pour désigner l'ensemble de ces définitions, sachant que le type de motif utilisé dans le cadre de cette thèse est le *pattern*.

1.1.2 Recherche de motifs

Les notions de position, de longueur, d'occurrence et de recherche sur les mots s'étendent aux motifs.

Définition 1.14 (Occurrence d'un motif) *Soit m un motif et T un texte. Une* occurrence *de m dans T peut se caractériser par une position dans T. Il y a une occurrence de m dans T à la position i si et seulement si $T_i \dots T_{i+|m|-1}$ est un mot du langage de m.*

Définition 1.15 (Recherche d'un motif) *Soit m un motif et T un texte. La recherche du motif m dans le texte T est la localisation des occurrences de m dans T.*

1.1.3 Significativité statistique

Significativité d'une occurrence

Lors de la recherche de motifs, on peut se demander quelle est la probabilité d'avoir une occurrence par hasard. Autrement dit, supposons que l'on ait trouvé une occurrence du motif, est ce le hasard ou non ?

Plus formellement, le problème statistique sous-jacent est le test d'hypothèse unilatérale suivant.

Définition 1.16 (Test statistique : significativité d'une occurrence)
Hypothèse nulle H_0 *: L'occurrence trouvée est due au hasard*
Hypothèse alternative H_1 *: L'occurrence trouvée a un sens, est significative*

Définition 1.17 (p-valeur d'une occurrence) *La* p-valeur *d'une occurrence est sa probabilité d'être due au hasard.*

$$p\text{-}valeur = \mathbb{P}(H_0) = \mathbb{P}(occurrence\ due\ au\ hasard)$$

Ainsi la p-valeur d'une occurrence donne la significativité statistique d'une occurrence.

Exemple 1.8 *Trouver le motif aa dans un texte composé de nombreux a est moins significatif que de le trouver dans un texte composé de peu de a.*

L'occurrence trouvée est-elle suffisamment significative pour être considérée ou sera-t-elle attribuée au hasard ? Afin de décider du test statistique, de répondre à la question *Est-ce que l'occurrence a été trouvée par hasard ?*, il faut décider d'un risque seuil : α, niveau de risque de première espèce. Ainsi la décision est la suivante

$$\begin{aligned} \text{Acceptation de } H_0 \quad & \text{si p-valeur} \geq \alpha \\ \text{Rejet de } H_0 \quad & \text{si p-valeur} < \alpha \end{aligned}$$

La p-valeur seuil α est donc le risque de rejeter l'hypothèse nulle à tort. Ce risque est appelé risque de première espèce. Calculer la p-valeur d'une occurrence d'un motif à une position d'un texte nécessite de définir un modèle de texte, différents modèles de texte seront présentés dans les sections 1.1.4.0 et 1.1.4.0. La significativité d'une occurrence n'a de sens que par rapport à un modèle, car c'est lui qui détermine la probabilité d'avoir, à une position, une occurrence par hasard.

Significativité du nombre d'occurrences

Un deuxième problème statistique, lié à la recherche de motifs, nous intéresse dans ce cadre. Est-ce que le nombre d'occurrences de ce motif dans ce texte est exceptionnel, significativement faible ou fort ? Est-ce que ce motif est rare ou fréquent dans ce texte ?

Plus formellement, le problème statistique sous-jacent est le test d'hypothèse bilatéral suivant.

Définition 1.18 (Test statistique : significativité du nombre d'occurrences)
Hypothèse nulle H_0 *: Le nombre d'occurrences est attendu*
Hypothèse alternative H_1 *: Le nombre d'occurrences est exceptionnel, significatif*

Définition 1.19 (E-valeur du nombre d'occurrences) *La* e-valeur *du nombre d'occurrences est le nombre d'occurrences attendu, le nombre d'occurrences en moyenne.*

$$\textit{E-valeur} = \mathbb{E}(\textit{nombre d'occurrences})$$

Définition 1.20 (P-valeur du nombre d'occurrences) *La* p-valeur *du nombre d'occurrences est la probabilité d'avoir un nombre d'occurrences supérieur ou égal au nombre d'occurrences observées. Soit X la variable aléatoire du nombre d'occurrences, N le nombre d'occurrences observées et M un modèle de texte.*

$$\textit{P-valeur} = \mathbb{P}(X(M) \geq N(M))$$

Ainsi cette p-valeur donne la significativité statistique d'un nombre d'occurrences.

Le nombre d'occurrences trouvé est-il suffisamment significatif pour être considéré comme exceptionnel ou est-il attribué au hasard ? Afin de décider du test statistique, de répondre à la question *Est-ce que le nombre d'occurrences est exceptionnellement rare, exceptionnellement fréquent ou attendu ?* il faut décider d'un risque seuil : α. Ce risque seuil est aussi appelé p-valeur seuil. Ainsi la décision est la suivante.

$$\begin{aligned} \text{Acceptation de } H_0 \quad & \text{si } \tfrac{\alpha}{2} \leq \text{p-valeur} \leq 1 - \tfrac{\alpha}{2} \\ \text{Rejet de } H_0 \quad & \text{sinon} \end{aligned}$$

Le nombre d'occurrences est considéré comme

$$\begin{array}{rl} \text{Attendu} & \text{si } \frac{\alpha}{2} \leq \text{p-valeur} \leq 1 - \frac{\alpha}{2} \\ \text{Significativement rare} & \text{si p-valeur} < \frac{\alpha}{2} \\ \text{Significativement fréquent} & \text{si p-valeur} > 1 - \frac{\alpha}{2} \end{array}$$

La p-valeur seuil α est donc le risque de rejeter l'hypothèse nulle à tort. Ce risque est appelé risque de première espèce. Calculer la p-valeur d'un nombre d'occurrences d'un motif dans un texte nécessite aussi de définir un modèle de texte. La significativité d'un nombre d'occurrences n'a de sens que par rapport à un modèle, car il détermine la distribution du nombre d'occurrences et donc l'écart entre ce qui est observé et ce qui est attendu.

Ainsi les calculs des deux p-valeurs présentées précédemment ne peuvent se faire sans modèle de texte.

1.1.4 Modèles de texte

Définir un modèle de texte permet de calculer ce à quoi on s'attend dans ce texte aléatoire et le comparer à ce qui est observé. Nous présentons dans cette partie deux modèles de texte

Définition 1.21 (Modèle de texte) *Nous présentons dans cette section deux modèles probabilistes de texte. Ils ont la particularité d'être constants tout au long du texte et d'être paramétré sur la composition en mots. Soit Σ un alphabet fini et T un texte de longueur n. Le texte T peut être modélisé par une suite de n variables aléatoires $T_0 \dots T_{n-1}$ à valeur dans l'alphabet fini Σ.*

Modèle de Bernoulli

La particularité de ce premier modèle est de supposer les variables aléatoires représentatives des lettres du texte T_i indépendantes les unes des autres. Les paramètres de ce modèle sont les fréquences des lettres de l'alphabet Σ notées μ_l pour $l \in \Sigma$ avec $\sum_{l \in \Sigma} \mu_l = 1$. Lorsque les paramètres ne sont pas connus ils peuvent être estimés par les fréquences des lettres de textes observés. Si on ne possède pas d'observation on peut supposer la répartition des lettres uniforme (Loi de Laplace). La probabilité de génération d'un mot $u = u_0 \dots u_{L-1}$ selon le modèle de Bernoulli $\mathcal{M}$ se calcule par la suite de multiplications suivantes.

$$\mathbb{P}(u|\mathcal{M}) = \prod_{i=0}^{L-1} \mu_{u_i}$$

Chaînes de Markov

Un modèle de Markov [3] [13] d'ordre k suppose les variables aléatoires représentatives des lettres du texte T_i dépendantes des k variables précédentes. Un tel modèle est donc défini par les probabilités suivantes pour tout $u_{i-k} \dots u_i \in \Sigma$.

$$\mathbb{P}(u_i|u_{i-k} \dots u_{i-1})$$

Le paramètre de ce modèle est sa *matrice des transitions* $|\Sigma|^k \times |\Sigma|$ notée Π telle que pour toute lettre $l \in \Sigma$ et tout mot $u \in \Sigma^k$ l'élément d'indices l et u de Π noté $\Pi_{u,l}$ est défini par

$$\mathbb{P}(l|u)$$

Lorsque la matrice de transitions n'est pas connue, ses éléments $\Pi_{u,l}$ peuvent être estimés par la fréquence du mot ul relativement à la fréquence du mot u d'un texte observé. La probabilité de génération d'un mot $u = u_0 \dots u_{L-1}$ selon le modèle de Markov $\mathcal{M}$ se calcule par la suite de multiplications suivantes.

$$\mathbb{P}(u|\mathcal{M}) = [\sum_{v \in \Sigma^k} \Pi_{v,u_0}] * [\sum_{v \in \Sigma^{k-1}} \Pi_{vu_0,u_1}] * \dots * \Pi_{u_{L-k-1} \dots u_{L-2},u_{L-1}}$$

1.2 Modèles de motifs

La complexité des modèles de motifs va du plus simple, avec des motifs précis et non ambigus tel que les mots, au plus complexe, avec des motifs flous tels que les motifs consensus et les expressions rationnelles et les modèles probabilistes.

La sélection d'un motif se fait en fonction de la pertinence de sa définition. Différents critères peuvent intervenir dans la définition d'un modèle de motif [76]

- sa proximité à un motif exact
- les lettres autorisées à chaque position
- sa forme, par exemple en dyade (deux motifs exacts espacés)
- l'existence de répétitions au sein du motif
- les fréquences des lettres à chaque position
- ...

Ainsi la qualité d'un motif ne dépend pas seulement de sa complexité mais aussi des informations qu'il représente. Un motif est la caractérisation d'un ensemble de mots défini par une suite de lettres ou de symboles. Dans cette partie nous détaillons la conception et le formalisme de différents modèles de motifs.

1.2.1 Motifs exacts

Les motifs les plus simples sont les mots, ils sont aussi appelés motifs exacts ou sans erreur.

Il est avantageux de travailler avec des motifs exacts car les mots ont de bonnes propriétés combinatoires permettant la création d'algorithmes efficaces de recherche de mots.

Lorsqu'il s'agit de caractériser un ensemble de mot par un seul mot, une approche consiste, pour chaque position des mots à compiler, à retenir la lettre le plus fréquente.

Exemple 1.9 *Par exemple, pour l'ensemble de mots de la figure 1.1 le mot représentatif est le suivant*

G G G G C G G

L'application bio-informatique de cette méthode est appelée *motif consensus de base* [25].

G	A	G	G	C	G	G
G	A	G	G	G	G	T
T	C	G	G	G	G	T
A	G	G	G	C	A	G
A	G	G	G	C	G	T
G	T	G	G	C	G	G
T	G	G	G	A	T	G
G	A	G	G	C	G	G
C	G	G	G	C	T	G
G	G	G	G	C	T	G
G	G	G	G	C	A	G

FIG. 1.1: Ensemble de 11 mots

C'est souvent à partir d'un ensemble de mots ayant un certain nombre de caractéristiques communes que l'on cherche à construire un motif.

Modéliser un ensemble de mots par un motif exact est particulièrement adapté au cas où l'ensemble de mots est très homogène. Ce motif à l'avantage d'être simple et concis, il est donc facile à rechercher, à mémoriser et à comprendre.

Cependant, il est trop pauvre et peu expressif pour être adapté au cas général, lorsque les mots de départ sont différents. En effet, il ne restitue qu'une faible partie de l'information contenue dans l'ensemble mots et sa recherche ne permet pas de retrouver les mots qu'il modélise. De plus il est très sensible à l'ensemble de départ.

1.2.2 Motifs à distance

Une manière d'assouplir le modèle est d'autoriser un certain nombre d'erreurs. Les motifs à distance approchent les motifs exacts en tolérant une certaine distance au motif [37]. Nous présentons dans cette partie deux types de motifs à distance : les motifs avec différences [82], connus sous le nom des k-différences, et les motifs avec inégalités [38], connus sous le nom des k-inégalités, en sont des applications.

Ces deux modèles se définissent à partir d'un motif exact. Ils modélisent, pour un entier naturel k donné, l'ensemble des mots à une distance d'au plus k du motif exact lui correspondant.

Concernant les motifs avec différences la distance est mesurée par la *distance de hamming*. Elle compte, entre deux mots de même longueur, le nombre de positions pour lesquelles leurs lettres diffèrent.

Concernant les motifs avec inégalités il s'agit de la *distance d'édition* [50]. Plus élaborée que la distance de hamming, elle calcule la distance entre deux mots de longueurs différentes. Elle dépend de la suite d'opérations nécessaires pour passer d'un mot à l'autre. Soit deux mots u et v, ces opérations sont appelées *opérations d'éditions* et permettent de passer d'un mot u à un mot v. Elles sont les suivantes.

- la substitution d'une lettre de u par une lettre de v à une position donnée
- la délétion (ou suppression) d'une lettre de u à une position donnée
- l'insertion d'une lettre de v dans u à une position donnée

Un coût est associé à chacune de ces opérations. Soit S l'ensemble des suites d'opérations d'édition transformant u en v. Le coût d'une suite s est la somme des coûts des opérations qui la compose. La distance d'édition entre u et v est définie par la formule suivante.

$$\arg\min_{s \in S} \{\text{coût de } s\}$$

Ces modèles permettent d'éviter la sur-adaptation à l'ensemble de mots de départ tout en restant compréhensibles et facilement détectables. Néanmoins, ils ne représentent pas l'ensemble de l'information contenue dans l'ensemble de mots, en particulier la variabilité des lettres à chaque position.

1.2.3 Expressions rationnelles

Afin de modéliser des motifs plus complexes, ambigus et/ou avec erreurs, des modèles plus souples sont nécessaires. Nous présentons les expressions rationnelles avant de présenter deux cas particuliers : les motifs à jokers et les motifs à alphabet étendu puis une représentation.

Expressions rationnelles

Les expressions rationnelles sur un alphabet Σ décrivent les langages rationnels et sont composées de constantes et d'opérateurs. Les constantes définies sont l'ensemble vide $\emptyset$, le mot vide ϵ et les lettres de l'alphabet Σ. Les opérateurs définis sont, par ordre de priorité croissante, l'union ensembliste ($+$), la concaténation ($.$ ou rien) et la fermeture de Kleene ou l'étoile ($*$). Les expressions rationnelles se définissent récursivement par :

- le langage vide
- les langages de mots à une lettre
- les langages formés à partir de l'étoile, de l'union et de la concaténation de langages rationnels

Exemple 1.10 *Par exemple, l'expression rationnelle représentante de l'ensemble de mots de la figure 1.1 est la suivante.*

$$\boldsymbol{(A+G+T).\,(A+C+G+T).\,(G).\,(G).\,(A+C+G).\,(A+G+T).\,(G+T)}$$

Motifs à jokers

Le motif à jokers [31] est un cas particulier d'expression rationnelle. Il consiste à ajouter des jokers aux motifs exacts. Un joker est une lettre représentative de l'alphabet. Soit Σ l'alphabet et $*$ la lettre jocker, un motif à jokers est un mot appartenant à l'ensemble $(\Sigma \cup \{*\})^*$. Pour un ensemble de mots donnés, les positions homogènes sont représentées par une lettre de l'alphabet des mots de départ tandis que les positions hétérogènes sont représentées par des jokers.

Exemple 1.11 *Par exemple, l'expression rationnelle représentante de l'ensemble de mots de la figure 1.1 est la suivante.*

$$\boldsymbol{* * G\ G * * *}$$

Motifs à alphabet étendu

A partir de l'alphabet des mots de l'ensemble de départ il est possible de définir un alphabet étendu. Soit Σ l'alphabet de l'ensemble de mots de départ, son alphabet étendu contient

- l'alphabet Σ
- une lettre jocker représentative de Σ : n'importe quelle lettre de Σ est autorisée
- l'ensemble de lettres représentant les lettres négatives de Σ :Toute les lettres de Σ sauf une sont autorisées
- l'ensemble de lettres représentant chacune un ensemble de lettres de Σ possible, toutes les alternatives de lettres possibles, les alternatives entre deux lettres de Σ jusqu'aux alternatives entre $|\Sigma| - 1$ lettres de Σ : une des lettres de l'alternative

Une application bio-informatique de cette méthode est appelée *motif consensus dégénérés* et s'appuie sur l'alphabet étendu de l'alphabet des acides nucléiques appelé code IUPAC pour International Union of Pure and Applied Chemistry (voir figure 1.2).

code	**description**
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
U	Uracil
R	Purine (A ou G)
Y	Pyrimidine (C, T, ou U)
M	C ou A
K	T, U, ou G
W	T, U, ou A
S	C ou G
B	C, T, U, ou G (pas A)
D	A, T, U, ou G (pas C)
H	A, T, U, ou C (pas G)
V	A, C, ou G (pas T ni U)
N	une base (A, C, G, T ou U)

FIG. 1.2: code IUPAC

La méthode proposée par Fondrat et Kalogero [24] est de représenter une position par une seule lettre lorsqu'elle a une fréquence d'au moins 60%, par deux lettres si leurs fréquences sont d'au moins 35%, par trois si leurs fréquences sont de plus de 20% et par l'alphabet si les lettres sont quasiment équi-réparties.

Exemple 1.12 *Par exemple, le motif consensus dégénéré représentant l'ensemble de mots de la figure 1.1 selon la méthode de Fondrat et Kalogero est le suivant.*

N N G G C N N

Exemple de représentation des expressions rationnelles

On peut représenter les expressions rationnelles grâce à la syntaxe suivante. Soit Σ l'alphabet de l'ensemble de mots de départ.

- l'alphabet Σ
- l'ambiguïté inclusive entre un ensemble de lettres s'écrit avec des crochets : [ ]. Exemple : [ILVM]
- l'ambiguïté exclusive entre un ensemble de lettres s'écrit avec des accolades : **{}**. Exemple : {FWY}
- la lettre jocker s'écrit **x**
- les répétitions de la position précédente s'écrit avec des nombres entre parenthèses : **(n)**. Exemple : [RD](2)
- une insertion variable comprise entre n et m s'écrit avec la lettre x suivie de deux nombres entre parenthèses : **x(n,m)**. Exemple : x(2,4)
- la séparation entre chaque position s'écrit avec le symbole '**-**'

Une application bio-informatique de cette syntaxe est appelée *motif prosite* [8] et a été développée spécifiquement pour représenter les motifs biologiques protéiques.

Exemple 1.13 *L'exemple suivant est un motif PROSITE.*

D - x - [DNS] - {ILVFYW} - [DENSTG] - {GP} - [DENQSTAGC] - x(2) - [DE]

Les expressions rationnelles permettent la prise en compte des insertions et des délétions dans le motif. En effet, extraire un motif d'un ensemble de mots peut nécessiter l'insertion de *sauts* dans certains d'entre eux afin de minimiser leurs distances deux à deux.

Exemple 1.14 *Par exemple, on peut ajouter des sauts à certains mots de la figure 1.3 afin de minimiser leurs distances deux à deux. (voir figure 1.4)*

V	E	D	L	I	R	Y		
V	E	D	L	R	R	Y		
P	N	E	L	R	R	F		
D	N	K	A	A	L	R	R	F
A	E	E	L	A				

FIG. 1.3: Ensemble de 5 mots

Par rapport aux motifs présentés précédemment les expressions rationnelles ont les avantages suivants :

- la capture de l'essentiel du motif
- la mise en valeur des positions hommogènes
- l'expression de la variabilité des mots en décrivant chaque position par un ensemble de lettres

V	E	D	-	-	L	I	R	Y
V	E	D	-	-	L	R	R	Y
P	N	E	-	-	L	R	R	F
D	N	K	A	A	L	R	R	F
A	E	E	-	-	L	A	-	-

FIG. 1.4: Ensemble de 5 mots avec ajout de sauts

– la prise en compte des insertions et des délétions d'un ensemble de mots.

Cependant elles présentent les inconvénients suivants.

– la variabilité des positions n'est pas quantifiée, la composition en lettre de chaque position n'est pas exprimée
– l'indépendance des positions

Cela est permis par les modèles probabilistes.

1.2.4 Modèles probabilistes

Modèles de markov cachés profils

Les modèles de Markov cachés profils [23] sont un type particulier de modèles de Markov cachés [47], Hidden Markov Models (HMM) en anglais. Ce modèle, d'abord très utilisé pour la reconnaissance de la parole, s'applique à la bio-informatique à partir de 1993 [19]. Nous présentons le cas général avant le cas particulier.

Les modèles de Markov cachés, très proches des automates probabilistes, sont composés d'états, de transitions, de probabilités de transitions et de probabilités d'états. Chaque état est associé à une lettre d'un l'alphabet Σ qui est générée à chaque passage.
Plus formellement un automate à états cachés de Markov est défini par un triplet d'ensembles $\{S, A, B\}$. Ils sont définis de la manière suivante.

– S contient L états ainsi que deux états particuliers : *début* et *fin*.
– S_i est le $i+1^{\text{ème}}$ état
– $a_{i,j}$ est la probabilité de la transition $S_i \rightarrow S_j$
– $b_i(k)$ est la probabilité d'émission de la lettre k à partir de l'état S_i

Avec

– $\sum_j a_{ij} = 1$ la somme des probabilités des transitions partant d'un état est égale à 1
– $\sum_k b_i(k) = 1$ la somme des probabilités des émissions partant d'un état est égale à 1.

Pour générer un mot avec un modèle de Markov caché il faut

1. partir de l'état *début*
2. passer d'état en état selon les probabilités de transition
3. générer une lettre à chaque passage dans un état selon les probabilités d'émission de l'état courant
4. itérer les étapes 2 et 3 jusqu'à ce que l'état *fin* soit atteint

Le nombre de mots possibles est fini. La probabilité de génération d'un mot $u = u_0 \dots u_{L-1}$ selon le modèle de Markov caché $\mathcal{M}$ se calcule par la suite de multiplications suivantes.

$$P(u|\mathcal{M}) = a_{début,0} \left[\prod_{i=0}^{L-2} b_i(u_i)a_{i,i+1}\right] b_{L-1}(u_{L-1})a_{L-1,fin}$$

Les automates définis par ces modèles sont stochastiques et non déterministes, on peut générer un même mot de manières différentes. Ces modèles sont dit *de Markov* car leurs probabilités de transitions dépendent de l'état courant et ils sont dit *cachés* car ils génèrent des mots sans la suite d'états qui les ont émis.

Les modèles de Markov cachés profils sont un type de modèle de Markov cachés particulièrement adapté à la caractérisation d'un ensemble de mots alignés. Afin d'illustrer cette partie nous allons nous appuyer sur l'ensemble de mots de la figure 1.4.

La première étape consiste à construire une suite d'états identiques, chacun correspondant à une position dans le motif, et une suite de transitions de probabilité 1 permettant de passer d'un état au suivant. Ces états sont appelés les *états match* et notés M_i. La probabilité de transition de l'état M_i vers l'état M_j est noté a_{M_i,M_j} et la probabilité d'émission de la lettre l par l'état M_i est noté $b_{M_i}(l)$.

Exemple 1.15 *Par exemple, la figure 1.5 illustre cette première étape pour l'ensemble de mots de la figure 1.4.*

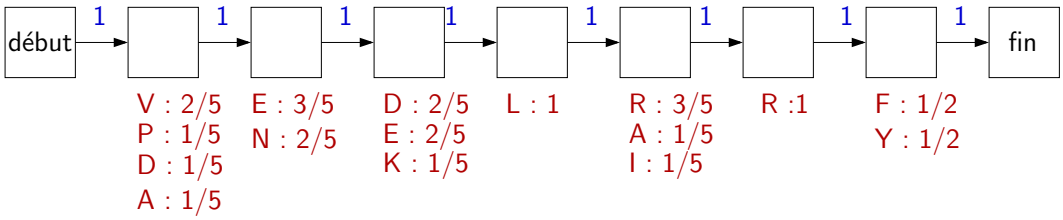

FIG. 1.5: Les états match, les probabilités d'émissions et de transitions du modèle de Markov caché de l'ensemble de mots de la figure 1.4

La deuxième étape consiste à prendre en compte les gaps ; d'abord les insertions puis les délétions.

Les insertions sont les positions trop pauvres en lettre, trop riches en gap, pour avoir fait l'objet d'un état match.
Ces positions sont représentées par les *états d'insertion* noté I_i.
L'état I_i a

- une transition arrivant de l'état M_i de probabilité notée a_{M_i,I_i}
- une probabilité d'émission de la lettre l noté $b_{I_i}(l)$
- boucle de transition sur lui-même, permettant les insertions multiples
- et une transition allant vers l'état M_{i+1}
- et une transition allant vers l'état D_{i+1} (voir ci-dessous) s'il existe

Exemple 1.16 *Par exemple, la figure 1.6 illustre l'ajout des états d'insertion à l'automate des la figure 1.5 afin de mieux représenter l'ensemble de mots de la figure 1.4.*

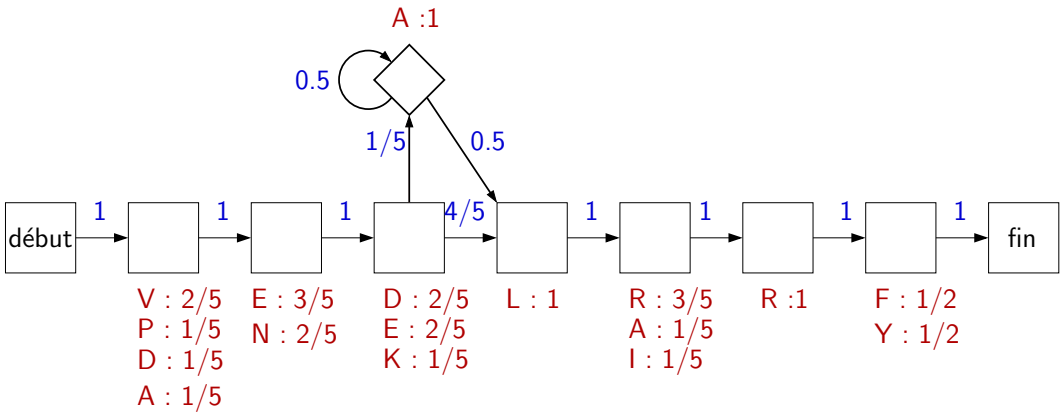

FIG. 1.6: Les états match, **les états d'insertions**, les probabilités d'émissions et de transitions du modèle de Markov caché de l'ensemble de mots de la figure 1.4

Les délétions pourraient être représentées par des transitions entre deux états match ne se suivant pas. Cela ajouterait inutilement un nombre considérable de transitions. Ainsi les délétions font l'objet d'états spécifiques appelés *états de délétion* et notés D_i. Ces états ont la particularité de n'émettre aucune lettre, donc de générer des gaps.
L'état D_i a

- une transition arrivant de l'état M_{i-1} de probabilité notée a_{M_{i-1},D_i}
- une transition arrivant de l'état D_{i-1} s'il existe
- une transition arrivant de l'état I_{i-1} s'il existe
- une transition allant vers l'état M_{i+1}
- une transition allant vers l'état D_{i+1} s'il existe
- et une transition allant vers l'état I_i s'il existe

Exemple 1.17 *Par exemple, la figure 1.6 illustre l'ajout des états de délétions à l'automate des la figure 1.6 afin de mieux représenter l'ensemble de mots de la figure 1.4.*

Afin d'estimer les probabilités d'émissions et de transitions on s'appuie sur l'ensemble des mots de départ. Les probabilités d'émission d'un état sont estimées par les fréquences des lettres à la position correspondante et les probabilités de transitions selon la proportion de mots passant par la transition. Pour ne pas sur-adapter le modèle à l'ensemble de mots de départ on peut corriger les probabilités observées par l'ajout d'un pseudo-compte. Cette méthode revient à ajouter à l'ensemble de mots de départ un mot représentatif soit du contexte lorsqu'il est connu soit de la répartition uniforme des lettres (règle de Laplace).

Ce modèle est le plus riche, il a les avantages suivants.

- la capture l'ensemble du motif
- la caractérisation du motif de manière quantitative, la composition en lettre des positions est exprimée
- la prise en compte des insertions et des délétions

Cependant il est plus difficile à représenter et à comprendre.

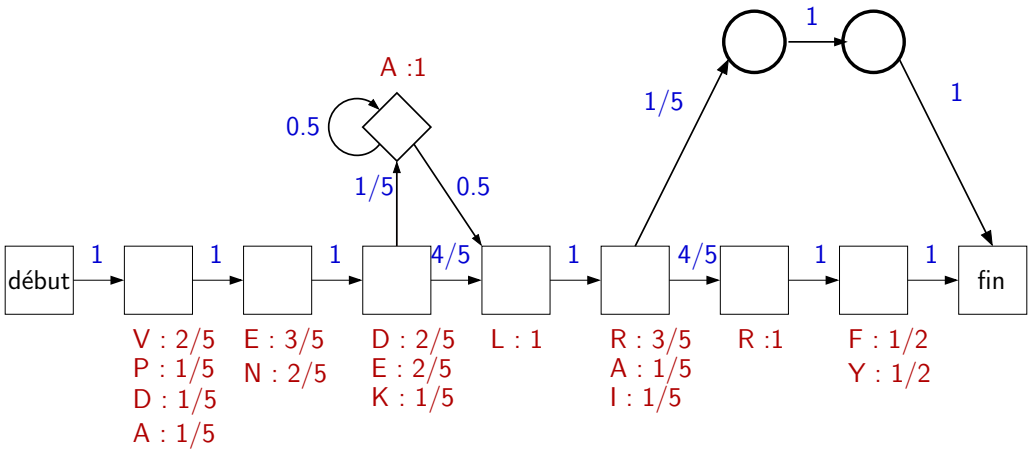

FIG. 1.7: Les états match, les états d'insertions, **les états de délétions**, les probabilités d'émissions et de transitions du modèle de Markov caché de l'ensemble de mots de la figure 1.4

Matrices de fréquences

Le modèle des matrices de fréquence est un cas simplifié du modèle de Markov caché profil. Il s'arrête à la construction des états match (figure 1.5) car il ignore les insertions et les délétions mais exprime quantitativement la variabilité en lettre des positions.

Exemple 1.18 *Par exemple, la figure 1.8 illustre la matrice de fréquence de l'ensemble de mots de la figure 1.1.*

A	2/11	3/11	0	0	1/11	2/11	0
C	1/11	1/11	0	0	8/11	0	0
G	6/11	6/11	1	1	2/11	6/11	8/11
T	2/11	1/11	0	0	0	3/11	3/11

FIG. 1.8: La matrice de fréquence calculée à partir de l'ensemble de mots de la figure 1.1

Les matrices score-position [74] [77] [15] sont une extension des matrices de fréquence. Elles seront présentées dans le section 1.4

1.3 Application bio-informatique, la localisation de SFFTs

La bio-informatique devient un domaine de recherche à part entière dans les années 90. C'est un champ interdisciplinaire qui met l'informatique et les mathématiques au service de problématiques biologiques. D'après Claverie, c'est le décryptage de la *bio-information*. Ce champ est aussi appelé *biologie in silico*, par analogie avec les termes *biologie in vitro* ou *biologie in vivo*.

La bio-informatique s'attaque au problème de l'analyse des séquences biologiques. Avec les grands projets de séquençage de génomes, l'information biologique croît de manière exponentielle mais il reste à comprendre la signification de la plupart de ces séquences.

L'informatique se charge de gérer et de stocker toutes ces données afin de les transformer en une source de connaissance riche et exploitable. La bio-informatique contribue à leurs interprétation par le développement de méthodes d'analyse ou de prédiction des séquences. Certaines grandes problématiques bio-informatiques sont les suivantes.

- l'identification de régions spécifiques
- la recherche de motifs dans une séquence
- la recherche de similarités entre séquences
- la recherche de similitudes d'une séquence avec l'ensemble des séquences d'une base de données
- l'alignement de séquences
- la recherche des structures secondaires de séquences
- le découverte de motifs
- etc.

C'est en utilisant les propriétés combinatoires de l'ADN vue comme un mot que la bio-informatique crée des algorithmes efficaces répondants aux problématiques justes citées.

L'objectif de cette section est de présenter la problématique biologique qui nous intéresse afin de dégager les problématiques bio-informatiques et algorithmiques qui en découlent Nous commençons par présenter le contexte biologique de notre problématique. Puis, nous présentons les mécanismes d'expression des gènes en général. Ces mécanismes permettent la création de protéines à partir de l'ADN et la modulation de leur expression. Enfin, nous détaillons l'un d'eux, la régulation transcriptionnelle, qui motive notre travail sur les facteurs de transcription.

1.3.1 Contexte biologique

La cellule est la brique de base de tout les organismes vivants. Il existe les deux types de cellules suivants.

- **procaryotes**, unicellulaire et sans noyau
- **eucaryotes** *vrai noyau en latin*, délimité par une enveloppe nucléaire isolant du reste leur patrimoine génétique, ce type de cellule s'organise et se différencie de manière coordonnée.

L'information génétique est structurée en chromosomes constitués de plusieurs brins linéaires d'ADN enroulés en double hélice sur des protéines, les histones (voit figure 1.9

Chaque cellule contient la même information mais l'exprime spécifiquement selon sa fonction et son rôle.

L'ADN est formée de molécules élémentaires, les nucléotides qui comprennent un sucre, le désoxyribose, un résidu phosphate et une des 4 bases azotées suivantes.

- adénine A
- cytosine C

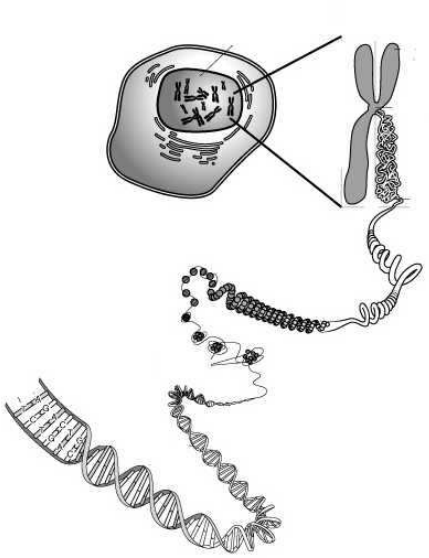

FIG. 1.9: De la cellule à l'ADN
source de l'image : `http ://www.chimie-sup.fr`

– guanine G
– thymine T

Ces 4 bases sont appareillées par paires, A ↔ T et G ↔ C.

Le génome est porté par les chromosomes constitués de gènes en *mosaïque*. Il contient des zones codantes, les gènes, séparés par des zones non codantes (voir figure 1.10).

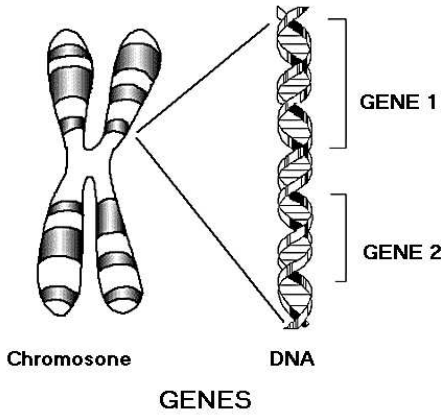

FIG. 1.10: Du chromosome aux gènes
source de l'image : `http ://www.accessexcellence.org/`

Un gène est un fragment d'ADN correspondant à une fonction spécifique. Il peut être synthétisé en protéine, chaîne d'acides aminés structurée tridimensionnellement. Selon le contexte, le milieu extracellulaire, l'état métabolique, il est synthétisé ou non. Ainsi, afin

de s'adapter à son environnement, la cellule peut moduler l'activité de ses gènes grâce aux mécanismes de régulation de l'expression des gènes.

1.3.2 L'expression des gènes

Certains mécanismes complexes de régulation permettent, selon le contexte, d'exprimer ou de réprimer l'expression d'une partie de l'information génétique. C'est à dire de moduler la production des protéines présentes dans la cellule.

Le dogme central est ici la synthèse des protéines à travers ses deux niveaux d'expression des gènes, la transcription et la traduction (voir figure 1.11).

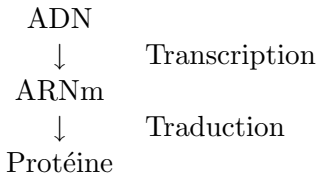

FIG. 1.11: Deux niveaux d'expression des gènes

La transcription

L'ARN, ou acide ribonucléique, est synthétisé à partir de l'ADN par des complexes protéiques : les ARN polymérases ou ribosome Ce niveau se découpe, dans l'ordre, en trois phases comme suit (voir Figure 1.12).

- **l'initiation de la transcription**. L'ARN polymérase se fixe dans une région particulière de l'ADN en amont du gène, appelé **promoteur**, et forme un complexe d'initiation avec d'autres protéines : **les facteurs de transcription**
- **L'élongation**. L'ARN polymérase se déplace le long de l'ADN et synthétise un ARN messager avec des ARN de transfert par complémentarité des bases. L'ARN est constitué des nucléotides A,U,G,C, qui s'apparient comme suit. A↔U, T↔A, G↔C et C↔ G
- **La terminaison**, ou phase d'arrêt de la transcription, est déclenchée par des signaux spécifiques portés par l'ADN

Les ARN messagers sont ensuite traduits en protéine.

La traduction

La traduction synthétise chaque ensemble de trois nucléotides, appelé codon, en acides aminés. Cette correspondance est déterministe et appelée code génétique. L'enchaînement de ces acides aminés forme une protéine.

La régulation

Les mécanismes de régulation d'une cellule lui permettent de s'adapter à son environnement, à ses besoins métaboliques, aux sollicitations extérieures, à son programme de

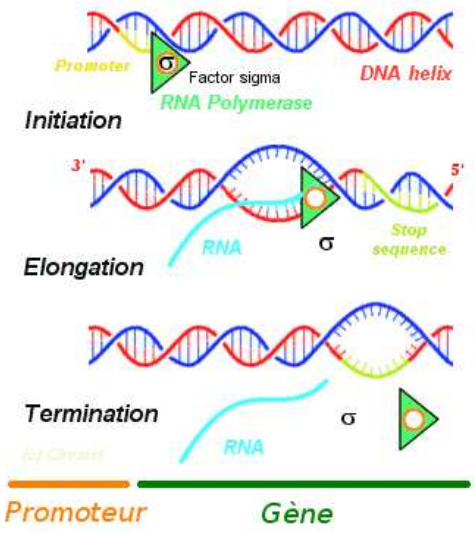

FIG. 1.12: Transcription
source de l'image : `http ://www.geneticengineering.org/`

développement, etc. Ces mécanismes synthétisent uniquement les gènes actifs en protéine. Dans l'ordre, ils peuvent intervenir aux différentes étapes suivantes de l'expression des gènes.

- l'initiation de la transcription
- le transport
- l'initiation de la traduction
- la dégradation des ARNm
- la dégradation des protéines

Différents leviers permettant d'agir sur l'activation ou non des gènes, notre intérêt se porte sur le tout premier : la régulation transcriptionnelle.

1.3.3 La régulation transcriptionnelle

Lors de l'initiation de la transcription, un complexe d'initiation, l'ARN polymérase et les facteurs de transcription, se fixe dans le promoteur. La présence, ou l'absence, de ces facteurs favorise, ou défavorise, le positionnement de l'ARN polymérase sur l'ADN et donc l'initiation de la transcription. L'intensité et la spécificité de la transcription sont déterminées par le nombre et la nature de ces facteurs (voir figure 1.13).

Les domaines de liaison de l'ADN avec les facteurs de transcription sont spécifiques et appelés **site de fixation** ou signaux de régulation[33]. Ces sites correspondent à des séquences courtes d'ADN de faible conservation, dites dégénérées. En effet, certains nucléotides des facteurs de transcription n'entrent pas en jeu dans les mécanismes de fixation. De plus, un site a différents degrés de liaisons avec les facteurs selon leur fonction ou rôle dans la régulation. (voir figure 1.14).

En résumé, les facteurs de transcription conditionnent l'initiation de la transcription donc l'expression des gènes, leurs synthèse en protéine. Ce sont donc des acteurs majeurs

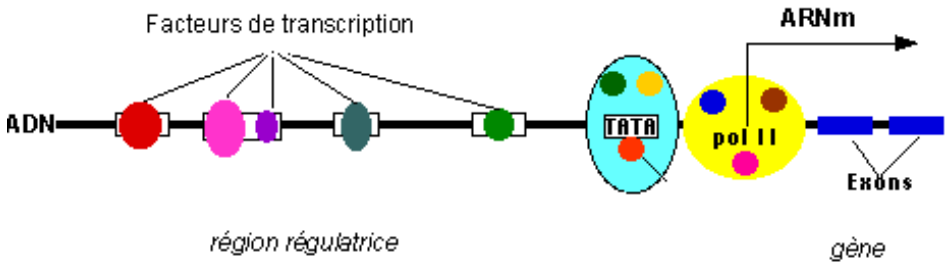

FIG. 1.13: Initialisation de la transcription par le présence de facteurs de transcription
source de l'image : `http ://www.lifl.fr/~touzet/`

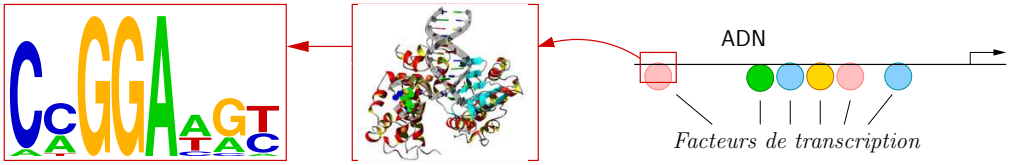

FIG. 1.14: Sites de fixation des facteurs de transcription
source de l'image : `http ://www.lifl.fr/~touzet/`

de la régulation. Il agissent en se fixant à l'ADN, ce qui fait du problème de **la localisation des sites de fixation potentiels des facteurs de transcription** un enjeu dans la compréhension des mécanismes de régulation. Certaines techniques, comme les puces à ADN [21], permettent de mesurer le niveau d'expression d'un ensemble de gènes dans un ensemble de cellules. Ces techniques peuvent être appliquées à un ensemble de gènes co-régulés. Un objectif biologique est de découvrir les facteurs de transcriptions impliqués dans la régulation de ces gènes. Une hypothèse serait la sur ou la sous-représentation sites de fixations de ces facteurs de transcription dans le promoteur. Ainsi, **la recherche de sites de fixations de facteurs de transcription sur ou sous-représentés** est un autre problème d'intérêt biologique.

Les manipulations nécessaires à la résolution de ces deux problématiques sont très longues et coûteuses compte tenu de la quantité de données à traiter. Une approche bioinformatique du problème parait donc opportune. Pour cela il faut modéliser les objets biologiques impliqués dans le problème, les sites de fixation des facteurs de transcription et les séquences d'ADN.

1.3.4 Modélisation des SFFTs

La conception d'un modèle des sites de fixation d'un facteur de transcription se base sur un ensemble de séquences correspondants aux sites reconnus du facteur. La méthode consiste à aligner les mots entre eux avant d'en extraire la *signature*. La première étape est la construction de l'alignement multiple des séquences. La seconde est le choix du modèle de représentation, par exemple parmi ceux présentés dans la section 1.2.

Alignement multiple

L'objectif de l'alignement d'un ensemble de séquences est de mettre en valeur les parties hommogènes en insérant, si cela est nécessaire, des indels, des *sauts* (gaps en anglais). Le résultat de cet alignement est appelé **alignement multiple**. Le problème de l'alignement est fondamental en informatique et consiste à optimiser les distances deux à deux des mots d'un ensemble.

L'alignement multiple est une mise en forme compilée et sans perte d'information de l'ensemble des séquences. Il permet de connaître la variabilité en base de chaque position et de différencier les zones à haute similarité, porteuses de sens, des zones observées par hasard.

Exemple 1.19 *Par exemple, l'alignement multiple de l'ensemble de mots de la figure 1.1 est très simple car il ne nécessite pas d'insertion d'indels (voir Figure 1.15*

G	A	G	G	C	G	G
G	A	G	G	G	G	T
T	C	G	G	G	G	T
A	G	G	G	C	A	G
A	G	G	G	C	G	T
G	T	G	G	C	G	G
T	G	G	G	A	T	G
G	A	G	G	C	G	G
C	G	G	G	C	T	G
G	G	G	G	C	T	G
G	G	G	G	C	A	G

FIG. 1.15: Alignement multiple des 11 mots de le figure 1.1

Il est possible de représenter graphiquement un motif d'alignement multiple grâce au logiciel WebLogo [40] [80]. La taille de chaque lettre de chaque position est proportionnelle à sa fréquence d'apparition dans l'alignement. La hauteur d'une position dépend de l'homogénéité de sa composition en lettres, de son contenu en information (voir section 2.3.2.0).

Exemple 1.20 *Par exemple, le WebLogo de l'alignement multiple de la figure 1.15 est illustré par la figure 1.16*

Choix du modèle des SFFTs

La construction du modèle se base sur l'alignement multiple des séquences Les séquences des sites de fixation d'un facteur de transcription validées expérimentalement sont courtes (entre 6 et 30 bases) et variables, hétérogènes. Leur modélisation doit donc prendre en compte ces spécificités. Nous détaillons les avantages et inconvénients de différents modèles [78].

[2] *source de l'image* : `http ://weblogo.berkeley.edu/`

FIG. 1.16: Représentation graphique de l'alignement multiple de la figure 1.15 [2].

Modéliser ces sites par des motifs consensus de base sans erreur, des motifs exacts, fut un succès chez la levure [46] car les sites de fixation des organismes unicellulaires sont souvent fortement conservés. Cependant ce type de modèle ne permet de retrouver qu'une petite partie des sites connus [78]. Il n'est parfois même pas le site le plus souvent rencontré comme signal. Ajouter des erreurs au modèle permet en partie d'y remédier mais rend le modèle moins sélectif, beaucoup d'occurrences sont trouvées. Une autre solution est l'ajout au modèle d'un deuxième motif consensus de base alternatif. La perte de la sélectivité est moins forte mais on ne retrouve toujours pas l'ensemble des sites connus même en ajoutant des erreurs aux deux motifs.

Le modèle le plus utilisé et reconnu en biologie pour modéliser un site de fixation des facteurs de transcription est la **matrice score-position**. Il existe de grandes banques de données publiques composées de signaux biologiques modélisés par ces motifs : TRANSFAC [85] et Jaspar [63].

Ainsi, à partir d'un ensemble de signaux biologiques découvert expérimentalement représentatif d'une même fonction biologique, il est possible de construire un modèle de cette fonction grâce aux matrices de score-position.

De plus, ces matrices sont adaptées au modèle physique de l'interaction protéine-ADN [10][11]. Il existe une corrélation forte entre les scores des matrices score-position représentative de sites de fixation et l'énergie de l'interaction entre le site et le facteur de transcription. Plus précisément entre le contenu en information de la matrice et l'énergie moyenne de l'interaction [12] [78]. Ainsi le score d'un mot selon une matrice représente l'affinité entre la séquence d'ADN représentée par le mot et le site de fixation représenté par la matrice.

Par contre, ce modèle est beaucoup plus rigide que le modèle de Markov caché profil 1.1.4.0. Il n'autorise pas les insertions, les délétions et les motifs en dyade qui peuvent être indispensables à certains facteurs.

Les matrices score-position sont une version étendue des matrices de fréquence présentées dans la section précédente 1.2.4.0. Nous les détaillons dans la section suivante 1.4.

Ainsi on obtient une modélisation bio-informatique de nos deux problématiques biologiques de localisation de sites de fixation des facteurs de transcription dans l'ADN et de recherche de sites de fixation sur ou sous-représentés.

Les deux problématiques algorithmiques suivantes, centrales à cette thèse, ont une application bio-informatique directe.

- **la localisation de matrices score-position dans un texte**
- **la recherche de matrices score-position sur ou sous-représentées dans un texte**

1.4 Recherche de matrices score-position dans un texte

Le coeur du travail de cette thèse est la localisation de matrice score-position dans un texte. Dans un premier temps nous détaillons la modélisation d'un ensemble de mots par une matrice score-position. Dans un second temps nous présentons le calcul du score d'un mot pour une matrice score-position donnée.

1.4.1 Matrices score-position

Dans cette section nous détaillons les modèles matriciels jusqu'au modèle des matrices score-position. La modélisation de motifs approchés à l'aide de matrices est une caractérisation précise des alignements multiples. En effet, mise à part les indels, les matrices permettent de restituer l'ensemble des informations contenues dans un alignement si l'on considère les positions indépendantes les unes des autres. Cependant il est possible d'étendre ce modèle aux dépendances entre les positions, par exemple aux dépendances d'une position [57].

Nous allons présenter les six types matriciels suivants.
- La *matrice de comptage* qui compte les occurrences de chaque lettre à chaque position
- La *matrice de fréquence* qui déduit de la matrice de comptage les fréquences d'apparition des lettres de chaque position
- La *matrice de fréquence corrigée* qui corrige les fréquences par l'ajout d'un pseudo-poids
- La *matrice de fréquence relative corrigée* qui ramène les fréquences corrigées au contexte, au modèle de texte
- La *matrice d'entropie* qui calcule le contenu informationnel de chaque lettre à chaque position
- Et la *matrice score-position* qui crée, à partir d'une matrice de fréquence et d'information contextuelles, un modèle de score additif

Matrices de comptage

La méthode la plus triviale, pour retranscrire la **composition en lettres des positions** d'un alignement multiple, est le calcul d'une matrice de comptage. Ce motif associe à chaque lettre de l'alphabet un compte selon sa position dans le motif.

Définition 1.22 (La matrice de comptage) *Soit A un alignement multiple de longueur L et Σ un alphabet fini. Une* matrice de comptage *notée C représentative de l'alignement A est une matrice $L \times |\Sigma|$ telle que pour toute lettre $l \in \Sigma$ et toute position $p = 0 \dots L-1$ l'élément d'indices l et p de C noté $C(p,l)$ est défini par le nombre d'occurrences de la lettre l à la position p de l'alignement multiple A.*

Exemple 1.21 *Par exemple, la construction d'une matrice de comptage à partir de l'alignement multiple de la figure 1.15 est illustrée par la figure 1.17.*

G	A	G	G	C	G	G
G	A	G	G	G	G	T
T	C	G	G	G	G	T
A	G	G	G	C	A	G
A	G	G	G	C	G	T
G	T	G	G	C	G	G
T	G	G	G	A	T	G
G	A	G	G	C	G	G
C	G	G	G	C	T	G
G	G	G	G	C	T	G
G	G	G	G	C	A	G

Alignement multiple de 11 séquences

→

A	2	3	0	0	1	2	0
C	1	1	0	0	8	0	0
G	6	6	11	11	2	6	8
T	2	1	0	0	0	3	3

La matrice de comptage construite à partir de l'alignement multiple

FIG. 1.17: Calcul d'une matrice de comptage

La matrice de comptage décrit quantitativement l'alignement multiple. Elle caractérise la composition en lettre des positions mais sans la ramener au total.

Matrices de fréquence

La matrice de fréquence **normalise** la composition en lettres des positions de la matrice de comptage. La matrice de fréquence associe à chaque lettre de l'alphabet une fréquence selon sa position dans le motif.

Définition 1.23 (La matrice de fréquence) *Soit A un alignement multiple de longueur L et Σ un alphabet fini. Une* matrice de fréquence *notée F représentative de l'alignement A est une matrice $L \times |\Sigma|$ telle que pour toute lettre $l \in \Sigma$ et toute position $p = 0 \dots L-1$ l'élément d'indices l et p de F noté $F(p,l)$ est défini par la fréquence de la lettre l à la position p de l'alignement multiple A.*

$$F(p,l) = \frac{C(p,l)}{\sum_{i \in \Sigma} C(p,i)}$$

Exemple 1.22 *Par exemple, la construction d'une matrice de fréquence à partir de la matrice de comptage de la figure 1.17 est illustrée par la figure 1.18.*

Ainsi la probabilité de génération d'un mot $u = u_0 \dots u_{L-1}$, selon la matrice de fréquence F, se calcule par la suite de multiplications suivantes.

$$P(u|F) = \prod_{p=1}^{L} F(p, u_p)$$

Un tel calcul fait par un ordinateur donnera un résultat très approché voir erroné. La précision des calculs par ordinateur ne permet pas une bonne approximation des suites de multiplications de nombres inférieurs à un.

Une autre lacune du modèle est sa sur-adaptation (overfitting en anglais), *lorsqu'on apprend par coeur on ne se trompe jamais.* Les fréquences de la matrice collent parfaitement à

A	2	3	0	0	1	2	0
C	1	1	0	0	8	0	0
G	6	6	11	11	2	6	8
T	2	1	0	0	0	3	3

La matrice de comptage de figure 1.17

↓

A	0,18	0,27	0	0	0,09	0,18	0
C	0,09	0,09	0	0	0,73	0	0
G	0,55	0,55	1	1	0,18	0,55	0,73
T	0,18	0,09	0	0	0	0,27	0,27

La matrice de fréquence construite à partir de la matrice de comptage

FIG. 1.18: Calcul d'une matrice de fréquence

l'ensemble de mots de départ alors qu'il ne s'agit que du sous-ensemble connus des données à représenter. Le modèle est donc trop proche des données de départ.

Matrices de fréquence corrigée

La **correction** du modèle des matrices de fréquence se fait par l'ajout d'un **pseudo-compte** et présente un double intérêt. Classiquement, afin de palier aux problèmes d'approximation que posent les suites de multiplications, on convertit les modèles multiplicatifs en modèle additif en passant à l'échelle logarithmique. Cette conversion nécessite d'éliminer les éléments nuls pour le passage au logarithme. De plus un moyen de remédier à la sur-adaptation d'un modèle est de lui apporter de la souplesse par l'ajout d'un pseudo-compte. Ce qui revient très précisément à ajouter aux données de départ des données représentatives du contexte. Dans notre cas, le contexte est le texte et la donnée à ajouter est un mot suivant le modèle de texte.

Il existe deux méthodes selon que la donnée de départ soit la matrice de comptage ou la matrice de fréquence.

Partir de la matrice de comptage c'est connaître le nombre d'observations de départ. On possède plus ou moins d'observations sur les objets que l'on souhaite modéliser. Il paraît naturel d'accorder plus de crédit à un modèle basé sur un jeux de données important qu'à un modèle base extrait de nombreuses observations qu'à un modèle extrait de peu. Dans ce cas le pseudo-compte représente un nombre de mots du contexte à ajouter aux données, ce qui donne plus de souplesse aux modèles calculés sur peu de données.

Définition 1.24 (La matrice de fréquence corrigée) *Soit Σ un alphabet fini, L un entier naturel, C une matrice de comptage, l une lettre $\in \Sigma$, p une position $= 0 \dots L-1$, c un pseudo-compte et f_l la fréquence attendue de la lettre l. La matrice de fréquence corrigée est*

notée F' et définie par la formule suivante.

$$F'(p,l) = \frac{C(p,l) + c * f_l}{\sum_{l \in \Sigma} C(p,l) + c}$$

Exemple 1.23 *Par exemple, le calcul d'une matrice de fréquence corrigée à partir de la matrice de comptage de la figure 1.17 avec des fréquences en lettre du contexte égales à $0,25$ et un pseudo-compte égal à 1 est illustré par la figure 1.19.*

A	2	3	0	0	1	2	0
C	1	1	0	0	8	0	0
G	6	6	11	11	2	6	8
T	2	1	0	0	0	3	3

La matrice de comptage de figure 1.17

↓

A	2,25/12	3,25/12	0,25/12	0,25/12	1,25/12	2,25/12	0,25/12
C	1,25/12	1,25/12	0,25/12	0,25/12	8,25/12	0,25/12	0,25/12
G	6,25/12	6,25/12	11,25/12	11,25/12	2,25/12	6,25/12	8,25/12
T	2,25/12	1,25/12	0,25/12	0,25/12	0,25/12	3,25/12	3,25/12

La matrice de fréquence corrigée calculée à partir de la matrice de comptage

FIG. 1.19: Calcul d'une matrice de fréquence corrigée

Partir de la matrice de fréquence c'est donner un poids, compris entre 0 et 1, au pseudo-compte en ignorant le nombre de mots sur lesquels le modèle se base.

Définition 1.25 (La matrice de fréquence corrigée) *Soit Σ un alphabet fini, L un entier naturel, F une matrice de fréquence, l une lettre $\in \Sigma$, p une position $= 0 \ldots L-1$, c un pseudo-compte et f_l la fréquence attendue de la lettre l avec $\sum_{l \in \sigma} f_l = 1$. La matrice de fréquence corrigée est notée F' et définie par la formule suivante :*

$$F'(p,l) = \frac{F(p,l) + c * f_l}{1 + c}$$

Exemple 1.24 *Par exemple, le calcul d'une matrice fréquence corrigée score-position à partir de la matrice de fréquence de la figure 1.18 avec des fréquences en lettre du contexte égales à $0,25$ et un pseudo-compte égale à $\frac{1}{10}$ est illustré par la figure 1.20.*

Les matrices de fréquence corrigée ou non donnent une bonne représentation des observations de départ mais indépendamment du contexte, sans tenir compte du modèle de texte.

A	0,18	0,27	0	0	0,09	0,18	0
C	0,09	0,09	0	0	0,73	0	0
G	0,56	0,56	1	1	0,18	0,56	0,73
T	0,18	0,09	0	0	0	0,27	0,27

La matrice de fréquence de figure 1.18

↓

A	0,19	0,27	0,02	0,02	0,10	0,19	0,02
C	0,10	0,10	0,02	0,02	0,67	0,02	0,02
G	0,53	0,53	0,93	0,93	0,19	0,53	0,67
T	0,19	0,10	0,02	0,02	0,02	0,27	0,27

La matrice de fréquence corrigée calculée à partir de la matrice de fréquence

FIG. 1.20: Calcul d'une matrice de fréquence corrigée

Matrices de fréquence relative corrigée

La matrice de fréquence relative corrigée permet de remettre le motif dans son contexte. C'est à dire de rapporter le motif au modèle de texte, de mesurer l'indépendance entre le motif et le modèle de texte.

Définition 1.26 (La matrice de fréquence relative corrigée) *Soit Σ un alphabet fini, L un entier naturel, F' une matrice de fréquence corrigé, l une lettre $\in \Sigma$, p une position $= 0 \dots L-1$ et f_l la fréquence attendue de la lettre l avec $\sum_{l \in \sigma} f_l = 1$. La matrice de fréquence corrigée relative est notée F'' et définie par la formule suivante :*

$$F''(p,l) = \frac{F'(p,l)}{f_l}$$

Exemple 1.25 *Par exemple, le calcul d'une matrice de fréquence relative corrigée à partir de la matrice de fréquence de la figure 1.20 avec des fréquences en lettre du contexte égales à $0,25$ est illustré par la figure 1.21.*

La matrice de fréquence relative corrigée représente les observations de départ en fonction de leur contexte mais ne permettent pas de mesurer le degré d'adéquation entre le mot et le motif.

Matrices d'entropies

Le contenu informationnel d'une matrice peut être mesuré par son entropie. L'*entropie* est une fonction mathématique qui correspond à la quantité d'information contenue ou délivrée par une source d'information, ici la matrice.

Une matrice construite à partir d'un ensemble de mots homogènes, voir identique, apporte beaucoup d'informations sur le motif. Inversement, lorsque les mots sont hétérogènes,

A	0,19	0,27	0,02	0,02	0,10	0,19	0,02
C	0,10	0,10	0,02	0,02	0,67	0,02	0,02
G	0,53	0,53	0,93	0,93	0,19	0,53	0,67
T	0,19	0,10	0,02	0,02	0,02	0,27	0,27

La matrice de fréquence corrigée de figure 1.20

↓

A	0,76	1,08	0,08	0,08	0,4	0,76	0,08
C	0,4	0,4	0,08	0,08	2,68	0,08	0,08
G	2,12	2,12	3,72	3,72	0,76	2,12	2,68
T	0,76	0,4	0,08	0,08	0,08	1,08	1,08

La matrice de fréquence relative corrigée calculée à partir de la matrice de fréquence corrigée

FIG. 1.21: Calcul d'une matrice de fréquence corrigée

spécialement lorsque les lettres des positions sont équi-réparties, l'information portée par le modèle est faible. Statistiquement la matrice d'entropie est définie comme suit.

Définition 1.27 (La matrice d'entropie) *Soit Σ un alphabet fini, L un entier naturel, F' une matrice de fréquence corrigée , l une lettre $\in \Sigma$ et p une position $= 0 \dots L-1$. La matrice d'entropie est notée E et définie par la formule suivante.*

$$E(p,l) = F'(p,l) \ln F'(p,l)$$

Cette formule d'entropie est indépendante du modèle de texte, elle donne une bonne estimation de l'adéquation du motif au texte lorsque les lettres sont équi-réparties. Dans le cas contraire il vaut mieux prendre en compte le contexte, le modèle de texte. Une matrice calculée à partir d'un ensemble de mots différents du modèle de texte apporte plus d'information sur le motif qu'une matrice calculée à partir d'un ensemble de mots suivant le modèle. La formule de l'entropie relative [12] [79] [26] prend en compte le modèle de texte et est définie comme suit.

Définition 1.28 (La matrice d'entropie relative) *Soit Σ un alphabet fini, L un entier naturel, F' une matrice de fréquence corrigée, F'' la matrice de fréquence relative corrigée de F' , l une lettre $\in \Sigma$, p une position $= 0 \dots L-1$ et f_l la fréquence attendue de la lettre l. La matrice d'entropie est notée E et définie par la formule suivante.*

$$E(p,l) = F'(p,l) \ln F''(p,l) = F'(p,l) \ln \frac{F'(p,l)}{f_l}$$

Chaque lettre contribue selon sa fréquence à l'entropie d'une position. L'entropie d'une position p d'une matrice de fréquence F, notée E_p, est définie comme suit.

$$E_p(F) = \sum_{l \in \Sigma} E(p,l)$$

Chaque position contribue indépendamment à l'entropie totale. L'entropie d'une matrice, notée E_p, est définie comme suit.

$$E(F) = \sum_{p=0}^{L-1} \sum_{l \in \Sigma} E(p, l)$$

La matrice d'entropie donne une bonne mesure de l'adéquation entre un mot et le motif.

Matrice score-position

La matrice score-position [74] [15] [77] est un modèle très proche de la matrice d'entropie. Elle est obtenue par la méthode du log-ratio qui consiste à faire, pour chaque élément de la matrice, le logarithme du ratio entre la fréquence du motif et la fréquence attendue. Elle est un modèle de score additif. Elle augmente la fiabilité du calcul des probabilités des mots du motif en transformant le modèle multiplicatif de la matrice de fréquence en un modèle additif. Elle permet la comparaison du motif à son contexte par la création d'un système de score, les éléments positifs de la matrice correspondent aux éléments de fréquence du motif supérieur à la fréquence du contexte, inversement pour les éléments négatifs.

Définition 1.29 (La matrice score-position) *Soit F'' une matrice de fréquence relative corrigée, Σ un alphabet fini et L un entier naturel. Une* matrice score-position *notée M est une matrice $L \times |\Sigma|$ telle que pour toute lettre $l \in \Sigma$ et toute position $p \in \{0 \dots L-1\}$ l'élément d'indices l et t de M noté $M(p, l)$ est défini par la formule suivante.*

$$M(p, l) = \ln F''(p, l)$$

Exemple 1.26 *Par exemple, le calcul d'une matrice score-position à partir de la matrice de fréquence relative corrigée de la figure 1.21 est illustrée par figure 1.22*

Les motifs modélisés par des matrices score-position expriment toute l'ambiguïté et la complexité d'un ensemble de mots. Ils mesurent l'adéquation entre un mot et le motif.

1.4.2 Recherche de matrices score-position dans un texte

Maintenant que nous avons défini le modèle des matrices score-position à partir d'un alignement multiple nous présentons comment localiser leurs occurences dans un texte. Nous commençons par quantifier l'adéquation d'un mot au modèle en définissant le score d'un mot selon une matrice score-position. Ensuite, nous définissons le problème de la recherche d'occurrences d'une matrice dans un texte. Enfin, nous terminons cette partie et ce chapitre sur la significativité statistique d'une occurrence d'une matrice dans un texte et du nombre d'occurrences d'une matrice dans un texte.

Le score d'une lettre dans un mot, selon une matrice score-position, est l'élément de la matrice correspondant à la lettre et à sa position dans le mot. Le score d'un mot complet est la somme des scores de ses lettres et est défini comme suit.

A	0,76	1,08	0,08	0,08	0,4	0,76	0,08
C	0,4	0,4	0,08	0,08	2,68	0,08	0,08
G	2,12	2,12	3,72	3,72	0,76	2,12	2,68
T	0,76	0,4	0,08	0,08	0,08	1,08	1,08

La matrice de fréquence relative corrigée de la figure 1.21

↓

A	-0,29	0,08	-2,49	-2,49	-0,88	-0,29	-2,49
C	-0,88	-0,88	-2,49	-2,49	8	-2,49	-2,49
G	0,73	0,73	1,32	1,32	-0,29	0,73	1,01
T	-0,29	-0,88	-2,49	-2,49	-2,49	0,08	0,08

La matrice score-position calculée à partir de la matrice de fréquence relative corrigée de la figure 1.21

FIG. 1.22: Calcul d'une matrice score-position à partir d'une matrice de fréquence relative corrigée

Définition 1.30 (Score d'un mot selon une matrice score-position) *Soit Σ un alphabet fini, M une matrice score-position de longueur L indexée par $\{0 \dots L-1\}$, p une position $= 0 \dots L-1$ et $u = u_0 \dots u_{L-1}$ un mot $\in \Sigma^L$. Le* score *de u selon M est noté $Score(M, u)$ et défini par*

$$Score(M, u) = \sum_{i=0}^{L-1} M(i, u_i)$$

Exemple 1.27 *Par exemple, le calcul du score du mot **A A G G C T T** selon la matrice score-position M de la figure 1.22 est illustré par la figure 1.23. Il est égal à $-0,29 + 0,08 + 1,32 + 1,32 + 1,01 + 0,08 + 0,08$ soit $3,60$.*

A	-0,29	0,08	-2,49	-2,49	-0,88	-0,29	-2,49
Con	-0,88	-0,88	-2,49	-2,49	1,01	-2,49	-2,49
G	0,73	0,73	1,32	1,32	-0,29	0,73	1,01
T	-0,29	-0,88	-2,49	-2,49	-2,49	0,08	0,08

FIG. 1.23: Calcul du score du mot **A A G G C T T** selon la matrice score-position de la figure 1.22

Une occurrence d'une matrice dans un texte est un mot de score supérieur ou égal à un score seuil choisi.

Définition 1.31 (Occurrence d'une matrice score-position) *Soit T un texte, M une matrice score-position de longueur L indexée par $\{0 \dots L-1\}$, α un score seuil, p une position $= 0 \dots |T| - L - 1\}$, T_p est une* occurrence *de M selon α dans T si et seulement si*

$$Score(M, T_p \dots T_{p+L-1}) \geq \alpha$$

Pour rappel, le problème de la localisation de matrices score-position dans un texte se définit comme suit.

Définition 1.32 (Problème de la localisation de matrices score-position) *Soit M une matrice score-position de longueur L indexée par $\{0 \dots L-1\}$ et T un texte. La recherche de M dans T est la localisation des occurrences de M dans T.*

Chapitre 2

Des mots aux matrices score-position

Sommaire

Nous avons vu dans le chapitre précédent l'importance et la formalisation du problème de la recherche de matrices score-position dans un texte, et du problème de la significativité statistique des résultats qui en découlent.

L'objectif de cette thèse étant de proposer des solutions efficaces à ces problèmes, notre démarche a été de nous inspirer des algorithmes sur les motifs exacts. En effet, ceux-ci sont les modèles de motifs les plus anciens, les plus étudiés et ayant les meilleures propriétés. Ces qualités leur valent de nombreux algorithmes efficaces. C'est pourquoi nous les présentons afin de proposer dans les chapitres suivants des extensions pour certains d'entre-eux.

Le chapitre s'organise autour de la présentation des algorithmes de recherche de mots exacts (section 2.1), de recherche de matrices score-position (section 2.2) et de calcul de la significativité statistique des résultats (section 2.3).

2.1 Algorithmes de recherche de motifs exacts dans un texte

Le problème de la recherche de mots exacts dans un texte fait partie des problèmes informatiques les plus anciens et les plus étudiés. Pour rappel, il est défini comme suit.

Définition 2.1 (Problème de la recherche d'un motif exact) *Soient $T = T_0 \dots T_{n-1}$ un texte de longueur n et $M = M_0 \dots M_{m-1}$ un motif de longueur m. Le problème de la recherche du motif M dans le texte T est de trouver toutes les occurrences de M dans T.*

Il existe de nombreux algorithmes efficaces qui résolvent ce problème. Les premiers créés dans les années 70 sont aussi les plus connus [18, 28]. Depuis, la recherche de motifs est devenue un champ de recherche à part entière et a donné naissance à un éventail d'algorithmes simples et efficaces. Comme point de départ, l'algorithme naïf opère en faisant glisser le long du texte une fenêtre de même longueur que le motif, et y recherche le motif position par position (voir Figure 2.1). Il existe trois grandes manières d'améliorer cet algorithme : accélérer les opérations de comparaison, pré-traiter le texte ou pré-traiter le motif.

Nous présentons dans cette section des algorithmes pour ces trois approches, mais détaillons plus amplement la dernière approche : le pré-traitement du motif.

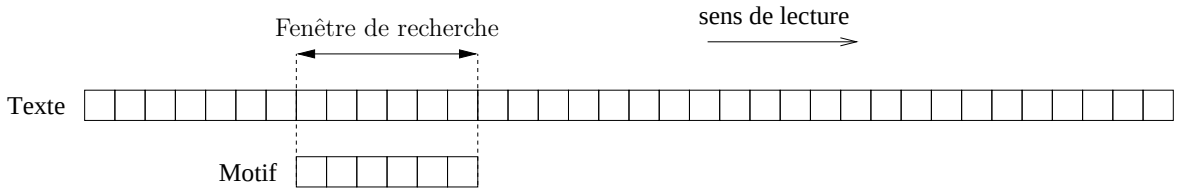

FIG. 2.1: La recherche s'effectue dans une fenêtre de la taille du motif glissant le long du texte

2.1.1 L'algorithme naïf

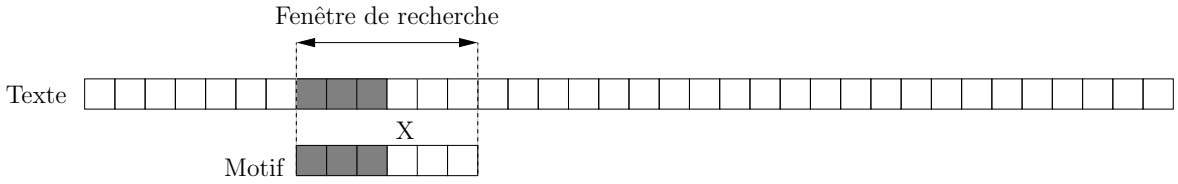

FIG. 2.2: La fenêtre est comparée au motif jusqu'à la rencontre d'une position d'erreur (marquée X) ou jusqu'à la fin du motif.

L'algorithme naïf consiste à balayer tout le texte position par position avec la fenêtre de recherche. À chaque position, la fenêtre est comparée au motif jusqu'à la rencontre d'une position d'erreur, ou jusqu'à la fin du motif (Figure 2.2). La complexité de cette algorithme est en $O(nm)$.

Définition 2.2 (Fenêtre) *Soient T un texte de longueur n et M un motif de longueur m. La* fenêtre F_i, $0 \leq i \leq n-m$ *est le mot* $T_i..T_{i+m-1}$.

Définition 2.3 (Tentative) *Soient T un texte de longueur n, M un motif de longueur m, on appelle* tentative à une position i, $0 \leq i \leq n-m$, *la comparaison entre la fenêtre F_i et le motif.*

2.1.2 Accélération des opérations de comparaison

L'opération de comparaison de deux lettres est l'opération de base des algorithmes de recherche de motifs. Une méthode consiste à transformer les opérations de comparaison en opérations numériques. Nous présentons dans cette partie l'algorithme de Karp et Rabin, basé sur une approche par hachage [62] et les algorithmes shift-or, shift-and, basés sur les vecteurs binaires [7].

Algorithme de Karp et Rabin

L'idée de base de cet algorithme est de coder chaque mot par un entier. Pour cela, on utilise une fonction bijective de hachage h qui à chaque mot de Σ^* associe un entier en base $b = |\Sigma|$. Soit $u = u_0 \dots u_{m-1}$ un mot de Σ^m et t une fonction bijective qui associe à chaque lettre de Σ un entier dans $\{0, \dots, |\Sigma| - 1\}$:

$$h(u) = t(u_0)b^{m-1} + t(u_1)b^{m-2} \dots t(u_{m-2})b + t(u_{m-1})$$

Ainsi on peut traduire par un entier le motif M de longueur m et chaque fenêtre de recherche F_i de longueur m correspondant à chaque position i de $\{0 \dots n-m\}$ du texte T de longueur n. Lorsque $h(F_i) = h(M)$ alors i est une occurrence de M dans T. Lors du parcours du texte par la fenêtre de recherche, le calcul de $h(F_{i+1})$ est obtenu à partir de $h(F_i)$ avec des opérations arithmétiques élémentaires comme suit :

$$h(F_{i+1}) = (h(F_i) - t(T_i)b^{m-1})b + T_{i+m}$$

Dans cette formule, la taille des entiers codant peut être limitante pour de grands alphabets et/ou de longs mots. Ainsi, la fonction de hachage est-elle modifiée en h' : $h'(u) = h(u) \mod q$, où q est le plus grand entier autorisé. L'algorithme devient un filtre. Si i est une occurrence, alors $h'(F_i) = h'(M)$. Mais si $h'(F_i) = h'(M)$ alors i n'est pas forcément une occurrence. Lorsque $h'(F_i) = h'(M)$ il faut donc vérifier F_i lettre par lettre. La complexité de cet algorithme dans le pire des cas reste en $O(mn)$ et il s'étend facilement à un ensemble de mots.

Algorithmes Shift-and et Shift-or

Le principe des algorithmes shift-and et shift-or est de simuler un automate de reconnaissance du motif. Pour cela, les deux algorithmes utilisent un vecteur binaire D de même longueur que le motif qui est mis à jour à chaque lettre lue dans le texte. Pour une tentative

i, un bit est introduit en première position du vecteur, qui indique si la première lettre de la fenêtre F_i est égale à la première lettre du motif. Ce bit est ensuite propagé tant que le motif est reconnu. L'efficacité de ces algorithmes vient de la maniabilité de la structure de données.

Algorithme Shift-and

Le vecteur binaire D_i de longueur m est défini comme suit :

$$D_i[j] = \begin{cases} 1 & \text{si } M_0 \dots M_j = T_{i-j+1} \dots T_i, \\ 0 & \text{sinon.} \end{cases}$$

où j, $0 \leq j \leq m-1$, est une position dans le motif.

Pour calculer D_{i+1} à partir de D_i, on introduit un vecteur de bits par lettre de l'alphabet. Pour tout x de Σ, on définit B_x de longueur m qui indique les positions d'occurrence de la lettre x dans le motif :

$$B_x[j] = \begin{cases} 1 & \text{si } M_j = x, \\ 0 & \text{sinon.} \end{cases}$$

Au départ $D_{-1} = 0^m$, et lorsqu'une nouvelle lettre T_{i+1} du texte est lue, le vecteur D_i est mis à jour et devient le vecteur D_{i+1} grâce à la règle suivante :

$$D_{i+1} = (D_i << 1 \ \vee \ 0^{m-1}1) \ \wedge \ B_{T_{i+1}}$$

où $<<$, $\wedge$ et $\vee$ les opérateurs binaires *shift*, *et logique* et *ou logique*. Cela a pour conséquence de propager les 1 si la lettre introduite est la première du motif. Les valeurs de i telles que $D_i[m-1] = 1$ indiquent que $i-j+1$ est une occurrence.

Algorithme Shift-or

L'algorithme Shift-or est juste une implémentation optimisée du Shift-and. Le vecteur D_i est défini en inversant les rôles de 0 et 1 :

$$D_i[j] = \begin{cases} 0 & \text{si } M_0 \dots M_j = T_{i-j+1} \dots T_i, \\ 1 & \text{sinon.} \end{cases}$$

$B_l[j]$ est également défini en inversant les rôles de 0 et 1 :

$$B_l[j] = \begin{cases} 0 & \text{si } M_j = l, \\ 1 & \text{sinon.} \end{cases}$$

Le calcul de D_{i+1} s'exprime alors plus simplement :

$$\begin{aligned} D_{-1} &= 1^m \\ D_{i+1} &= D_i << 1 \ \wedge \ B_{T_{i+1}} \end{aligned}$$

Lorsque $D_i[m-1] = 0$, $i-j+1$ est une occurrence.

Ces algorithmes sont très efficaces. Leur temps d'exécution est en $O(n)$ lorsque m est inférieur à la longueur d'un mot machine.

Le shift-or ne s'adapte pas au cas de la recherche simultanée de plusieurs motifs mais le shift-and très facilement. Soient k motifs $M^0, \dots M^{k-1}$ de longueur respective $m0, \dots m_{k-1}$. Le motif M utilisé est la concaténation des k motifs, les D_i se calculent alors comme suit

$$D_{i+1} = (D_i << 1 \ \vee \ 0^{m_0-1}10^{m_1-1}1 \dots 0^{m_k-1}1) \ \wedge \ B_{T_{i+1}}$$

et il y a occurrence du $j^{\text{ème}}$ motif, $0 \leq j \leq k-1$, lorsque $D_i[m_j - 1] = 1$.

2.1.3 Pré-traitement du texte

Indexation

Une solution pour améliorer l'approche naïve consiste à indexer à l'avance tous les motifs contenus dans le texte. Il suffit alors de consulter ce "dictionnaire" pour trouver les occurrences du motif.

Il existe plusieurs manière de construire un index de taille linéaire contenant tous les facteurs d'un texte. La première est l'arbre des suffixes [83]. L'arbre des suffixes indexe tous les suffixes du texte concaténé avec une lettre spéciale qui n'est pas dans l'alphabet du texte, notons-là \$. Une feuille de l'arbre est un suffixe du texte et un chemin dans l'arbre est un préfixe d'un suffixe du texte, c'est-à-dire un facteur. Les feuilles de l'arbre sont étiquetées par la liste des positions du texte dont le suffixe est occurrence, et les branches par une lettre ou un mot. La lettre \$ évite que certains suffixes se terminent sur un noeud interne de l'arbre. D'autres structures, plus compactes, sont l'automate des suffixes [14], le vecteur de suffixes [56] ou le tableau des suffixes [52].

L'algorithme de recherche basé sur un index est très simple. Il suffit de lire le motif dans la structure. La recherche s'effectue alors en $O(m)$, une fois que l'index a été construit. Cependant l'espace mémoire reste un problème pour les textes longs.

Compression

Pré-traiter un texte peut aussi se faire grâce à la compression de celui-ci. Nous présentons deux algorithmes de compression : l'encodage run-length (RLE) [65] et la compression Lempel-Ziv (LZ78) [49].

Le RLE encode s occurrences de la lettre l par le couple (s, l). La première phase de l'algorithme encode le texte avec le RLE et pré-calcule M' la matrice $m \times |\Sigma|$, l'élément de M' d'indice i de $\{0 \dots m-1\}$ et l de Σ noté $M'(i, l)$ est défini par

$$M'(i, l) = \sum_{t=i}^{m-1} M(t, l)$$

La phase de recherche consiste à parcourir le texte avec une fenêtre de recherche de longueur m. Lorsque le couple (s, l) correspond aux positions $i \dots i+s-1$ de la fenêtre, sa contribution au score de la fenêtre est

$$M'(i, l) - M'(i + s, l)$$

et se calcule en $O(1)$. La complexité de cette algorithme est en $O(\frac{nm}{s_{moy}})$ avec s_{moy} la moyenne des s.

Le LZ78 exploite les répétitions du texte et représente le texte comme une suite de blocs. Un bloc représente un facteur du texte et se compose d'un lien vers un facteur déjà encodé et d'une lettre. Par exemple le bloc représentant le facteur ul avec u de Σ^* et l de Σ se compose d'un lien vers u étiqueté par l et de l. Chaque bloc est donc une extension d'un autre déjà encodé et l'ensemble des bloc forme un arbre. La construction de l'arbre est linéaire. Pour encoder un facteur du texte on parcourt l'arbre depuis sa racine en suivant le chemin étiqueté par les lettres lues. Lorsque que le chemin ne peut plus être étendu on crée une nouvelle branche vers un nouveau bloc étiqueté de la dernière lettre lue. Le nouveau bloc étend son bloc parent d'une lettre. Après la création d'un nouveau bloc on repart de la racine. Le premier bloc crée correspond à la première lettre du texte et est lié à la racine de l'arbre. La première phase de l'algorithme est l'encodage LZ78 du texte et le pré-calcul, pour chaque bloc, de sa longueur et de sa contribution au score pour toutes ses positions possibles dans la fenêtre de recherche. La complexité de cet algorithme est en $O(\frac{mn}{\log(n)})$.

2.1.4 Pré-traitement du motif

Une autre manière d'améliorer l'approche naïve est d'essayer de décaler la fenêtre de recherche de plus d'une seule position à chaque étape. Le problème devient alors de faire des décalages plus importants avec l'assurance de trouver toutes les occurrences. Par un pré-traitement du motif, il est possible de prédire l'occurrence suivante potentielle du motif dans le texte, sans avoir à connaître le texte. L'avantage de cette méthode, comparée au pré-traitement du texte, est bien sûr le gain en espace que l'on peut espérer.

Les algorithmes présentés dans la suite de cette section s'appuient sur cette approche. Nous distinguons trois types de méthodes, qui dépendent de la manière de rechercher le motif dans la fenêtre. La première approche est dite préfixe, elle consiste à lire le plus long préfixe de la fenêtre aussi préfixe du motif. La deuxième est dite suffixe et consiste à lire le plus long suffixe de la fenêtre aussi suffixe du motif. Et la troisième est dite facteur et consiste à lire le plus long suffixe de la fenêtre aussi facteur du motif. Nous présentons des algorithmes des trois approches sélectionnées parmi les plus connues, les plus efficaces et/ou les plus utiles à la compréhension de cette thèse.

L'approche préfixe

L'approche par préfixe est la plus simple des méthodes de recherche de motifs utilisant le pré-traitement du motif pour effectuer des décalages de plus d'une position. La particularité de l'approche par préfixe est qu'elle permet de ne lire chaque lettre du texte qu'une seule fois (voir Figure 2.3).

L'algorithme le plus connu basé sur cette approche est l'algorithme de Knuth-Morris-Pratt, appelé le KMP [18]. C'est un enrichissement de l'algorithme de Morris-Pratt [39], certainement le premier algorithme efficace de recherche de motif.

Algorithme de Morris et Pratt

Lorsqu'une tentative à une position i du texte échoue à la lecture de la position j du motif alors on connaît le mot $T_i \dots T_{i+j-1}$, c'est le préfixe $M_0 \dots M_{j-1}$ du motif. L'idée première de

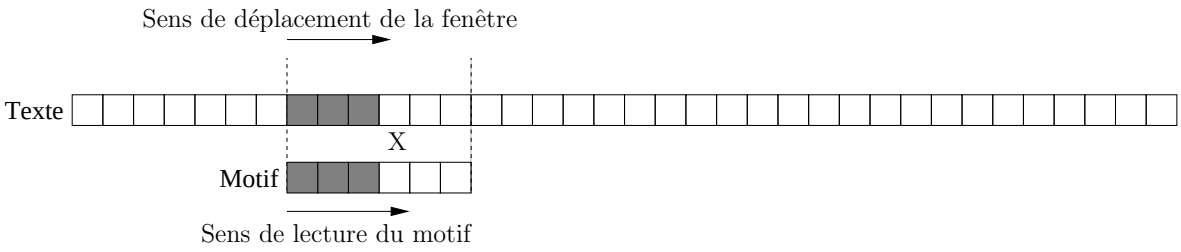

FIG. 2.3: L'approche par préfixe consiste à lire le plus long préfixe de la fenêtre aussi préfixe du motif.

Morris et Pratt est d'utiliser cette information pour sauter des positions du texte qui n'ont aucune chance d'être occurrence. En effet, la prochaine position du texte susceptible d'être occurrence est la position de début du *plus long suffixe de* $T_i \dots T_{i+j-1}$ *aussi* préfixe du motif. La figure 2.4 illustre la recherche de motif avec l'algorithme de Morris-Pratt.

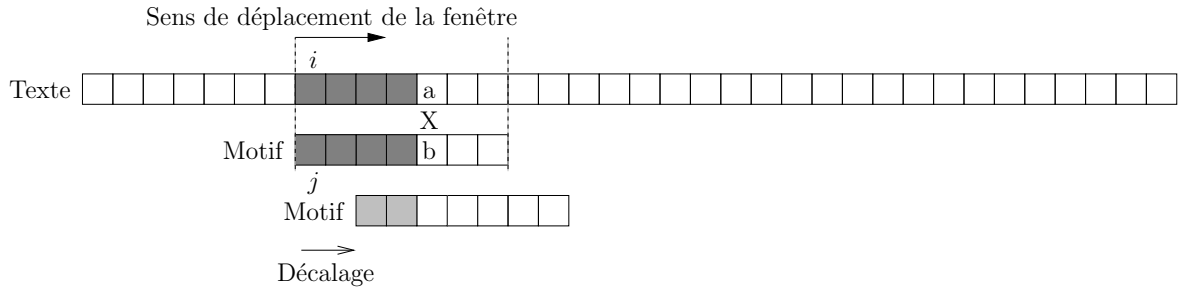

FIG. 2.4: La recherche avec l'algorithme de Morris-Pratt. On compare le motif et la fenêtre jusqu'à un échec (ou occurrence), partie gris foncée. On décale le motif de la position d'échec dans le motif à laquelle on soustrait la longueur du plus long suffixe également préfixe du motif, partie gris clair.

Le calcul du plus long préfixe-suffixe commun peut se faire par avance. On réalise un pré-traitement du motif qui consiste à pré-calculer la longueur de ce mot pour chaque position d'échec dans le motif, et indique le saut à effectuer dans le texte lors d'un échec. Ces pré-calculs sont stockés dans une *table des correspondances partielles*.

Afin de définir plus précisément le contenu de cette table, nous donnons la définition de *facteur propre* et *bord* d'un mot.

Définition 2.4 (Facteur propre) *Un facteur u d'un mot v est qualifié de* propre *si $u \neq v$*

Définition 2.5 (Bord) *Un* bord *d'un mot non vide u est un facteur propre de u qui est à la fois préfixe et suffixe de u. On note* `Bord`(u) *le plus long bord de u.*

Définition 2.6 (Table des correspondances partielles `mpNext`) *Soit $M = M_0 \dots M_{m-1}$ un motif de longueur m. La* table des correspondances partielles de Morris-Pratt

de M, notée `mpNext`*, est un vecteur de taille $m+1$ dont les éléments sont définis par :*

$$\begin{aligned} \texttt{mpNext}[0] &= -1 \\ \texttt{mpNext}[i] &= \texttt{Bord}(M_0 \dots M_{i-1}), \forall i \in [1, \dots, m] \end{aligned}$$

L'algorithme de recherche se découpe donc en deux phases : le calcul de la table des correspondances partielles, puis la recherche effective du motif dans le texte. Pendant la seconde phase, lorsqu'une tentative échoue à la position j du motif on décale la fenêtre de recherche de $j - \texttt{mpNext}[j]$ positions vers la droite et la comparaison reprend à partir de la position $\texttt{mpNext}[j]$ du motif.

La complexité de cette algorithme est en $O(m)$ dans le pire des cas et en moyenne pour la phase de pré-traitement du motif, et en $O(n)$ pour la phase de recherche.

Algorithme de Knuth-Morris-Pratt

De la même information que la tentative en position i du texte échoue à la position j du motif, on peut aussi déduire que la lettre T_{i+j} est différente de la lettre M_j. En résumé, on tire de l'événement d'échec les deux informations suivantes :

$$T_i \dots T_{i+j-1} = M_0 \dots M_{j-1}$$

$$T_{i+j} \neq M_j$$

Une illustration est donnée figure 2.5.

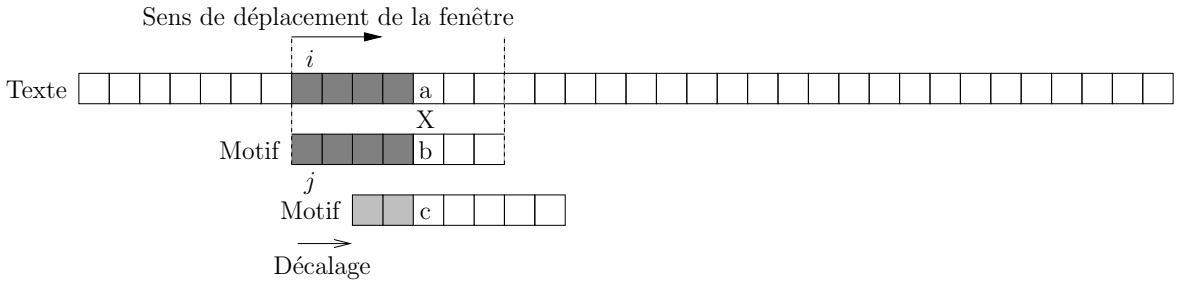

FIG. 2.5: La recherche avec l'algorithme de Knuth-Morris-Pratt. On compare le motif et la fenêtre jusqu'à un échec (ou occurrence), partie gris foncée. On décale le motif de la position d'échec dans le motif à laquelle on soustrait la longueur du plus long suffixe également préfixe du motif, partie gris-clair. On a de plus l'assurance que $b \neq c$.

Définition 2.7 (Bord étiqueté) *Soit $u = u_0..u_{m-1}$ un mot de Σ^m, x une lettre de Σ. Pour toute position l, $0 \leq l \leq m-1$, le préfixe $u_0..u_{l-1}$, est un* bord étiqueté *du mot ux, si :*

1. *c'est un bord de u, et*
2. *$u_l \neq x$*

On note $\texttt{Bord}_E(u, x)$ la longueur du plus long bord étiqueté de ux, si celui-ci existe.

Définition 2.8 (Table des correspondances partielles `kmpNext`) *Soit* $M = M_0 \dots M_{m-1}$ *un mot de longueur* m*. La table des correspondances partielles de Knuth-Morris-Pratt de* M*, notée* `kmpNext`*, est un vecteur de taille* $m+1$ *dont les éléments sont définis par :*

$$\begin{array}{rcl} \texttt{kmpNext}[0] & = & -1 \\ \texttt{kmpNext}[i] & = & \begin{cases} \texttt{Bord}_E(M_0..M_{i-1}, M_i), 0 < l < i & \textit{s'il existe,} \\ -1 & \textit{sinon} \end{cases} \quad \forall i \in [1, \dots, m-1] \\ \texttt{kmpNext}[m] & = & \texttt{Bord}(M) \end{array}$$

Algorithme de Aho-Corasick

Une extension de l'algorithme de Knuth-Morris-Pratt pour le cas d'un ensemble de mots, est l'algorithme de Aho-Corasick [32].

Cet algorithme utilise un automate construit sur l'ensemble de mots. Il s'agit d'un arbre des motifs augmenté de liens. Soit u_q le mot résultant du chemin de l'état initial à l'état q et $l(u)$ le plus long suffixe du mot u pouvant atteindre un état depuis l'état initial, c'est donc aussi le plus long suffixe de u préfixe d'un motif (extension de la notion de Bord à un ensemble de mots). Pour chaque état q de l'automate on ajoute un lien qui pointe vers l'état atteint par $l(u_q)$ depuis l'état initial. La phase de pré-traitement du motif est la construction de cet automate. La phase de recherche se réduit à passer le texte dans l'automate. Au départ, on se positionne sur l'état initial. A chaque lettre lue dans le texte, on essaie de passer par la transition de base (appartenant à l'arbre des motifs) correspondant à cette lettre et partant de l'état courant. Si elle n'existe pas on passe d'abord par la transition supplémentaire partant de l'état courant si elle existe et par l'état initial sinon. A chaque passage dans un état final il y a occurrence. La complexité de l'algorithme de recherche est en $O(n+k)$ avec k le nombre d'occurrences.

L'approche suffixe

L'approche par suffixe lit le motif et la fenêtre de recherche de droite à gauche (voir Figure 2.6). Pour illustrer cette approche nous présentons l'algorithme de Boyer-Moore [28].

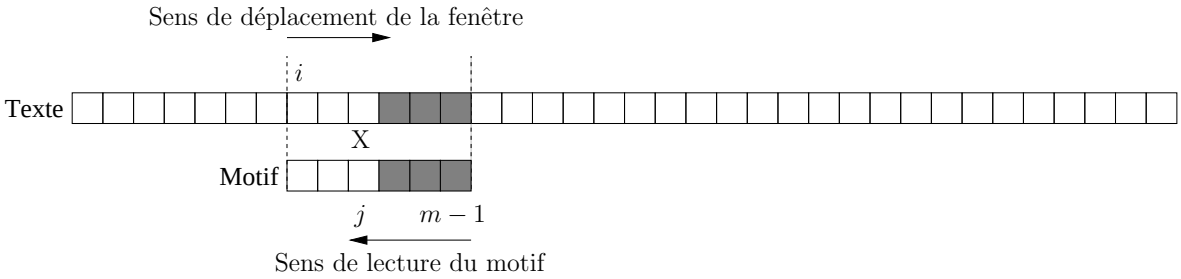

FIG. 2.6: L'approche par suffixe consiste à lire le plus long suffixe de la fenêtre aussi suffixe du motif.

Algorithme de Boyer-Moore

Cet algorithme a la particularité de pré-calculer deux tables afin de décider du décalage à effectuer lors d'une tentative qui échoue. Les deux tables contiennent des informations différentes et complémentaires.

Les informations contenues dans l'événement *la tentative à la position i du texte a échoué à la position j du motif* et utilisées par l'algorithme sont les suivantes.

$$T_{i+j+1} \dots T_{i+m-1} = M_{j+1} \dots M_{m-1} \tag{2.1}$$

$$T_{i+j} \neq M_j \tag{2.2}$$

$$T_{i+j} = x \tag{2.3}$$

où x est la lettre lue à la position d'échec. Cet algorithme tire, d'une tentative échouée, le même type d'information que l'algorithme KMP (Formules 2.1 et 2.2). Il retient en plus la lettre ayant provoquée l'échec (Formule 2.3).

La première table, appelée *table du bon suffixe*, donne le décalage à effectuer si on ne prend en compte que les informations 2.1 et 2.2. On recherche l'occurrence la plus à droite du suffixe qui vient d'être lu, et non précédée de la lettre qui a créé l'échec.

Définition 2.9 (Table du bon suffixe) *Soit $M = M_0 \dots M_{m-1}$ un mot de longueur m. La table du bon suffixe GS de M est un vecteur de longueur m tel que pour tout j de $\{0 \dots m-1\}$, l'élément noté $GS[j]$ est défini par*

$$\begin{aligned} GS[j] \quad = \quad \min\{l, \quad & M_{j+1} \dots M_{m-1} = M_{j+1-l} \dots M_{m-1-l} \quad \wedge \quad M_j \neq M_{j-l} \\ & \vee \quad M_l \dots M_{m-1} = M_0 \dots M_{m-1-l} \\ & \vee \quad l = m\} \end{aligned}$$

Le seconde table, appelée *table du mauvais caractère*, utilise l'information 2.3. Elle donne la longueur du plus long suffixe de M ne contenant pas x, mis à part en dernière position.

Définition 2.10 (Table du mauvais caractère) *Soit $M = M_0 \dots M_{m-1}$ un motif exact de longueur m, x une lettre de Σ et u un mot $\in \Sigma^*$. La table du mauvais caractère MC de M est un vecteur $|\Sigma|$ tel que pour tout $x \in \Sigma$ l'élément noté $MC[x]$ est défini par*

$$MC[x] = \max\{|u|, \ \forall k, 0 \leq k \leq |u| - 2 \quad u_k \neq x\}$$

La phase de pré-calcul de l'algorithme consiste à construire les tables du bon suffixe et du mauvais caractère. Pendant la phase de recherche, lorsqu'une tentative échoue à la position j du motif, l'idée est de choisir le plus grand des décalages proposés par les deux tables. On décale la fenêtre de recherche de $\max\{GS[j], MC[T_j]\}$ positions vers la droite.

L'approche facteur

Comme l'approche par suffixe, l'approche par facteur lit le motif et la fenêtre de recherche de droite à gauche, et décale la fenêtre vers la droite.

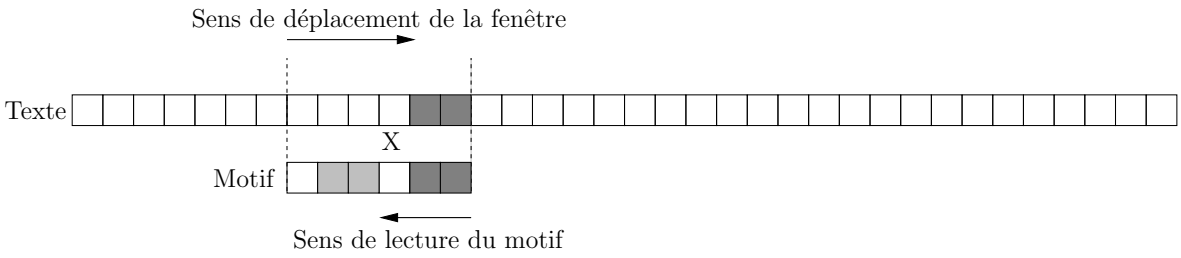

FIG. 2.7: L'approche par facteur consiste à lire le plus long suffixe de la fenêtre (en gris foncé) aussi facteur du motif (en gris clair).

L'idée principale de cette approche est de s'arrêter dès que le suffixe de la fenêtre de recherche n'est plus un facteur du motif puis de décaler la fenêtre juste après la dernière lettre lue. Nous présentons dans cette partie l'algorithme de recherche *Backward Dawg Matching*, qui utilise un automate des suffixes du mot.

Backward Dawg Matching

L'algorithme *Backward Dawg Matching* utilise les trois propriétés suivantes de l'automate des suffixes d'un motif exact.

1. L'automate détermine si un mot u est un facteur du motif en $O(|u|)$;
2. Il reconnaît les suffixes du motif par un état final ;
3. Il se construit en $O(m)$.

Le pré-traitement du motif consiste à construire l'automate des suffixes du motif lu de droite à gauche. Lors de la recherche, dès qu'un état final est rencontré, la position est stockée dans une variable *fin*. Cette position stockée correspond au début du plus long suffixe de la fenêtre de recherche aussi préfixe du motif.

Les deux cas d'arrêt de la recherche sont les suivants :
- l'état de la dernière lettre de la fenêtre lue n'a pas de transition, le suffixe de la fenêtre de recherche n'est donc plus un facteur du motif,
- une occurrence du motif est reconnue.

Dans les deux cas on décale la fenêtre de recherche pour qu'elle commence à la position *fin*.

Cet algorithme est en $O(mn)$ dans le pire des cas, mais a une complexité moyenne optimale : $O(\frac{n \log_{|\Sigma|} m}{m})$. Il s'étend directement à la recherche multiple et l'idée devient alors de rechercher dans la fenêtre le plus long suffixe aussi facteur d'un des motifs.

2.2 Algorithmes de recherche de matrices score-position dans un texte

Nous nous intéressons ici au problème de la recherche d'une matrice score-position dans un texte non pondéré. Pour rappel les matrices score-position sont des motifs pondérés qui associent un score à chaque lettre selon sa position dans le motif. Nous présentons le problème et l'algorithme naïf, puis trois méthodes de recherche utilisant des techniques différentes. La

première est basée sur le pré-calcul de l'ensemble des mots pouvant être occurrence du motif, la seconde sur une accélération du calcul des scores, la dernière sur le pré-traitement du texte.

Définition 2.11 (Tranche de matrice) *Soit M une matrice score-position de longueur m. Une* tranche de la matrice M entre i et j, $0 \leq i, j \leq m-1$, notée $M[i..j]$ est définie comme la matrice composée des colonnes i à j incluses. Si $j < i$ la matrice est vide.

Par la propriété d'additivité du score de la définition 1.30, on a les propriétés suivantes :

Lemme 2.1 *Soit M une matrice score-position de longueur m, p un entier de $\{1 \dots m\}$, pour tout mot u de Σ^p, pour tout mot v de Σ^p et tout mot w de $\in \Sigma^{m-p}$ on a :*

$$\begin{array}{rcl} \texttt{Score}(M[0 \dots p-1], u) = \texttt{Score}(M[0 \dots p-1], v) & \Rightarrow & \texttt{Score}(M, uw) = \texttt{Score}(M, vw) \\ \texttt{Score}(M[m-p \dots m-1], u) = \texttt{Score}(M[m-p \dots m-1], v) & \Rightarrow & \texttt{Score}(M, wu) = \texttt{Score}(M, wv) \end{array}$$

2.2.1 Algorithme naïf et amélioration de Wu *et al.*

L'algorithme naïf de recherche d'une matrice score-position dans un texte consiste à tester toutes les positions du texte, et pour chaque position calculer le score de la fenêtre du texte associée. Sa complexité est en $\Theta(mn)$. Il existe de nombreux logiciels utilisant l'algorithme naïf pour répondre à ce problème : FingerPrintScan [69], Blimps [43], Matinspector [59], Patser [44], Match [1].

Une première amélioration de cet algorithme, proposée par Wu *et al.* [86] et implantée dans leur logiciel Ematrix, consiste à arrêter le test d'une position du texte dès qu'elle n'a plus aucune chance d'être occurrence. Lorsque le score d'un préfixe de longueur $p+1$ d'un mot u est inférieur à un certain score $\texttt{BI}(M, p, \alpha)$ alors, quelque soit son suffixe correspondant, le score de u sera nécessairement inférieur à α. $\texttt{BI}(M, p, \alpha)$ représente le score minimum à atteindre en position p pour espérer avoir une occurrence pour le seuil fixé α.

Définition 2.12 (Score maximum) *Soit M une matrice score-position de longueur m, et $M[i..j]$ une tranche de la matrice M, le* score maximum de $M[i..j]$, noté $\texttt{ScoreMax}(M[i..j])$ est défini par :

$$\texttt{ScoreMax}(M[i..j]) = \sum_{k=i}^{j} \max_{x \in \Sigma} M(k, x)$$

Définition 2.13 (Score intermédiaire minimum) *Soit M une matrice score-position de longueur m, α un score seuil et p un entier de $\{0 \dots m-1\}$. Le* score intermédiaire minimum*, ou la* borne inférieure*, du préfixe de longueur $p+1$, noté $\texttt{BI}(M, p, \alpha)$, est défini par*

$$\texttt{BI}(M, p, \alpha) = \alpha - \texttt{ScoreMax}(M[p+1..m-1])$$

Symétriquement, on peut définir un autre score intermédiaire $\texttt{BS}(M, p, \alpha)$ tel que, lorsque le score d'un préfixe de longueur $p+1$ d'un mot u est supérieur ou égal à $\texttt{BS}(M, p, \alpha)$ alors, quelque soit son suffixe correspondant, le score de u sera supérieur à α.

Définition 2.14 (Score minimum) *Soit M une matrice score-position de longueur m, et $M[i..j]$ une tranche de la matrice M, le* score maximum de $M[i..j]$, noté $\texttt{ScoreMin}(M[i..j])$ est défini par :

$$\texttt{ScoreMin}(M[i..j]) = \sum_{k=i}^{j} \min_{x \in \Sigma} M(k,x)$$

Définition 2.15 (Score intermédiaire maximum) *Soit M une matrice score-position de longueur m, α un score seuil et p un entier de $\in \{0 \dots m-1\}$. Le* score intermédiaire maximum*, ou la* borne supérieure*, noté $\texttt{BS}(M,p,\alpha)$, est défini par*

$$\texttt{BS}(M,p,\alpha) = \alpha - \texttt{ScoreMin}(M[p+1..m-1])$$

Lemme 2.2 *Soit M une matrice score-position de longueur m indéxée de 0 à $m-1$ et α un score seuil, pour tout mot $u = u_0 \dots u_{m-1}$ sur Σ^m et toute position p de $\{0 \dots m-1\}$*

$$\begin{array}{lcl} \texttt{Score}(M, u_0..u_p) < \texttt{BI}(M,p,\alpha) & \Leftrightarrow & u \text{ n'est pas une occurrence} \\ \texttt{Score}(M, u_0..u_p) \geq \texttt{BS}(M,p,\alpha) & \Leftrightarrow & u \text{ est une occurrence} \end{array}$$

Preuve : Soit M une matrice score-position de longueur m, une position p de $\{0 \dots m-1\}$ et u un mot de Σ^m. D'après la Définition 1.30 $\texttt{Score}(M,u) = \texttt{Score}(M[0..p], u_0..u_p) + \texttt{Score}(M[p+1..m-1], u_{p+1}..u_{m-1})$*. u est une occurrence équivaut à* $\texttt{Score}(M,u) \geq \alpha$*, or*

$$\begin{array}{rcl} \texttt{Score}(M[0..p], u_0..u_p) & \geq & \alpha - \texttt{Score}(M[p+1..m-1], u_{p+1}..u_{m-1}) \\ & \geq & \alpha - \texttt{ScoreMin}(M[p+1..m-1] \\ & \geq & \texttt{BS}(M,p,\alpha) \end{array}$$

donc la seconde équivalence est prouvée. Il en va de même pour la première. ◄

En moyenne, l'algorithme de Wu *et al.* est en $O(kn)$, où k est le nombre moyen de positions de la matrice nécessaires lors d'une tentative. Dans le pire des cas la complexité est en $O(mn)$. En pratique, la mise en oeuvre de la propriété 1 du lemme 2.2 se révèle très efficace.

2.2.2 Enumération d'un ensemble de mots de la matrice

Pour aller plus loin, une première idée est de rechercher dans un texte l'ensemble des mots acceptés par la matrice selon le seuil donné. Ce type d'algorithme est donc composé de deux phases. La première consiste à extraire de la matrice l'ensemble des mots qu'elle reconnaît pour un seuil donné. La seconde est la recherche de l'ensemble de mots obtenus. Nous présentons dans la section suivante différents algorithmes d'extraction des mots d'une matrice pour un seuil donné. Les différents algorithmes de recherche multiple (voir section 2.1) peuvent prendre en charge la deuxième phase. Une méthode de ce type s'appuyant sur l'algorithme d'Aho-Corasick vient d'être proposée [16].

Concernant l'énumération des mots de score supérieur à α, l'approche naïve consiste à énumérer l'ensemble des mots de longueur m, de calculer leur score et de ne garder que les mots de score adéquat. La complexité de cet algorithme est en $O(|\Sigma|^m)$.

Une première amélioration a été proposée par Hertzberg [48]. On part du mot de meilleur score, puis on le dégénère successivement sur chacune de ses positions Dégénérer un mot $u = u_0 \dots u_{m-1}$ sur chaque de ses positions consiste à générer pour $i \in \{0 \dots m-1\}$ l'ensemble des mots $v = v_0 \dots v_{m-1}$ tel que $v_j = u_j$ pour $j \neq i$ et v_i correspond à la lettre de meilleur score suivant celui de u_i. Ainsi on obtient m mots ne différant de u que d'une seule position. On ne garde que les mots de score supérieur ou égal à α (on sait que les autres ne pourront jamais avoir un score supérieur à α grâce au lemme 2.1. On itère ensuite jusqu'à ce que l'ensemble des nouveaux mots de score supérieur à α soit vide. Remarquons que cette méthode n'est pas optimale car elle peut générer plusieurs fois les mêmes mots.

Une seconde amélioration a été proposée par Zhang *et al.* [87]. La méthode s'appuie sur les scores intermédiaires présentés plus haut. On note $x_{i,j}$, $0 \leq i \leq m-1$, $0 \leq j < |\Sigma|$, la lettre de la $i^{\text{ème}}$ colonne possédant le$j^{\text{ème}}$ score par ordre décroissant. On appellera j le rang. $x_{0,0}$ est donc la lettre de score le plus élevé à la position 0. A partir du mot de score maximum

$$m_{0,0}m_{1,0} \dots m_{m-1,0}$$

on énumère par ordre lexicographique des rangs tous les mots de longueur m. Le mot suivant le mot

$$m_{0,r_0}m_{1,r_1} \dots m_{m-1,r_{m-1}}$$

est

$$m_{0,r_0}m_{1,r_1} \dots m_{i,r_i+1}m_{i+1,0} \dots m_{m-1,0}$$

avec i la plus grande position tel que $r_i \neq |\Sigma| - 1$. Lorsqu'un mot

$$m_{0,r_0}m_{1,r_1} \dots m_{m-1,r_{m-1}}$$

a un score inférieur à α alors on saute tous les mots

$$m_{0,r_0}m_{1,r_1} \dots m_{k,r_k} \dots m_{k+1,*} \dots m_{m-1,*}$$

tel que k est la plus grande position avec $r_k \neq 0$. Le mot suivant est alors

$$m_{0,r_0}m_{1,r_1} \dots m_{i,r_i+1}m_{i+1,0} \dots m_{m-1,0}$$

avec i la plus grande position inférieure à k tel que $r_i \neq |\Sigma| - 1$. Avec cette méthode chaque mot est considéré au plus une fois.

2.2.3 Accélération des opérations de calcul

Une autre manière d'améliorer l'approche naïve est de traiter les opérations de calcul des scores plus rapidement. C'est ce que propose l'algorithme de Rajasekaran *et al.* [60], qui utilise les Transformées de Fourrier Rapides (TFR). L'objectif est d'obtenir un vecteur contenant les scores des alignements suivants : T_0 avec M_{m-1}, $T_{0\dots1}$ avec $M_{m-2} \dots M_{m-1}$, $\dots$, T_{n-1} avec M_0. L'idée est de partir des vecteurs indicateurs du texte et de la matrice pour chaque lettre de l'alphabet, de les transformer avec une TFR et de les sommer dans un vecteur qui subira une TFR inverse.

Soit x et y deux lettres de Σ, I la matrice identité sur $\Sigma \times \Sigma$ définie comme suit :

$$I(l,l') = \begin{cases} 1 & \text{si } l = l' \\ 0 & \text{sinon} \end{cases}$$

$x_l = (I(l,T_0), I(l,T_1), \dots I(l,T_{n-1}))$ le vecteur indicateur de la lettre l dans le texte T,
$y_l = (M(0,l), M(1,l), \dots M(m-1,l))$ le vecteur correspondant à une ligne de la matrice M,
$z_l = (z_{l,0}, z_{l,1}, \dots z_{l,n+m-2})$ le vecteur correspondant à la corrélation croisée de x_l et y_l égal aux scores attribuables à la lettre l, l'élément d'indices j de z_l noté $z_{l,j}$ est défini par

$$z_{l,j} = \sum_{i=\max(0,j-m+1)}^{\min(n-1,j)} I(l,T_i) M(i+m-1-j,l)$$

z le vecteur des scores finaux, l'élément d'indices j de z noté z_j est défini par

$$z_j = \begin{array}{l} \sum_{l\in\Sigma} z_{l,j} \\ \sum_{l\in\Sigma} \sum_{i=\max(0,j-m+1)}^{\min(n-1,j)} I(l,T_i) M(i+m-1-j,l) \\ \sum_{i=\max(0,j-m+1)}^{\min(n-1,j)} M(i+m-1-j,T_i) \end{array}$$

L'algorithme de base consiste, pour toute lettre $l \in \Sigma$, à calculer
- les vecteurs x_l et y_l
- les vecteurs z_l, corrélations croisées des vecteurs x_l et y_l
- le vecteur z somme des vecteur z_l

L'implantation de cet algorithme avec les TFR consiste, pour toute lettre $l \in \Sigma$, à calculer
- les vecteurs x_l et y_l
- les vecteurs Z_l transformés de fourrier des vecteur z_l avec une TFR, l'élément d'indices j de Z noté Z_j est défini par

$$Z_j = \sum_{i=0}^{n-m-2} z_{l,i} wij$$

  avec $w = e^{\frac{2\pi i}{n+m-2}}$ la $n+m-2^{\text{ème}}$ racine complexe de l'unité
- le vecteur Z somme des vecteur Z_l
- le vecteur z transformé de fourrier inverse du Z avec une TFR inversé

Cet algorithme est en $O(|\Sigma|(n+m)\ln(n+m))$

Il y a aussi une variante de cet algorithme utilisant le schéma de compression de Cheever *et al.* [22] qui est aussi en $O(|\Sigma|(n+m)\ln(n+m))$.

2.2.4 Pré-traitement du texte

Une autre méthode d'amélioration est de pré-traiter le texte, en l'indexant ou en le compressant.

Indexation du texte

Dorohonceanu *et al.* [5] sont les premiers à avoir proposé d'utiliser une structure d'index pour accélérer la recherche de matrices score-position. Leur algorithme est basé sur l'arbre des suffixes. Une fois l'arbre construit, on le parcourt depuis sa racine en calculant les scores des facteurs profondeur par profondeur. Afin d'élaguer certains sous-arbres, les scores intermédiaires `BI` et `BS` sont à nouveau utilisés. Lorsque le score du préfixe de facteurs est inférieur au score intermédiaire minimum, alors il est inutile de continuer le calcul des scores, les facteurs ne seront par des occurrences. Symétriquement, il est inutile de continuer le calcul dans les sous-arbres fils si le score du préfixe est supérieur au score intermédiaire maximum car les facteurs seront tous des occurrences. L'arbre des suffixes compact [53] est utilisé pour stocker les facteurs.

Beckstette *et al.* [9] proposent une amélioration de l'algorithme précédent en remplaçant l'arbre compact des suffixes par une table des suffixes [55]. Le principe reste le même. Les facteurs sont indexés par préfixe communs et parcourus par ordre lexicographique. Les sous-arbres des préfixes extérieurs à l'intervalle des scores intermédiaires sont "sautés".

Compression du texte

Freschi et Bogliolo [84] proposent deux algorithmes basés sur cette méthode, le premier utilise l'encodage run-length (RLE) [65] et le second la compression Lempel-Ziv (LZ78) [49].

2.3 Propriétés statistiques des occurrences

Dans cette section, nous nous intéressons aux propriétés statistiques des motifs à travers deux indicateurs : la significativité d'une occurrence, et la significativité du nombre d'occurrences. Dans les deux cas, l'évaluation se fait par le calcul de la P-valeur pour un modèle de génération aléatoire donné pour le texte. Comme nous l'avons vu au chapitre 1, ces P-valeurs donnent la probabilité d'observer le résultat obtenu dans le modèle. Elles permettent donc d'estimer le caractère exceptionnel des résultats.

2.3.1 P-valeur d'une occurrence

La P-valeur d'une occurrence d'un motif est sa probabilité d'être observée par hasard à une position donnée dans le modèle de texte. Nous présentons les calculs de la P-valeur pour un mot exact, un ensemble de mots exacts et une matrice score-position.

P-valeur d'un mot exact

Cette probabilité est facile à calculer. Il s'agit de la probabilité de génération d'un mot selon le modèle de texte choisi. Pour rappel, la formule de probabilité d'un mot est définie comme suit. Soit $u = u_0 \dots u_{m-1}$ un mot de Σ^m, μ_x la fréquence de la lettre x de Σ et $\Pi_{u,x}$ la probabilité $\mathbb{P}(x|u)$.

– selon $\mathcal{M}$ un modèle de Bernoulli

$$\mathbb{P}(u|\mathcal{M}) = \prod_{i=0}^{m-1} \mu_{u_i}$$

– selon $\mathcal{M}'$ un modèle de Markov d'ordre r

$$\mathbb{P}(u|\mathcal{M}') = [\sum_{v\in\Sigma^k} \Pi_{v,u_0}] * [\sum_{v\in\Sigma^{k-1}} \Pi_{vu_0,u_1}] * \cdots * \Pi_{u_{L-k-1}\ldots u_{L-2},u_{L-1}}$$

P-valeur d'un ensemble de mots

La probabilité d'avoir une occurrence d'un ensemble de mots de même longueurs à une position d'un texte selon un modèle de texte est la somme des probabilités de chacun des mots de l'ensemble. Soit E un ensemble de mots et $\mathcal{M}$ un modèle de texte.

$$\mathbb{P}(E) = \sum_{u\in E} \mathbb{P}(u|\mathcal{M}) \tag{2.4}$$

P-valeur d'une matrice score-position

Définition 2.16 (P-valeur d'une occurrence d'une matrice score-position) *Soient M une matrice score-position de longueur m et α un score seuil. Pour un modèle de fond donné, la* P-valeur *de M et α, notée* `P-valeur`(M,α) *est la probabilité de l'ensemble des mots u de Σ^m d'avoir un score supérieur ou égal à α :*

$$\texttt{P-valeur}(M,\alpha) = \sum_{u\in\Sigma^m} \mathbb{P}(score(M,u) \geq \alpha) \tag{2.5}$$

Une première approche du problème du calcul de la P-valeur d'une occurrence d'une matrice score-position et d'un score seuil α est d'extraire de la matrice l'ensemble des mots de score supérieur à α, à l'image des méthodes présentées en Section 2.2.2, puis de calculer la P-valeur de cet ensemble en utilisant l'Equation 2.4. Le nombre de mots à considérer, peut rendre toutefois le calcul rédhibitoire. Il est préférable de travailler directement sur la matrice, soit en estimant la distribution par une loi de probabilité usuelle, soit en construisant de manière exacte la distribution des scores. Nous détaillons ces deux solutions.

Estimation de la distribution de scores. La distribution des scores d'une matrice score-position a d'abord été estimée par une loi normale [75]. Elle fut ensuite approchée par une loi extrême, la loi de Gumbel. D'après Claverie et Audic [17] les Z-scores, les scores centrés et réduits, d'une matrice score-position suivent la loi de Gumbel suivante :

$$G(Z) = \frac{1}{\beta} e^{\frac{-(Z-\alpha)}{\beta}} e^{-e^{\frac{-(Z-\alpha)}{\beta}}}$$

avec $\alpha \cong -0.45$ et $\beta \cong 0.78$. Plus la matrice est longue, mieux elle s'ajuste à la loi de Gumbel.

Distribution exacte des score

Nous avons vu que la fonction de score d'une matrice score-position était additive (Lemme 2.1). Cette propriété fait que le score d'un mot s'exprime facilement en fonction du score de ses préfixes. En conséquence, les P-valeurs de la sous-matrice correspondant aux colonnes 0 à i se déduisent des P-valeurs de la sous-matrice correspondant aux colonnes 0 à $i-1$.

Soit M une matrice score-position de longueur m et α un score seuil. Si on suppose que le modèle de fond est un modèle de Bernoulli, on a

$$\begin{aligned} \texttt{P-valeur}(M[0\cdots-1],\alpha) &= \begin{cases} 1 & \text{si } \alpha \leq 0 \\ 0 & \text{sinon} \end{cases} \\ \texttt{P-valeur}(M[0\ldots i],\alpha) &= \sum_{x\in\Sigma} \texttt{P-valeur}(M[0\ldots i-1],\alpha - M(p,x)) \times \mu_x \end{aligned}$$

Cette formule a été proposée initialement par Staden [75]. Elle a été étendue aux modèles de Markov d'ordre 1 dans [86]. Huang [45] a ensuite donné la fonction génératrice d'une matrice score-position pour un modèle de fond modélisé par un modèle de Markov d'ordre quelconque.

L'implémentation se fait par programmation dynamique. Lors du calcul de la P-valeur d'un score seuil, il n'est pas nécessaire de calculer l'ensemble de la distribution des scores de la matrice. Seule la partie correspondant à la queue de la distribution, au delà du score seuil α est intéressante. La formule est implémentée naturellement par programmation dynamique. Pour chaque valeur de i, il est suffisant de considérer l'intervalle de score $[\texttt{BI}(M,i,\alpha)..\texttt{ScoreMax}(M,i,\alpha)]$.

Cette équation peut être mise en œuvre de manière efficace en utilisant la programmation dynamique descendante paresseuse. C'est ce qui a été proposé dans Beckstette *et al.* [9] et Malde *et al.* [51] pour un texte modélisé par un modèle de Bernoulli. Le principe est de n'évaluer et de ne mémoriser que les valeurs qui sont nécessaires. On calcule les P-valeurs des scores supérieurs ou égaux au score seuil, score par score en décroissant depuis le score maximum. Cette méthode propose aussi de permuter les colonnes de la matrice selon leurs contenu informationnel. Les colonnes les mieux conservées sont traitées en premier, ce qui maximise l'effet de l'approche paresseuse. Cela est correct car l'ordre des positions dans un modèle de Bernoulli n'a aucun impact sur la probabilité d'un mot, ni sur son score.

Enfin, une dernière optimisation a été proposée dans [81]. L'idée principale est d'utiliser une suite d'approximations successives pour la distribution de score, améliorant étape par étape le calcul de la P-valeur. Les approximations sont basées sur des discrétisations de plus en plus fines de la matrice score-position. Un critère de convergence garantit l'exactitude du résultat, même dans le cas de matrices à coefficient réel.

2.3.2 P-valeur d'un nombre d'occurrences

La P-valeur du nombre d'occurrences d'un motif dans un texte est la probabilité d'observer au moins ce nombres d'occurrence dans le modèle de texte. Afin de décider de l'exceptionnalité d'un nombre d'occurrences d'un motif dans un texte, il faut le comparer au nombre d'occurrences attendu. Cela nécessite de connaître la distribution du nombre d'occurrences du motif dans un texte. Par rapport au calcul de la P-valeur d'une occurrence, le calcul est compliqué par le fait qu'il faut prendre en compte les chevauchements possibles entre occurrences. Ces chevauchements dépendent naturellement de la structure du motif. On parle alors d'*auto-chevauchement*, qui peut se capturer avec le calcul d'un coefficient de corrélation. Ce problème a récemment fait l'objet de nombreuses investigations. Pour avoir une vue d'ensemble de ce problème, on peut consulter [42] et [67].

P-valeur du nombre d'occurrences d'un mot exact

Une approche de ce problème consiste à estimer la distribution du nombre d'occurrences par une loi approchée. Dans le cas d'un modèle de fond markovien, cette distribution est approchée par une loi gaussienne [6] lorsque l'espérance du comptage est grande, par une loi de Poisson composée [66] lorsque qu'au contraire l'espérance est faible. Une autre approche utilise les grandes déviations [58, 41]. Une formule permettant de calculer la loi exacte par programmation dynamique a été proposée par Robin et Daudin en 1999 [27] et par Zhang *et al* en 2007 [87].

P-valeur du nombre d'occurrences d'un ensemble de mots

Cette distribution est approchée par une loi normale [6] lorsque l'espérance est grande et par une loi de poisson composée [**?**] lorsque l'espérance est faible. Concernant le calcul de la distribution exacte, Régnier [61] a donné en 1998 la fonction génératrice et les formules des deux premiers moments, Hertzberg *et al.* [48] ont proposé en 2005 une formule et une méthode de calcul pour la probabilité d'avoir au moins une occurrence pour un ensemble de mots de même longueur et Zhang *et al.* [87] ont proposé en 2007 une formule de calcul de la distribution complète.

P-valeur du nombre d'occurrences d'une matrice score-position

Jusqu'à aujourd'hui, les méthodes de calculs proposées traitent le problème en considérant explicitement l'ensemble de mots associé [48, 87]. Le problème de déterminer directement la P-valeur du nombre d'occurrences d'une matrice score-position est ouvert.

Chapitre 3

Algorithmes à la Knuth-Morris-Pratt pour les matrices score-position

Sommaire

Nous avons présenté dans le premier chapitre le problème général de la recherche de matrices score-position dans un texte et dans le deuxième chapitre différentes solutions proposées dans la littérature.

Ce troisième chapitre est consacré à une solution originale à la recherche basée sur le pré-traitement du motif, dans l'esprit de Knuth-Morris-Pratt. Cette approche n'a pas encore été explorée dans le cadre des matrices, et s'est montrée très efficace sur les motifs exacts.

Dans l'algorithme de Knuth-Morris-Pratt, la phase de pré-traitement consiste à extraire du motif une information suffisante pour déterminer à quelle position du texte continuer la recherche après la conclusion d'une tentative, sans perte d'occurrence. L'idée est d'extraire de l'événement *fin d'une tentative* la longueur du prochain décalage de la fenêtre de recherche à partir de la connaissance de la séquence lue jusqu'à la position courante. Nous proposons d'étendre cette approche aux matrices, dans l'esprit de Morris-Pratt et Knuth-Morris-Pratt. Cela nous conduira à proposer deux *tables de correspondances partielles*, directement inspirées de celles proposées par les algorithmes sur les motifs exacts. Nous proposerons ensuite le calcul

d'une nouvelle table qui, comme nous le verrons, fournit de meilleurs résultats. La définition et le calcul de ces tables sont au cœur de ce chapitre.

3.1 Extension de la table de Morris et Pratt

Dans le cas des mots exacts, une tentative se termine dès qu'une lettre de la fenêtre de recherche ne correspond pas au motif ou lorsque la fin de la fenêtre de recherche est atteinte. Dans le cas des matrices score-position, une tentative se termine dès que le score sort de l'intervalle des scores intermédiaires, tel que défini au chapitre 2. Soit elle réussit car le score est supérieur à la borne supérieure de l'intervalle, soit elle échoue car le score est inférieur à la borne inférieure.

Lemme 3.1 *Soient M une matrice de longueur m, T un texte de longueur n, α un score seuil, i une position de M ($0 \leq i \leq m-1$), p une position de T ($0 \leq p \leq n-m$). Une tentative peut prendre fin à la position i de deux manières :*

- $\texttt{Score}(M[0..i], T_p..T_{p+i}) \geq \texttt{BS}(M, i, \alpha)$ *: dans ce cas, on sait que l'on a une occurrence de M à la position p du texte ;*
- $\texttt{Score}(M[0..i], T_p..T_{p+i}) < \texttt{BI}(M, i, \alpha)$ *: dans ce cas, on sait qu'il n'y a pas d'occurrence de M à la position p du texte.*

Preuve : C'est une conséquence du Lemme 2.2 introduit au chapitre 2. ◄

L'arrêt prématuré d'une tentative grâce à la condition sur `BS` arrive très rarement. Sur les expérimentations que nous avons menées, pour une P-valeur de 10^{-3}, l'arrêt grâce à `BS` arrive dans moins de 0.1% des cas alors que l'arrêt grâce à `BI` se produit dans plus de $99,9\%$ des cas. Par conséquent, dans la suite, nous développerons des méthodes qui ne considèrent comme cas d'arrêt que le cas d'échec, c'est-à-dire la condition sur `BI`.

3.1.1 Notion de bord d'une matrice

Définition 3.1 (Bord d'une matrice) *Soit M une matrice de longueur m, et soit α un score seuil. Un* bord de M et de α *est un mot u de longueur $\ell \geq 1$ tel qu'il existe deux mots v et w de longueur $m-\ell$ satisfaisant*

1. $\texttt{Score}(M, uv) \geq \alpha$, *et*
2. $\texttt{Score}(M, wu) \geq \alpha$.

On note $Bord(M, \alpha)$ la longueur du plus long bord de M et α.

On peut vérifier que cette définition coïncide bien avec la Définition 2.5 dans le cas d'une matrice modélisant un mot exact. Soit t, un mot de Σ^m. On lui associe la matrice M_t dont tous les coefficients sont nuls, sauf $M_t(t_i, i)$, qui est égal à 1. Les occurrences de M_t pour le score seuil m sont exactement les occurrences de t. Regardons maintenant les bords de M_t et m. D'après la Définition 3.1, un mot u de longueur ℓ est un bord de M_t et m si, et seulement si, il existe deux mots v et w tels que

1. $\texttt{Score}(M, uv) \geq m$, et

2. $\texttt{Score}(M, wu) \geq m$

Par construction de M_t, il s'ensuit que u est un bord de t, et réciproquement.

Nous énonçons maintenant une caractérisation équivalente, qui facilitera le calcul effectif de $Bord(M, \alpha)$.

Lemme 3.2 *Soit M une matrice de longueur m, et soit α un score seuil. Le mot u de longueur ℓ est un bord de M et de α si, et seulement si, u satisfait les deux conditions suivantes :*

1. $\texttt{Score}(M[0..\ell-1], u) \geq \texttt{BI}(M, \ell-1, \alpha)$, *et*
2. $\texttt{Score}(M[m-\ell..m-1], u) \geq \alpha - \texttt{ScoreMax}(M[0..m-\ell-1])$.

Preuve :

$\Leftarrow$) *Pour tout mot w de $\Sigma^{m-\ell}$, on a $\texttt{Score}(M[0..m-\ell-1], w) \geq \texttt{ScoreMin}(M[0..m-\ell-1])$. Par conséquent, $\texttt{Score}(M, wu) \geq \alpha - \texttt{ScoreMax}(M[0..m-\ell-1]) + \texttt{ScoreMin}(M[0..m-\ell-1])$ et donc $\texttt{Score}(M, wu) \geq \alpha$.*

Pour tout mot v de $\Sigma^{m-\ell}$, on a $\texttt{Score}(M[\ell..m-1], v) \geq \texttt{ScoreMin}(M[\ell..m-1])$. Par conséquent, $\texttt{Score}(M, uv) \geq BI(M, \ell-1, \alpha) + \texttt{ScoreMin}(M[\ell..m-1])$, et donc $\texttt{Score}(M, uv) \geq \alpha - \texttt{ScoreMax}(M[\ell..m-1]) + \texttt{ScoreMin}(M[\ell..m-1])$. Ainsi $\texttt{Score}(M, uv) \geq \alpha$.

En conséquence u est un bord de M.

$\Rightarrow$) *Soit v un mot de score $\texttt{ScoreMax}(M[\ell..m-1])$, w un mot de score $\texttt{ScoreMax}(M[0..m-\ell-1])$ et u un bord de M de longueur $\ell \geq 1$. On a $\texttt{Score}(M, uv) \geq \alpha$ et donc*

$$\begin{aligned} \texttt{Score}(M[0..\ell-1], u) &\geq \alpha - \texttt{Score}(M[\ell..m-1], v) \\ &\geq \alpha - \texttt{ScoreMax}(M[\ell..m-1]) \\ &\geq \texttt{BI}(M, \ell-1, \alpha) \end{aligned}$$

D'autre part, on a $\texttt{Score}(M, wu) \geq \alpha$ et donc

$$\begin{aligned} \texttt{Score}(M[\ell..m-1], u) &\geq \alpha - \texttt{Score}(M[0..m-\ell-1], w) \\ &\geq \alpha - \texttt{ScoreMax}(M[0..m-\ell-1]) \end{aligned}$$

Par conséquent les propriétés du lemme sont vérifiées.

◀

3.1.2 Table des correspondances partielles

Une fois la notion de bord établie pour une matrice, on peut définir le tableau des correspondances partielles, à l'image de ce qui a été présenté dans la Section 2.1.4.

Définition 3.2 (Table des correspondances partielles `mpNext`) *Soit M une matrice de longueur m, α un score seuil, La table des correspondances partielles de M et α, notée $mpNext$, est un vecteur de taille $m+1$ dont les éléments sont définis par*

$$\begin{aligned} mpNext[0] &= -1 \\ mpNnext[i] &= \texttt{Bord}(M[0..i-1], \texttt{BI}(M, i-1, \alpha)), \text{ pour tout } i \text{ de } \{1, \ldots, m\} \end{aligned}$$

Lemme 3.3 *Soit M une matrice de longueur m, T un texte de longueur n, p une position dans le texte comprise entre 0 et $n-m-1$, α un score seuil, et i une position dans la matrice ($0 \leq i \leq m-1$). Si la tentative p se termine à la position i de M, alors on connaît par avance le résultat des tentatives $p+1 \ldots p+(i-\texttt{mpNext}[i])$.*

Preuve : Supposons qu'il existe une position $p+k$, $1 \leq k \leq i-\texttt{mpNext}[i]$ telle que $T_{p+k}..T_{p+k+m-1}$ soit une occurrence. Alors $\texttt{Score}(M, T_{p+k}..T_{p+k+m-1}) \geq \alpha$ et donc $\texttt{Score}(M[0..i-k], T_{p+k}..T_{p+i-1}) \geq BI(M, i-k, \alpha)$. Or nécessairement $\texttt{Score}(M[k..i-1], T_{p+k}..T_{p+i-k}) < \alpha - \texttt{ScoreMax}(M[0..k])$ sinon $T_{p+k}..T_{p+i}$ serait un bord de $M[0..i-1]$, ce qui est impossible puisque $\texttt{mpNext}[i] < i-k+1$. Par conséquent, $\texttt{Score}(M[k..i-1], T_{p+k}..T_{p+i-k}) + \texttt{ScoreMax}(M[0..k]) < \alpha$ ce qui contredit le fait que la tentative se termine en position i. ◄

Par conséquent, si une tentative échoue à la position p du texte, alors la longueur du décalage du motif à effectuer sur le texte est $i - \texttt{mpNext}[i]$.

3.1.3 Phase de recherche d'une matrice score-position dans un texte

La phase de recherche des occurrences d'une matrice score-position dans un texte à partir d'une table des décalages de la matrice est tout à fait similaire à celle dans le cas des mots exacts. L'algorithme consiste à calculer les scores successifs des préfixes de la matrice à une position du texte. Dès que l'on se trouve dans un cas d'échec à une position i de la matrice, on consulte la table pour obtenir la prochaine position du texte à tester, obtenue en ajoutant à la position actuelle la valeur de table stockée à la position i. L'algorithme est donné Algorithme 1.

Dans les algorithmes de Morris-Pratt et de Knuth-Morris-Pratt, le décalage possède la propriété supplémentaire que, comme le préfixe (le bord) est exactement le texte, il est inutile de relire les lettres du texte correspondant à ce préfixe. Dans le cas des matrices score-position, on sait également que le préfixe est potentiellement une occurrence mais pour conclure il est nécessaire de connaître le score du préfixe, ce dont on ne dispose pas. Il est donc nécessaire de relire les lettres du texte pour calculer le score.

3.1.4 Calcul effectif des éléments de la table

Nous allons maintenant voir comment calculer la table des correspondances partielles. Comme pour le calcul de la P-valeur présenté au chapitre 2, on ne raisonne pas sur les mots eux-mêmes, mais sur leur score. En pratique, le nombre de scores distincts est en effet largement inférieur au nombre de mots distincts.

Définition 3.3 **($b(M,i,j,\beta,\gamma)$ et $c(M,i,j,\alpha)$)** *Soit M une matrice de longueur m, i,j deux positions de M ($0 \leq j \leq i \leq m-1$), et β,γ des réels. On définit le prédicat $b(M,i,j,\beta,\delta)$ par*

$$b(M,i,j,\beta,\delta) = \exists u \in \Sigma^{i-j+1} \quad (\texttt{Score}(M[j..i], u) = \beta \ \wedge \ \texttt{Score}(M[0..i-j], u) = \delta)$$

A partir de b, on définit $c(M,i,j,\alpha)$ par

$$c(M,i,j,\alpha) = \exists \beta \geq \texttt{BI}(M,i-1,\alpha) - \texttt{ScoreMax}(M[0..j-1]) \exists \delta \geq \texttt{BI}(M,i-j-1,\alpha) \ \ b(M,i-1,j,\beta,\delta)$$

Algorithme 1 : Recherche des occurrences avec une table de décalage

Entrée: T un texte de longueur n, M la matrice de longueur m, BI le tableau des score minimum intermédiaire, `next` la table des correspondances partielles
Sortie: les occurrences de M dans T

$j \leftarrow 0$
tant que $j < n - m + 1$ **faire**
  $i \leftarrow 0$
  $s \leftarrow 0$
  tant que $i < m$ and $s >= BI[i]$ **faire**
    $s \leftarrow s + M(i, T[j + i])$
    $i++$
  si $i = m$ **alors**
    print j
    $j++$
  sinon
    $j \leftarrow i -$ `next`$[i]$

Nous montrons maintenant que l'introduction de b et c permet bien de calculer les éléments de la table `mpNext`.

Lemme 3.4 *Si $c(M, i, j, \alpha')$ est vrai alors il existe un bord de longueur $i - j$ pour $M[0..i-1]$ et le score seuil* $\texttt{BI}(M, i-1, \alpha')$.

Preuve : $c(M, i, j, \alpha)$ est vrai implique qu'il existe un mot u de longueur $i - j$ tel que

1. $\texttt{Score}(M[j..i-1], u) \geq \texttt{BI}(M, i-1, \alpha') - \texttt{ScoreMax}(M[0..j-1])$
2. $\texttt{Score}(M[0..i-j-1], u) \geq \texttt{BI}(M, i-j-1, \alpha')$

d'après les définitions de c et b. En posant, $m = i$, $l = i - j$ et $\alpha = \texttt{BI}(M, i-1, \alpha')$, 1) et 2) expriment les deux conditions du lemme 3.2. Par conséquent u est un bord de longueur $i - j$ pour $M[0..i-1]$ et le score seuil $\texttt{BI}(M, i-1, \alpha')$. ◄

Lemme 3.5 *$mpNext[i]$ est égal à $i - j$, où j est la plus petite valeur telle que $c(M, i, j, \alpha)$ soit vrai.*

Preuve : Grâce au lemme 3.4 on sait qu'il existe un bord de longueur $i - j$ pour la matrice $M[0..i-1]$ et le score seuil $\texttt{BI}(M, i-1, \alpha)$ *lorsque $c(M, i, j, \alpha)$ est vrai. Comme $mpNext[i]$ est la longueur du plus long bord de $M[0..i-1]$ pour le score seuil $BI(M, i-1, \alpha)$, il suffit de choisir j pour que $i - j$ soit maximum, c'est-à-dire j minimum.* ◄

Comme $\texttt{mpNext}[i] = i - j$, le décalage du motif à effectuer sur le texte est $i - \texttt{mpNext}[i] = i - (i - j) = j$.

Le calcul de $b(M, i, j, \beta, \delta)$ s'exprime avec les formules récursives suivantes.

Lemme 3.6

$$\begin{aligned} b(M, j, j, 0, 0) &= 1 \\ b(M, i, j, \beta, \delta) &= \exists x \in \Sigma\ b(M, i-1, j, \beta - M(i, x), \delta - M(i-j, x)) \end{aligned}$$

Preuve : Pour le cas d'arrêt, c'est la définition de b appliquée avec $i = j$. Dans le cas général, $b(M, i, j, \beta, \delta)$ est vrai alors :

$$\exists u \in \Sigma^{i-j+1} \quad (\texttt{Score}(M[j..i], u) = \beta \ \wedge \ \texttt{Score}(M[0..i-j], u) = \delta)$$

ce qui équivaut à

$$\exists x \in \Sigma \quad \exists v \in \Sigma^{i-j} \\ (\texttt{Score}(M[j..i-1], v) = \beta - M(i, x) \ \wedge \ \texttt{Score}(M[0..i-j-1], v) = \delta - M(i-j, x))$$

c'est-à-dire

$$\exists x \in \Sigma \ b(M, i-1, J, \beta - M(i, x), \delta - M(i-j, x))$$

◄

Mise en oeuvre du calcul de la table `mpNext`

Ces formules de récurrence invitent naturellement à résoudre le problème par programmation dynamique. Calculer b et c pour tous les couples de scores serait bien trop coûteux. De plus tous les b ne seront pas utiles. Nous allons donc limiter les calculs en procédant par itération croissante sur la position de fin de tentative dans le motif, en s'aidant du lemme ci-dessous.

Lemme 3.7 *Si $c(M, i, j, \alpha) = 0$, alors $c(M, i+1, j, \alpha) = 0$*

Preuve : Si $c(M, i, j, \alpha) = 0$ alors il pour tout mot u de longueur $i - j$ on a :

1. $\texttt{Score}(M[j..i-1], u) < \texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..j-1])$, *et*
2. $\texttt{Score}(M[0..i-j-1], u) < \texttt{BI}(M, i-j-1, \alpha)$

d'après les définitions de c et b. Pour toute lettre x de Σ on a alors :

1. $\texttt{Score}(M[j..i], ux) < \texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..j-1]) + M(i, x) < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..j-1]$, *et*
2. $\texttt{Score}(M[0..i-j-1], ux) < \texttt{BI}(M, i-j-1, \alpha) + M(i-j, x) < \texttt{BI}(M, i-j, \alpha)$

En conclusion, pour tout u de Σ^{i-j} et pour tout x de Σ les deux conditions ci-dessus sont vérifiées et $c(M, i+1, j, \alpha) = 0$. ◄

Lemme 3.8 $c(M, j, j, \alpha) = 1$

La remarque ci-dessous nous permet de choisir le sens de parcours des j pour le calcul de c.

Remarque 3.1 *Dès que $c(M, i, j, \alpha)$ est vrai alors il est inutile de considérer les j plus grands puisque d'après le Lemme 3.5 on choisit le j le plus petit.*

Cela nous conduit à un algorithme où, pour chaque longueur j de décalage possible, nous allons tester les positions i de fin de tentative de manière croissante.

Définition 3.4 *Pour tout j dans $\{1..m-1\}$, on définit :*

$$\begin{aligned}\mathcal{S}_j^j &= \{(s_1,0)|\exists u \in \Sigma^j, s_1 \geq \texttt{BI}(M,j-1,\alpha)\}\\ \mathcal{S}_i^j &= \{(s_1,s_2)|\\ &\quad \exists x \in \Sigma, \exists (s_1',s_2') \in \mathcal{S}_{i-1}^j,\\ &\quad s_1 = s_1' + M(i-1,x) \geq \texttt{BI}(M,i-1,\alpha), s_2 = s_2' + M(i-j-1,x)\}\forall i \in \{j+1..m-1\}\end{aligned}$$

Lemme 3.9 *Si un couple de scores (s_1,s_2) appartient à $\mathcal{S}_i^j$ alors il existe un mot u de Σ^j tel que $b(M,i-1,j,s_1 - \texttt{Score}(M[0..j-1],u_0..u_{j-1}),s_2) = 1$.*

Preuve : u est le préfixe de longueur j d'un mot de score s_1. Ce mot de score s_1 existe nécessairement par construction de $\mathcal{S}_i^j$. De même il existe un mot v de score s_2. On retrouve la définition de b. ◄

Lemme 3.10 *Pour tout couple (s_1,s_2) de $\mathcal{S}_i^j$, $j \leq i \leq m-1$, il existe un mot u de Σ^j tel que $s_1 - \texttt{Score}(M[0..j-1],u_0..u_{j-1}) \geq \texttt{BI}(M,i-1,\alpha) - \texttt{ScoreMax}(M[0..j-1])$.*

Preuve : C'est vrai à la construction de $\mathcal{S}_j^j$. Ensuite, soit (s_1,s_2) un couple de $\mathcal{S}_{i+1}^j$, soit v le mot de longueur i tel que $\texttt{Score}(M[0..i-1],v) = s_1$, u est le préfixe de longueur j de v, et on a :

$$\begin{aligned}s_1 - \texttt{Score}(M[0..j-1],u_0..u_{j-1}) &\geq \texttt{BI}(M,i,\alpha) - \texttt{Score}(M[0..j-1],u_0..u_{j-1})\\ &\geq \texttt{BI}(M,i,\alpha) - \texttt{ScoreMax}(M[0..j-1])\end{aligned}$$

◄

Lemme 3.11 *Pour tout couple (s_1,s_2) de $\mathcal{S}_i^j$, $j \leq i \leq m-1$, si $s_2 \geq \texttt{BI}(M,i-j-1,\alpha)$ alors $c(M,i,j,\alpha) = 1$.*

Preuve : Soit (s_1,s_2) un couple de $\mathcal{S}_i^j$, avec les lemmes 3.9 et 3.10 on sait qu'il existe un mot u de longueur j tel que :

$$b(M,i-1,j,s_1 - \texttt{Score}(M[0..j-1],u_0..u_{j-1}),s_2) = 1$$

et :

$$s_1 - \texttt{Score}(M[0..j-1],u_0..u_{j-1}) \geq \texttt{BI}(M,i-1,\alpha) - \texttt{ScoreMax}(M[0..j-1])$$

Si $s_2 \geq \texttt{BI}(M,i-j-1,\alpha)$ alors $c(M,i,j,\alpha)$ est vrai par définition de c. ◄

Nous proposons ainsi de construire successivement la suite des S_i^j, puis de mettre à jour $c(M,i,j,\alpha)$ suivant la valeur de s_2 de chaque couple. L'algorithme est donné Algorithme 2.

3.2 Extension de la table de Knuth, Morris et Pratt

Nous expliquons maintenant comment enrichir l'extension Morris et Pratt en une extension Knuth-Morris et Pratt. Nous avons vu au chapitre 2 que l'amélioration de KMP par rapport à MP consistait à tirer partie de l'information d'échec à la position de lecture courante pour anticiper un éventuel échec sur le même caractère après décalage. C'est la condition 2 de la définition 2.7 de bord étiqueté.

Algorithme 2 : Calcul de la table `mpNext`

Entrée: M une matrice de longueur m, α un score seuil
Sortie: `mpNext`

1: `mpNext`$[i] = -1$ pour tout i de $\{0..m\}$
2: $c(M, i, j, \alpha) = 0$ pour tout $0 \leq i \leq m$ et $0 \leq j \leq m$
3: **pour** j allant de 1 à $m-1$ **faire**
4: calculer $\mathcal{S}_j^j$
5: $c(M, j, j, \alpha) \leftarrow 1$ { *Lemme 3.8* }
6: **pour** i allant de j à $m-1$ **faire**
7: $\mathcal{S}_{i+1}^j \leftarrow \emptyset$
8: **pour tout** (s_1, s_2) de $\mathcal{S}_i^j$ **faire**
9: **pour tout** $x \in \Sigma$ **faire**
10: **si** $s_1 + M(i, x) \geq$ `BI`(M, i, α) **alors**
11: $\mathcal{S}_{i+1}^j \leftarrow \mathcal{S}_{i+1} \cup \{(s_1 + M(i, x), s_2 + M(i - j, x))\}$
12: **si** $\nexists j' \leq j$ $c(M, i, j, \alpha) = 1$ {*Remarque 3.1*} **alors**
13: **si** $s_2 \geq$ `BI`$(M, i - j - 1, \alpha)$ **alors**
14: $c(M, i, j, \alpha) \leftarrow 1$ {*Lemme 3.11*}
15: **si** $c(M, i, j, \alpha) = 0$ **alors**
16: `mpNext`$[i] = i - j$ {*Lemme 3.5*}
17: **pour tout** (s_1, s_2) de $\mathcal{S}_m^j$ **faire**
18: **si** $\nexists j' \leq j$ $c(M, i, j, \alpha) = 1$ { *Remarque 3.1*} **alors**
19: **si** $s_2 \geq$ `BI`$(M, m - j - 1, \alpha)$ **alors**
20: $c(M, m, j, \alpha) \leftarrow 1$ { *Lemme 3.11* }
21: **si** $c(M, m, j, \alpha) = 0$ **alors**
22: `mpNext`$[m] = m - j$ { *Lemme 3.5* }

3.2.1 Table des correspondances partielles

Dans le contexte de matrices score-position, cette condition de caractères distincts ne fait pas directement de sens. Elle devient une condition sur les scores observés aux deux positions.

Définition 3.5 (Table des correspondances partielles `kmpNext`) *Soit M une matrice de longueur m, α un score seuil. La table des correspondances partielles de l'extension de Knuth-Morris-Pratt de M, notée* `kmpNext`*, est un vecteur de taille $m+1$ dont les éléments sont définis par* `kmpNext`$[0] = -1$*, et pour toute valeur de i de $\{1, \ldots, m\}$,* `kmpNext`$[i]$ *est la longueur ℓ du plus long bord u de $M[0..i-1]$ et* `BI`$(M, i-1, \alpha)$ *tel qu'il existe x de Σ satisfaisant*

1. `Score`$(M[0..\ell], ux) \geq$ `BI`(M, ℓ, α)*, et*
2. `Score`$(M[i-\ell..i], ux) <$ `BI`$(M, i, \alpha) -$ `ScoreMax`$(M[0..i-\ell-1])$.

-1 si u n'existe pas.

On peut vérifier que cette définition coïncide avec la définition 2.8 dans le cas d'une matrice modélisant un mot exact. On associe à un mot t de Σ^m la même matrice M_t que pour Morris-Pratt (voir ci-dessus). Dans le cas de la matrice M_t, `BI`$(M_t, \ell, m) = \ell + 1$, `BI`$(M_t, i, m) = i + 1$

et $\texttt{ScoreMax}(M[0..i-\ell-1]) = i-l$. Il s'ensuit que $\texttt{kmpNext}[i]$ est le plus long bord $t_0..t_{l-1}$, $0 < l \leq i$, tel que il existe $x \in \Sigma$ satisfaisant :

1. $\texttt{Score}(M_t[0..l], t_0..t_{l-1}x) \geq l+1$
2. $\texttt{Score}(M_t[i-1..i], t_0..t_{l-1}x) < l+1$

Pour que 1) soit vrai il faut nécessairement que $x = t_l$ et pour que 2) soit vrai il faut que x soit différent de t_i. On a alors $kmpNext[i]$ est le plus long bord $t_0..t_{l-1}$, $0 < l \leq i$, tel que $t_l \neq t_i$. On retrouve la définition de $\texttt{kmpNext}[i]$ pour les mots exacts.

Lemme 3.12 *Soit u un mot de Σ^ℓ et x une lettre de Σ. u et x satisfont les conditions de la Définition 3.5 si, et seulement si*

1. $\texttt{Score}(M[0..\ell-1], u) \geq \texttt{BI}(\ell, \alpha) - M(\ell, x)$, *et*
2. $\texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1]) \leq \texttt{Score}(M[i-\ell..i-1], u) < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1]) - M(i, x)$

Preuve : L'équivalence des conditions 1) est directe.

Supposons avoir les conditions de la Définition 3.5. Si il existe x tel que la condition 2) de la Définition 3.5 est vérifiée alors trivialement $\texttt{Score}(M[i-\ell..i-1], u) < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1]) - M(i, x)$. *De plus comme u est un bord de $M[0..i-1]$, on* $\texttt{Score}(M[i-\ell..i-1], u) \geq \texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1]$. *On retrouve donc la condition 2) du lemme.*

Supposons avoir maintenant les conditions du lemme vérifiées, on a $\texttt{Score}(M[i-\ell..i-1], u) \geq \texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1])$ *d'après la partie gauche de la condition 2) et* $\texttt{Score}(M[0..\ell-1], u) \geq \texttt{BI}(M, \ell, \alpha) - M(\ell, x) \geq \texttt{BI}(M, \ell-1, \alpha)$ *d'après la condition 1). Ainsi u est un bord. On obtient trivialement la condition 2) de la définition avec la partie droite de la condition 2) du lemme.* ◄

3.2.2 Calcul effectif des éléments de la table

À l'instar du prédicat c utilisé pour l'extension de Morris-Pratt, on définit c' qui permet de résoudre le calcul de la table des correspondances partielles pour KMP.

Définition 3.6

$c'(M, i, j, \alpha) =$
$\quad \exists x \in \Sigma$
$\quad \exists \texttt{BI}(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..j-1]) \leq \beta < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..j-1]) - M(i, x)$
$\quad \exists \delta \geq \texttt{BI}(M, i-j, \alpha) - M(i-j, x)$
$\quad b(M, i-1, j, \beta, \delta)$

Lemme 3.13 *$kmpNext[i]$ est égal à $i-j$, où j est la plus petite valeur telle que $c'(M, i, j, \alpha)$ soit vrai.*

Preuve : Si $c'(M, i, j, \alpha)$ est vrai alors il existe un mot u de longueur $i-j$ tel que $\texttt{Score}(M[j..i-1], u) = \beta$ *et* $\texttt{Score}(M[0..i-j-1], u) = \delta$ *(d'après la définition de b) et une lettre x de Σ tels que :*

1. $BI(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..j-1]) \leq \beta < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..j-1]) - M(i, x)$, *et*
2. $\delta \geq \texttt{BI}(M, i-j, \alpha) - M(i-j, x)$

Par conséquent :

1. $BI(M, i-1, \alpha) - \texttt{ScoreMax}(M[0..j-1]) \leq \texttt{Score}(M[j..i-1], u) < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..j-1]) - M(i, x)$, *et*
2. $\texttt{Score}(M[0..i-j-1], u) \geq \texttt{BI}(M, i-j, \alpha) - M(i-j, x)$

En posant, $\ell = i - j$, on retrouve les deux conditions du lemme 3.12 et donc la définition de $\texttt{kmpNext}[i]$ *sous la condition de prendre j tel que $i - j$ soit le plus grand, c'est-à-dire le plus petit j.* ◄

Le calcul de la table `kmpNext` se fait de la même manière que pour le calcul de la table `mpNext`. Les seuls changements à opérer dabs l'algorithme sont : que l'initialisation de $c(M, j, j, \alpha) = 1$ n'est plus vrai, et que la condition sur le score s_2 qui était $s_2 \geq \texttt{BI}(M, i-j-1, \alpha)$ devient $s_2 + M(i-j, x) \geq \texttt{BI}(M, i-j, \alpha)$.

3.3 Table des correspondances partielles à $|\Sigma|$ entrées

L'amélioration proposée par Knuth-Morris-Pratt consistait à chercher le plus long bord du motif suivi d'un caractère différent du caractère de la position d'échec dans le motif. Le but était d'augmenter les chances de ne pas subir un échec immédiat à la tentative suivante. On peut étendre cette notion, dans le cadre des matrices, en proposant de calculer, pour chaque caractère possible x de l'alphabet, la longueur du plus long bord non suivi de x. Plus précisément, on calculera, pour chaque caractère possible x de l'alphabet, la longueur l du plus long bord u vérifiant les deux conditions de la Définition 3.5. On utilisera alors le caractère lu dans le texte, et sur lequel s'est produit l'échec, pour effectuer le décalage et être certain de ne pas avoir un échec immédiat.

Définition 3.7 (Table des correspondances partielles $\texttt{kmpNext}_\Sigma$) *Soit M une matrice de longueur m, α un score seuil. La table des correspondances partielles* $\texttt{kmpNext}_\Sigma$ *est une matrice de taille $m + 1 \times |\Sigma|$ dont les éléments sont définis par,* $\texttt{kmpNext}_\Sigma[0][x] = -1 \forall x \in \Sigma$ *et* $\texttt{kmpNext}_\Sigma[i][x]$ *est la longueur ℓ du plus long bord u de $M[0..i-1]$ et* $\texttt{BI}(M, i-1, \alpha)$ *tel que :*

1. $\texttt{Score}(M[0..\ell], ux) \geq BI(M, \ell, \alpha)$, *et*
2. $\texttt{Score}(M[i-\ell..i], ux) < \texttt{BI}(M, i, \alpha) - \texttt{ScoreMax}(M[0..i-\ell-1])$.

-1 si u n'existe pas.

Lemme 3.14 $\texttt{kmpNext}[i] = \min_{x \in \Sigma} \texttt{kmpNext}_\Sigma[i][x]$

Preuve : $\texttt{kmpNext}_\Sigma[i][x]$ *fournit la longueur du plus long bord u vérifiant les conditions 1) et 2) de la Définition 3.7. Si on prend $\ell = \min_{x \in \Sigma} \texttt{kmpNext}_\Sigma[i][x]$ alors nécessairement il existe une lettre y de Σ telle que u et y vérifient les conditions de la Définition 3.5.* ◄

Le calcul effectif de $\texttt{kmpNext}_\Sigma$ n'est en fait pas réalisé à partir du calcul de `kmpNext`. On peut redéfinir c'' à partir de c', la seule différence étant que la lettre sur laquelle il y a échec est fixée.

Définition 3.8

$$
\begin{aligned}
&c''(M,i,j,\alpha,x) = \\
&\quad \exists \texttt{BI}(M,i-1,\alpha) - \texttt{ScoreMax}(M[0..j-1]) \leq \beta < \texttt{BI}(M,i,\alpha) - \texttt{ScoreMax}(M[0..j-1]) - M(i,x) \\
&\quad \exists \delta \geq \texttt{BI}(M,i-j,\alpha) - M(i-j,x) \\
&\quad b(M,i-1,j,\beta,\delta)
\end{aligned}
$$

Tout ce qui a été dit sur `kmpNext` reste valable. On remarquera qu'il suffit d'échanger l'ordre des boucles sur x et sur les couples de $\mathcal{S}_i^j$ pour calculer directement $\texttt{kmpNext}_\Sigma$.

Pour l'utilisation de l'algorithme 1 avec la table $\texttt{kmpNext}_\Sigma$, il suffira de modifier l'instruction $j \leftarrow j+$ next$[i]$ par $j \leftarrow j+$ next$[i]$[T[j+i]] .

3.4 Résultats

Nous avons lancé la calcul des tables de décalage dans les différentes versions et pour des P-valeurs allant de 10^{-3} à 10^{-6} (Figure 3.1). Pour chaque matrice, nous avons ensuite comparé le nombre de positions testées avec les différents algorithmes et reporté le gain. Le gain représente donc le pourcentage de positions non testées par rapport à l'algorithme avec élagage sans table de décalage. La longueur de la séquence test était de 10^7 lettres. L'utilisation de la table $\texttt{kmpNext}_\Sigma$donne les meilleurs résultats.

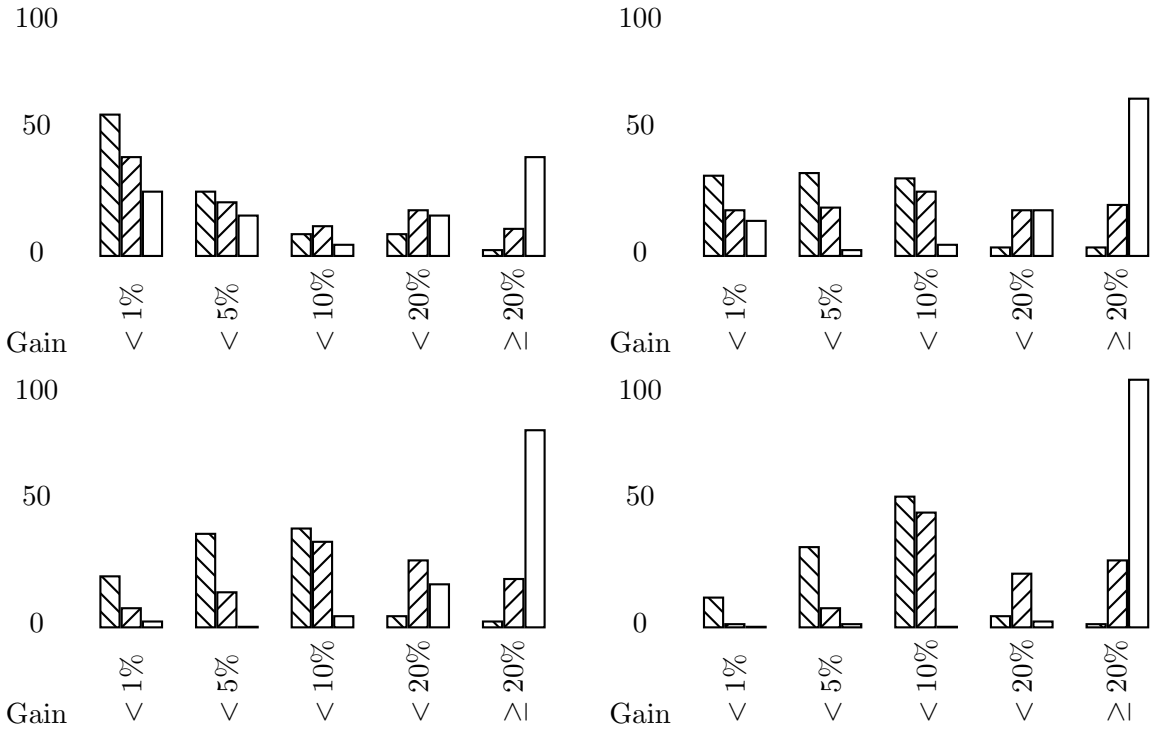

FIG. 3.1: En ordonnée, le pourcentage de matrices Jaspar pour lesquelles on observe le gain en abscisse apporté par l'usage de la table des correspondances partielles, en terme de caractères lus (nombre de caractères lus avec la table par rapport au nombre de caractères lus avec l'algorithme de Wu *et al.*) pour les P-valeur 10^{-3} (en haut à gauche), 10^{-4} (en haut à droite), 10^{-5} (en bas à gauche) et 10^{-6} (en bas à droite). Chaque groupe de trois barres représente, de gauche à droite, le gain apporté avec les extensions MP, KMP, KMP avec alphabet.

Chapitre 4

Recherche de matrices avec découpage

Sommaire

Nous avons présenté dans le troisième chapitre une optimisation du problème de la localisation des occurrences d'une matrice score-position dans un texte en adaptant des techniques de pré-traitement du motif à la Knuth-Morris-Pratt. Dans ce quatrième chapitre, nous explorons une autre piste de recherche basée sur le caractère même des matrices et du calcul du score. L'idée initiale est simplement d'accélérer le calcul du score en agglomérant des colonnes de la matrice. Cela se fait en découpant la matrice en tranches, et en pré-calculant tous les scores possibles pour les sous-mots de chaque tranche. Ces scores pourront être stockés dans une structure d'index.

Dans la première section, nous définissons formellement le découpage d'une matrice et sa structure d'index associé. Dans la deuxième section, nous montrons comment cette approche peut être combinée à l'élagage et expliquons comment construire un découpage optimal dans ce cadre. Dans la troisième section, nous expliquons comment combiner également le traitement par découpage à la Knuth-Morris-Pratt.

Dans les deux dernières sections du chapitre, nous abordons le problème de la recherche multiple de matrices, c'est-à-dire de la recherche simultanée des occurrences d'un ensemble de matrices. Ce problème trouve sa justification dans l'existence de larges banques de données de sites de fixation de facteurs de transcription, présentées au chapitre 1. Un mode d'utilisation standard de ces banques est de rechercher toutes les occurrences possibles de toutes les matrices, sans préjugé, puis à appliquer un post-traitement sur l'ensemble des occurrences trouvées. C'est ce qui est fait par exemple dans `TFM-Explorer` [20], TOUCAN [2], TargetExplorer [73] ou BLISS [54]. Les approches à la Knuth-Morris-Pratt sont mal adaptées à la recherche simultanée de plusieurs matrices. Les approches par découpage s'y prêtent mieux. Nous présentons cette adaptation dans la Section 4. Enfin dans la Section 5, nous étudions le problème de la recherche multiple en présence de matrices similaires. Là encore, ce problème est motivé par l'exemple pratique des banques de sites de fixation de facteurs de transcription, où figurent de nombreuses entrées similaires ou même redondantes.

4.1 Découpage d'une matrice en tranches et index

L'idée de base est de pré-calculer la distribution de score pour l'ensemble des mots possibles pour une matrice. Ces scores pourront être stockés dans un index, et il suffira d'accéder à cet index pour connaître le score d'une position. En pratique, il n'est bien sûr pas possible de considérer l'intégralité des mots pour des valeurs de m et de Σ courantes : $m = 20$ et $|\Sigma| = 4$ par exemple. La solution que nous développons consiste à découper la matrice en tranches, ce qui est une situation intermédiaire entre la matrice initiale et le pré-calcul intégral. Il suffit alors de pré-calculer l'ensemble des scores possibles pour chaque tranche de la matrice.

Définition 4.1 (Découpage) *Soit M une matrice score-position de longueur m. Un découpage de M en ℓ tranches est défini par une suite croissante de $\ell + 1$ positions $i_0, \ldots, i_\ell$ telles que $i_0 = 0$ et $i_\ell = m$. Les valeurs $i_0, \ldots, i_{\ell-1}$ correspondent chacune à la première position de chaque tranche. La* structure d'index *pour M et $i_0, \ldots, i_{\ell-1}$, notée $S_{M,i_0 \ldots i_\ell}$ est un tableau à deux entrées. Pour un indice de tranche k de $[0..\ell - 1]$ et un mot u de $\Sigma^{i_{k+1}-i_k}$, $S_{M,i_0 \ldots i_\ell}(k, u)$ est défini par*

$$S_{M,i_0 \ldots i_\ell}(k, u) = \texttt{Score}(u, M[i_k..i_{k+1} - 1])$$

Le score final d'un mot u de Σ^m pour la matrice M s'obtient par la somme entre les tranches, grâce à la propriété d'additivité des scores :

$$\texttt{Score}(u, M) = \sum_{k=0}^{\ell-1} S_{M,i_0 \ldots i_\ell}(k, u_{i_k} \ldots u_{i_{k+1}-1})$$

Concernant la mise en œuvre de l'index, pour une tranche de longueur t, on définit un vecteur de $|\Sigma|^t$ éléments, qui contient les valeurs de scores des $|\Sigma|^t$ mots de longueur t pour la tranche.

Les mots sont naturellement classés par ordre lexicographique. La clé de chaque mot est calculée avec la fonction de hachage classique h :

$$h(u) = \sum_{i=0}^{t-1} \text{pos}(u_i, \Sigma) \times |\Sigma|^i$$

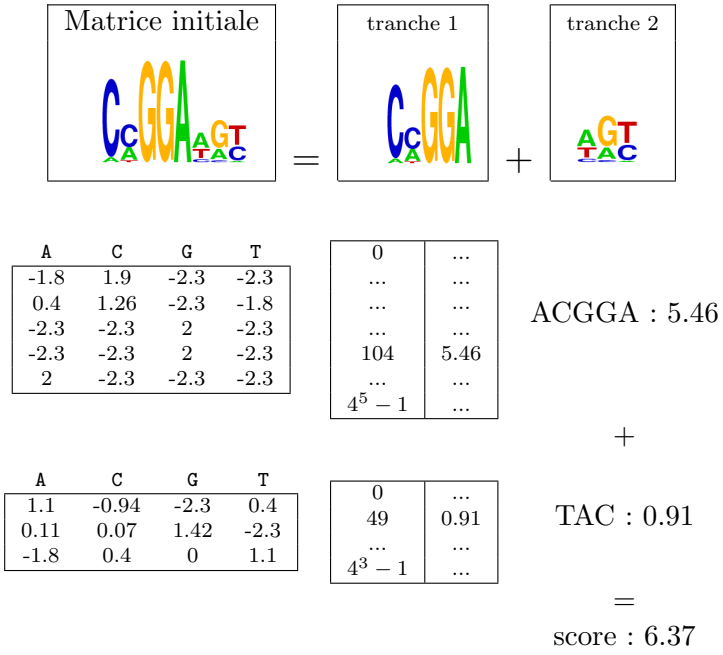

FIG. 4.1: Découpage en tranche et calcul de score

où $\text{pos}(x, \Sigma)$ est le numéro d'ordre de la lettre x dans l'alphabet Σ (x est compris entre 0 et $|\Sigma| - 1$). La Figure 4.1 illustre cela.

L'accès à l'index se fait ainsi en temps constant. Le nombre d'accès pour un mot donné est égal au nombre de tranches du découpage, ℓ au lieu de m. C'est en fait l'espace mémoire disponible qui impose une limite inférieure sur le nombre de tranches, à travers la taille de l'index.

4.2 Combinaison de la méthode d'élagage et découpage

On peut naturellement combiner le découpage avec la méthode d'élagage présentée au chapitre 2, et déjà reprise au chapitre 3. Le calcul s'arrête à la sortie d'une tranche dès que le score obtenu est inférieur à la borne inférieure requise. Un exemple est détaillé en Figure 4.2.

Dans ce cas, le comportement en temps de l'algorithme de recherche ne dépend plus uniquement du nombre de tranches, mais également des bornes de ces tranches. On a intérêt à ce que le choix des bornes des tranches favorise les cas d'abandon par élagage. Intuitivement, le découpage doit privilégier les bornes des tranches qui correspondent aux positions discriminantes de la matrice, avec un poids important dans la décision finale. Plus exactement, nous choisissons comme définition de découpage optimal celui qui minimise le nombre moyens d'accès à la table pour un espace mémoire maximum donné. Ce nombre moyen d'accès est lié au nombre total d'opérations, puisque chaque accès donne lieu à une addition, et réciproquement. 0n a donc besoin de déterminer le nombre moyen d'accès par tranche.

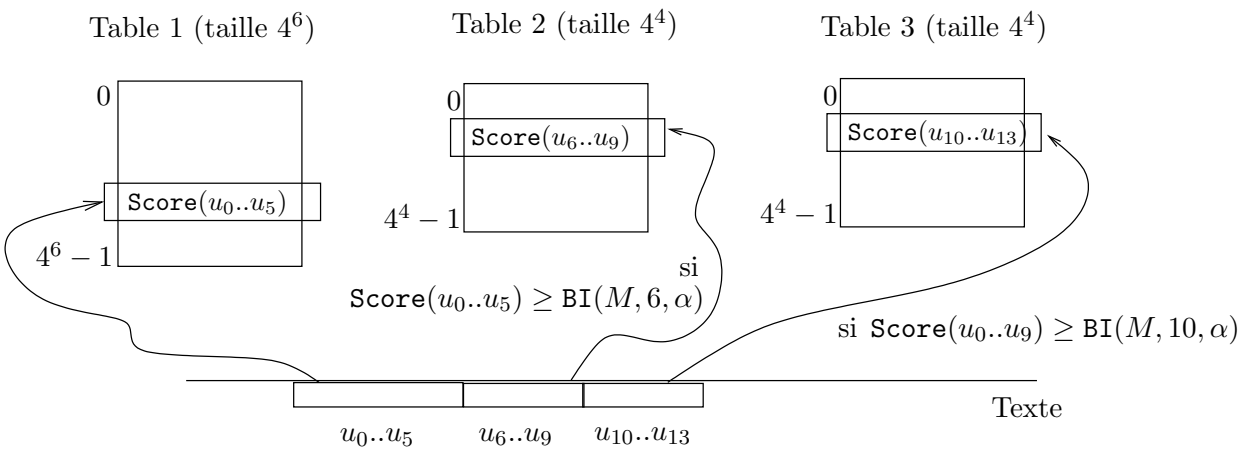

FIG. 4.2: Exemple d'accès à l'index

Pour chaque position, on considère l'ensemble des mots *candidats* pour lesquels il faudra au moins calculer le score jusqu'à cette position pour conclure si le mot est occurrence ou non. La probabilité de cet ensemble représente donc la probabilité qu'une tentative se poursuive après le calcul du score en position $i-1$ de la matrice.

Définition 4.2 (Ensemble des mots candidats Cand) *Soient M une matrice de longueur m, i une position de M et α un score seuil. On définit l'ensemble des mots candidats de M, i et α, noté $\texttt{Cand}(M, i, \alpha)$ par :*

$$\texttt{Cand}(M, i, \alpha) = \{u \in \Sigma^m,\ \texttt{Score}(M[0..i-1], u_0 \dots u_{i-1}) \geq \texttt{BI}(M, i-1, \alpha)\}$$

Lemme 4.1 $\mathbb{P}(\texttt{Cand}(M, i, \alpha)) = \texttt{P-valeur}(M[0..i-1], \texttt{BI}(M, i-1, \alpha))$

Preuve : Par définition de la P-valeur donnée par la Définition 2.16 et de BI. ◄

Dans le cas de l'algorithme d'élagage seul, le calcul de score s'interrompt dès que le score courant est inférieur au score intermédiaire minimum $\texttt{BI}(M, i-1, \alpha)$. Le nombre moyen d'opérations est alors $\sum_{i=0}^{m-1} \mathbb{P}(\texttt{Cand}(M, i, \alpha))$.

Pour une matrice découpée, le nombre moyen d'opérations devient $\sum_{k=0}^{\ell-1} \mathbb{P}(\texttt{Cand}(M, i_k, \alpha))$. Le découpage optimal est donc celui qui minimise cette quantité sous contrainte d'espace mémoire. Soit e, l'espace mémoire maximum disponible. On note $\phi(j, e)$ le nombre moyen d'accès pour l'intervalle de positions $[1..j]$ de M.

$$\phi(0, e) = \begin{cases} 0, & \text{if } e \geq 0 \\ +\infty, & \text{if } e < 0 \end{cases} \quad (4.1)$$

$$\phi(j, e) = \min_{0 \leq i < j} \phi(i, e - |\Sigma|^{j-i}) + \mathbb{P}(\texttt{Cand}(M, i, \alpha)) \quad (4.2)$$

Lemme 4.2 *Soit M une matrice de longueur m. Le découpage optimal de M pour un espace mémoire maximal e est obtenu grâce au calcul de $\phi(m, e)$.*

La formule de récurrence induit naturellement un calcul de ϕ par programmation dynamique. L'obtention des indices de début de tranche est obtenu par remontée de la matrice de programmation dynamique.

4.3 Combinaison des méthodes KMP et découpage

Il est également possible de combiner le découpage d'une matrice avec les extensions de KMP présentées au chapitre 3.

4.3.1 Saut associé à une tranche

Dans l'algorithme KMP usuel, la table des correspondances partielles indique le saut qui peut être effectué lors d'un échec, en fonction de la position d'échec i. Cette valeur de saut est donnée par $i - \texttt{kmpNext}(i)$. Cette fois, nous ne pouvons exécuter un décalage qu'aux positions qui correspondent à la fin d'une tranche de la matrice découpée. Pour une tranche donnée, le décalage maximum qui peut être réalisé avec l'assurance de ne pas perdre d'occurrences est en fait le décalage minimum qui peut être réalisé à une des positions de cette tranche. Cela donne lieu à la définition suivante.

Définition 4.3 (Table des décalages par tranche) *Soit M une matrice munie d'un découpage. À chaque tranche $[i..j]$, on associe le saut*

$$\texttt{Saut}[i..j] = \min\{k - \texttt{kmpNext}(k),\ i \leq k \leq j\}$$

où `kmpNext` *est la table des correspondances partielles de M, suivant la Définition 3.5 du chapitre 3.*

Lors de la lecture d'un mot u, un saut est effectué à l'issue du calcul du score de la tranche $[i..j]$ si les deux conditions suivantes sont satisfaites :

- le score jusqu'à la position $i-1$ comprise ne conduit pas à un échec : $\texttt{Score}(M[0..i-1], u_0 \dots u_{i-1}) \geq \texttt{BI}(M, i-1, \alpha)$,
- le score jusqu'à la position j comprise conduit à un échec : $\texttt{Score}(M[0..j], u_0 \dots u_j) < \texttt{BI}(M, j, \alpha)$.

Cela nous conduit à définir l'ensemble $F(M, i, j\alpha)$ des mots induisant un saut sur la tranche $[i..j]$ exactement :

$$\begin{aligned} F_{M,i,j\alpha} \quad = \quad & \{u \in \Sigma^m, \texttt{Score}(M[0..i-1], u_0 \dots u_{i-1}) \geq \texttt{BI}(M, i-1, \alpha) \\ & \wedge \texttt{Score}(M[0..j], u_0 \dots u_j) < \texttt{BI}(M, j, \alpha)\} \end{aligned}$$

Lemme 4.3 $\mathbb{P}(F(M, i, j, \alpha)) = \mathbb{P}(\texttt{Cand}(M, i, \alpha)) - \mathbb{P}(\texttt{Cand}(M, j, \alpha))$.

Preuve : $\texttt{Cand}(M, \alpha, i)$ *est l'union disjointe de* $\texttt{Cand}(M, \alpha, j)$ *et* $F(M, \alpha, j, i)$. ◀

Ce lemme montre qu'il est possible de calculer effectivement $\mathbb{P}(F(M, i, j, \alpha))$, en utilisant le Lemme 4.1. De plus, le calcul des deux termes $\mathbb{P}(\texttt{Cand}(M, i, \alpha))$ et $\mathbb{P}(\texttt{Cand}(M, j, \alpha))$ de la soustraction repose en fait sur un même calcul de P-valeur, avec la même table de programmation dynamique. À partir de là, on peut calculer le *saut moyen* associé à une tranche. C'est simplement le saut pondéré par la probabilité de saut de la tranche : $\texttt{Saut}[i..j]\mathbb{P}(F(M, i, j, \alpha))$.

4.3.2 Détermination d'un bon découpage

Etant données une matrice M de longueur m et ℓ tranches $i_0, \ldots, i_{\ell-1}$, on choisit comme découpage celui qui minimise le saut moyen total :

$$\sum_{k=0}^{\ell-1} \texttt{Saut}[i_k..i_{k+1}-1]\mathbb{P}(F(M, i_k, i_{k+1}-1, \alpha))$$

Cette minimisation se fait bien sûr sous une contrainte d'espace donnée e. Cela nous conduit de nouveau de définir une formule de récurrence permettant d'obtenir les indices des tranches de matrices, à l'image de l'Equation 4.1 de la section précédente. On note $\psi(i, e)$ le saut moyen pour le découpage construit entre les positions i et $m-1$, avec un espace mémoire de e.

$$\psi(m, e) = \begin{cases} 0, & \text{si } e \geq 0 \\ +\infty, & \text{si } e < 0 \end{cases} \quad (4.3)$$

$$\psi(i, e) = \min_{i<j}\{\psi(j, e - |\Sigma|^{j-i}) + \texttt{Saut}[i..j-1]\mathbb{P}(F(M, i, j-1, \alpha))\ \} \quad (4.4)$$

4.4 Recherche multiple de matrices

4.4.1 Définition de l'index pour la recherche multiple

De manière formelle le problème de la localisation multiple d'un ensemble de matrices score-position dans un texte est défini comme suit.

Définition 4.4 (Localisation d'un ensemble de matrices score-position) *Soient $\mathcal{M} = \{M_1, \ldots, M_k\}$ un ensemble de matrices score-position, $\alpha_1, \ldots, \alpha_k$ les scores seuils associés, et T un texte. Le problème de la localisation d'un ensemble de matrices score-position, ou de recherche multiple, consiste à rechercher toutes les occurrences de toutes les matrices M_j, $1 \leq j \leq k$, de $\mathcal{M}$ dans le texte T.*

Pour aborder ce problème de manière efficace, l'idée de base est de reprendre et d'adapter la structure d'index présentée en Section 1 dans le cas de la recherche simple. Dans le cas d'une unique matrice, l'index permet de stocker l'ensemble des scores possibles des tranches pour un découpage donné de la matrice. La différence est qu'on doit maintenant stocker les scores pour les tranches induites par un découpage commun à l'ensemble des matrices.

Définition 4.5 (Structure d'index multiple) *Soient $\mathcal{M}$ un ensemble de matrices score-position, ℓ le nombre de tranches, et $i_0 \ldots i_\ell$ le découpage correspondant. La* structure d'index *pour $\mathcal{M}$ et $i_0 \ldots i_\ell$, notée $S_{\mathcal{M},i_0 \ldots i_\ell}$, associe à chaque matrice M de $\mathcal{M}$, à chaque tranche k de $[0..\ell-1]$ et à chaque mot u de $\Sigma^{i_{k+1}-i_k}$ le score de u pour la tranche k de M, en tenant compte de la longueur de M :*

$$S_{\mathcal{M},i_0 \ldots i_\ell}(M, k, u) = \texttt{Score}(M[i_k \ldots \min\{i_{k+1}-1, |M|\}], u)$$

Le score final d'un mot u pour une matrice M s'obtient par la somme suivante :

$$\texttt{Score}(u, M) = \sum_{k=0}^{l-1} S_{\mathcal{M},i_0 \ldots i_{\ell-1}}(M, k, u_{i_k} \ldots u_{i_{k+1}-1})$$

4.4.2 Calcul du découpage optimal

L'index pour la recherche multiple s'accommode également de l'optimisation par élagage. Comme précédemment, se pose la question de la définition des tranches avec le choix du découpage optimal. L'objectif est maintenant de munir l'ensemble des matrices d'un même découpage de manière à minimiser le nombre total moyen d'accès. Cela dépend à la fois de l'élagage et de la répartition des longueurs des matrices au sein de l'ensemble.

On étend la définition de mot candidat pour prendre en compte le fait que des matrices peuvent être de longueur inférieure à ce qu'autorise l'index. Par convention, on définit $\texttt{Cand}(M, i, \alpha)$ comme étant l'ensemble vide pour toutes les valeurs de i supérieures à la longueur de M. Dans ce cas, la probabilité associée est naturellement nulle.

On note $\phi(j, e)$ le nombre moyen d'accès pour les positions $0 \ldots j-1$ et un espace mémoire maximum e.

$$\phi(0, e) = \begin{cases} \text{Card}(\mathcal{M}) & \text{si } e \geq 0 \\ +\infty & \text{sinon} \end{cases} \tag{4.5}$$

$$\phi(j, e) = \min_{0 \leq i < j} \{\phi(i, e - |\Sigma|^{j-i} + \sum_{M_k \in \mathcal{M}} \mathbb{P}(\texttt{Cand}(M_k, i, \alpha_k) \tag{4.6}$$

Le découpage complet s'obtient avec $\phi(\max_{M \in \mathcal{M}} |M|, e)$ où e est la mémoire disponible.

4.4.3 Résultats expérimentaux

L'algorithme de recherche multiple avec élagage a été mis en œuvre dans un logiciel écrit en C++, appelé TFM-Scan. TFM-Scan est disponible à l'URL `http ://bioinfo.lifl.fr/TFM/TFMscan/`. Ce logiciel permet également de traiter la recherche simple.

Nous avons évalué les performances de TFM-Scan sur l'ensemble des matrices des bases de données Transfac et Jaspar, soit 608 matrices. Sur ces données, la structure d'index compte quatre tranches. La taille de chacune de ces tranches et le nombre de matrices par tranche sont données par la Figure 4.3.

La Figure 4.4 montre les temps de calculs comparés pour l'algorithme naïf et pour l'algorithme avec index, pour différentes longueurs de texte sur ce jeux de matrices. Le gain est d'un facteur huit.

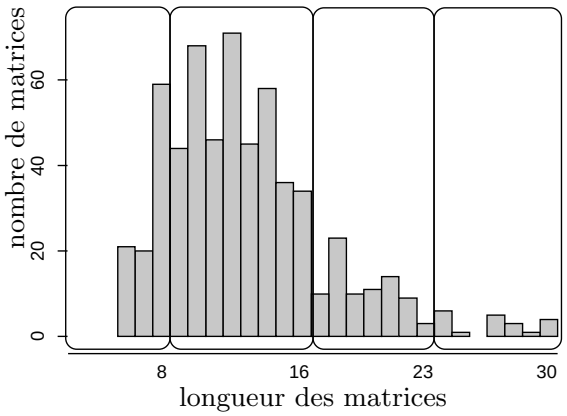

FIG. 4.3: L'histogramme représente la distribution des longueurs de matrices Transfac and Jaspar (602 au total). Le découpage optimal pour ce jeu de matrices est composé des 4 tranches : 1..8, 9..16, 17..23 et 24..30. Le nombre de matrices pour chaque tranche est respectivement 602, 502, 100 and 20. Après l'élagage, le nombre d'accès moyen est respectivement 602, 216, 18 and 1.

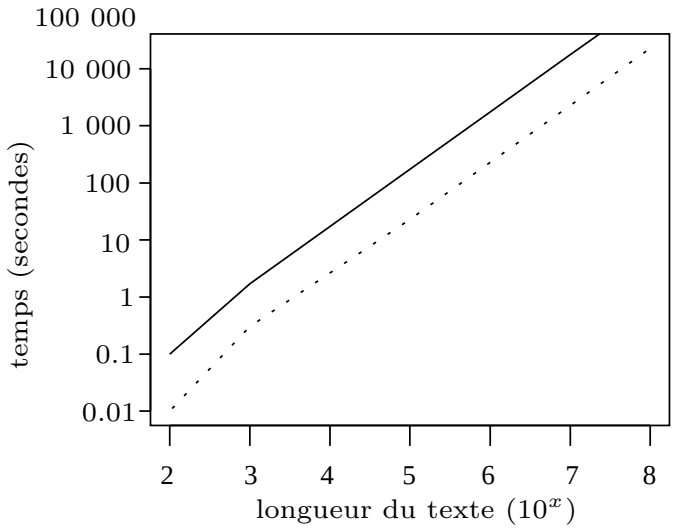

FIG. 4.4: Le graphique compare les temps de calcul de l'algorithme naïf (ligne pleine) avec celui proposé (en pointillé) pour un espace mémoire disponible de 256MB.

4.5 Le traitement de matrices similaires

Dans la section précédente, nous avons présenté un algorithme efficace de recherche d'un ensemble arbitraire de matrices dans un texte. Nous n'avions fait aucune hypothèse sur le jeu de données. Dans notre contexte d'application, il arrive fréquemment que les matrices considérées soient similaires. Il peut s'agir de sites de fixation de facteurs de transcription associés à des protéines régulatrices homologues. La ressemblance peut également provenir de la redondance entre banques de données, ou au sein d'une banque de données. La Figure 4.5 montre un exemple de quatre matrices similaires extraites de Transfac.

Dans ce cas, les occurrences des matrices sont corrélées. L'idée est d'exploiter cette corrélation pour accélérer la recherche. Nous proposons dans cette section deux solutions au problème de la recherche de matrices similaires dans un texte, une exacte et une approchée.

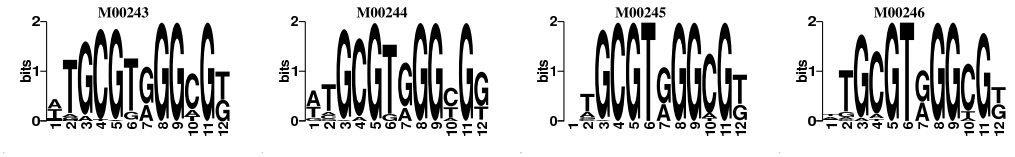

FIG. 4.5: Quatre matrices de la famille EGR-1 extraites de Transfac.

4.5.1 Comparaison et regroupement de matrices

Les deux algorithmes que nous proposons s'articulent suivant le même schéma :

1. classification des matrices de $\mathcal{M}$ en groupes suivant leur similitude ;
2. choix d'une matrice représentante pour chaque groupe. Cette matrice aidera à décider pour toutes les matrices du groupe si celles-ci sont ou non occurrences à une position donnée.

Pour évaluer la similitude entre matrices, les articles [68, 72, 64] proposent différentes méthodes de mesure de la corrélation entre les coefficients des matrices. Nous présentons ici deux nouvelles manières de comparer deux matrices, qui travaillent directement sur les occurrences des matrices, et non sur leurs coefficients. C'est plus pertinent quand il s'agit de faire de la recherche de motifs.

Avant de définir formellement comment comparer les matrices, nous nous plaçons dans le cadre de travail où les matrices sont de même longueur et dèja alignées. L'alignement consiste ici à mettre en correspondance les positions similaires des matrices par décalage, sans introduire de gaps. Une fois les matrices alignées, on peut facilement les égaliser en longueur en remplissant les colonnes manquantes à droite ou à gauche par des colonnes de 0.

La première approche de comparaison entre matrices s'appuie sur la comparaison des scores, avec le *seuil d'incertitude*. L'idée est que deux matrices sont proches si la connaissance du score de l'une donne une indication précise sur le score de la seconde.

Définition 4.6 (Seuil d'incertitude) *Soit M et N deux matrices de même longueur m. Le* seuil d'incertitude *entre M et N, noté $U(M,N)$, est défini comme le plus petit entier naturel positif n tel que*

$$\forall u \in \Sigma^m \quad \forall \alpha \in \mathbf{N} \quad \texttt{Score}(N,u) \leq \alpha - n \Rightarrow \texttt{Score}(M,u) \leq \alpha$$

Lemme 4.4

$$U(M,N) = \sum_{i=0}^{m-1} \max_{x \in \Sigma} \left(M(i,x) - N(i,x)\right)$$

Preuve : Soit u un mot de Σ^m. Si on suppose que

$$\texttt{Score}(N,u) \leq \alpha - \sum_{i=0}^{m-1} \max_{x\in\Sigma}(M(i,x) - (N(i,x))$$

alors on en déduit

$$\sum_{i=0}^{m-1} N(i,u_i) + \max_{x\in\Sigma}(M(i,x) - (N(i,x)) \leq \alpha$$

or

$$M(i,u_i) - N(i,u_i) \leq max_{x\in\Sigma}(M(i,x) - N(i,x)) \forall i \in \{0..m-1\}$$

et donc

$$\sum_{i=0}^{m-1} M(i,u_i) \leq \alpha$$

Il s'ensuit

$$\texttt{Score}(N,u) \leq \alpha - \sum_{i=0}^{m-1} \max_{x\in\Sigma}(M(i,x) - (N(i,x)) \Rightarrow \texttt{Score}(M,u) \leq \alpha$$

On voit également que la solution de l'inéquation de la définition 4.6 ne peut pas être inférieure à $\sum_{i=0}^{m-1} \max_{x\in\Sigma}(M(i,x) - (N(i,x))$.

◀

La seconde approche propose de raffiner cette définition, en mesurant la probabilité de l'ensemble des mots pour lesquelles deux matrices donnent des résultats divergents. Pour cela, nous définissons les *faux positifs* et les *faux négatifs*.

Définition 4.7 (Faux positifs et faux négatifs) *Soit M et N deux matrices et α et β leurs scores seuils respectifs. Soit u un mot de Σ^*.*

- *u est un* faux positif *pour N par rapport à M si* $\texttt{Score}(u,M) \geq \alpha$ *et* $\texttt{Score}(u,N) < \beta$,
- *u est un* faux négatif *pour N par rapport à M si* $\texttt{Score}(u,M) < \alpha$ *et* $\texttt{Score}(u,N) \geq \beta$.

On note $FP(M,N,\alpha,\beta)$ la probabilité de l'ensemble des faux positifs, et $FN(M,N,\alpha,\beta)$ la probabilité de l'ensemble des faux négatifs.

La Figure 4.6 montre schématiquement la répartition des faux positifs et des faux négatifs en deux dimensions. Il est possible de calculer de manière exacte et efficace. FP et FN.

Lemme 4.5 $FP(M,N,\alpha_1,\alpha_2) = l(|M|,\alpha_1,\alpha_2)$,
où la fonction l est définie récursivement par

$$\begin{aligned} l(0,s_1,s_2) &= \begin{cases} 1 & si\ s_1 \geq 0\ et\ s_2 < 0 \\ 0 & sinon \end{cases} \\ l(i,s_1,s_2) &= \sum_{x\in\Sigma} l(i-1, s_1 - M(x,i), s_2 - N(x,i)) \end{aligned}$$

$FN(M,N,\alpha_1,\alpha_2) = l'(|M|,\alpha_1,\alpha_2)$,
où la fonction l' est définie récursivement par

$$\begin{aligned} l'(0,s_1,s_2) &= \begin{cases} 1 & si\ s_1 < 0\ et\ s_2 \geq 0 \\ 0 & sinon \end{cases} \\ l'(i,s_1,s_2) &= \sum_{x\in\Sigma} l'(i-1, s_1 - M(x,i), s_2 - N(x,i)) \end{aligned}$$

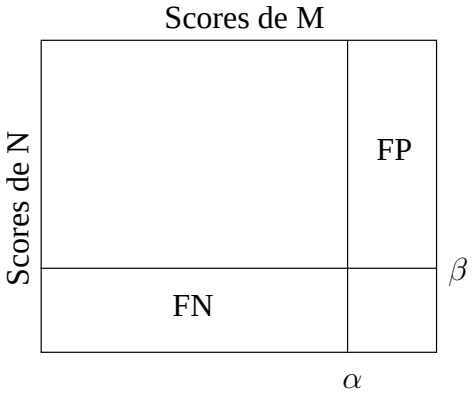

FIG. 4.6: Faux positifs et faux négatifs pour N par rapport à M. Les seuils d'acceptation sont α pour M et β pour N

Le calcul des FP et des FN se fait par programmation dynamique sur i. Il s'apparente en fait à un calcul de P-valeur en dimension 2.

4.5.2 Algorithme exact

L'algorithme exact de recherche de matrices utilise le seuil d'incertitude pour mettre en place une approche par filtrage sans perte. Pour chaque groupe, la matrice représentante N sert de filtre. Les autres matrices M_i du groupe ne sont évaluées que si le score trouvé avec la matrice représentante tombe dans l'intervalle d'incertitude de N et M_i.

Algorithme 3 : Algorithme exact de filtrage

Data : un texte T, un ensemble de matrices score-position $\{(M_1, \alpha_1) \dots (M_k, \alpha_k)\}$
Result : toutes les occurrences de $M_1, \dots, M_k$ sur T

Définir une matrice filtre N pour $\{(M_1, \alpha_1), \dots, (M_k, \alpha_k)\}$ **pour** j *allant de* 1 *à* p **faire**
  Calculer $U(M_j, N)$;
pour *chaque position* i *du texte* T **faire**
  Calculer $s \leftarrow \texttt{Score}(S, i, N)$;
pour j *allant de* 1 *à* p **faire**
  si $s > \alpha_j - U(M_j, N)$ **alors**
    calculer $\texttt{Score}(S, i, M_j)$;

Nous listons ici quelques exemples de matrices filtres pour un ensemble de matrices $\{M_1, \dots, M_k\}$:

– Matrice *moyenne arithmétique*

$$\forall x \in \Sigma, \forall i \in \{1 \dots |M|\}, \quad M(i,x) = \sum_{j=1}^{k} \frac{M_j(i,x)}{k}$$

– Matrice *minimum*

$$\forall x \in \Sigma, \forall i \in \{1 \dots |M|\}, \quad M(i,x) = \min\{M_1(i,x) \dots M_k(i,x)\}$$

– Matrice *maximum*

$$\forall x \in \Sigma, \forall i \in \{1 \dots |M|\}, \quad M(i,x) = \max\{M_1(i,x) \dots M_k(i,x)\}$$

– Matrice *cœur*

La proportion de texte nécessitant le calcul du score de M_j, dans la dernière partie de l'algorithme, est `P-valeur`$(N, \alpha_j - U(M_j, N))$. Par exemple, pour les quatre matrices de la Figure 4.5 avec un score seuil associé correspondant à une P-valeur de $exp(-5)$ et une matrice filtre moyenne arithmétique, la proportion de texte nécessitant le calcul des scores des matrices `M00243`, `M00244`, `M00245`, `M00246` est, respectivement, 3%, 2.3%, 2.1% et 5%.

4.5.3 Algorithme approché

On peut aussi exploiter la similarité entre matrices pour concevoir un algorithme approché extrêmement simple. On ne calcule que les scores de la matrice filtre et on conclut directement sur les occurrences de toutes les matrices du groupe. L'erreur ainsi faite est contrôlée grâce au calcul des faux positifs et des faux négatifs, donné par le Lemme 4.5.

Par exemple, pour les trois définitions de matrice filtre donnée au dessus,
– la matrice minimal de score seuil $\alpha = \max\{\alpha_1, \dots, \alpha_k\}$ garantit l'absence de faux positif.
– la matrice maximal de score seuil $\alpha = \min\{\alpha_1, \dots, \alpha_k\}$ garantit l'absence de faux négatif.
– et la matrice moyenne arithmétique autorise des faux positifs et des faux négatifs.

Algorithme 4 : Algorithme de filtrage approché

Data : un texte T, un ensemble de matrices $\{(M_1, \alpha_1) \dots (M_k, \alpha_k)\}$
Result : estimer toutes les occurrences de $M_1, \dots, M_k$ par les occurrences de la matrice filtre N sur T

Définir une matrice filtre N et un score seuil α pour $\{(M_1, \alpha_1), \dots, (M_k, \alpha_k)\}$
pour *chaque position i du texte T* **faire**
  Calculer `Score`(S, i, N) ;

Pour cet algorithme approché, le choix de la matrice filtre détermine la sélectivité et la sensibilité de l'estimation. Pour l'exemple de la Figure 4.5, les taux de FP et de FN sont respectivement 19% et 21% avec une matrice filtre moyenne arithmétique et pour un score seuil correspondant à une P-valeur de $exp(-5)$.

Conclusion et perspectives

Dans ce mémoire de thèse, nous nous sommes intéressés à la recherche de matrices score-position dans un texte. C'est un champ de recherche actuel, qui tient de l'algorithmique de texte en général et qui trouve des applications à la bio-informatique avec l'analyse de séquences biologiques.

Nous avons proposé deux contributions principales : l'adaptation des approches à la Knuth-Morris-Pratt aux matrices score-position en chapitre 3, et une approche par découpage qui exploite l'additivité du système de score en chapitre 4. Dans les deux cas, l'amélioration est obtenue grâce à un pré-traitement du motif. Cela correspond bien au contexte biologique, où le motif est stable et le texte variable (le nombre de motifs est faible en nombre et en croissance contrairement au texte). D'un point de vue méthodologique, ces deux méthodes ont également en commun de combiner des techniques algorithmiques, telles que la programmation dynamique, et des méthodes statistiques pour déterminer précisément les probabilités d'apparition de motifs et sous-motifs. L'approche par découpage s'applique aussi à des ensembles de matrices, tout en permettant de gérer la redondance entre motifs.

Nous avons développé un logiciel, TFM-Scan, mettant en oeuvre ces différents algorithmes et avons pratiqué des tests sur des données biologiques réelles. Les résultats expérimentaux montrent, qu'au delà de la motivation théorique, nos algorithmes ont de bons comportements pratiques. Nous les avons évalués sur des modèles de sites de fixations de facteurs de transcription dans le génome, à grande échelle. TFM-Scan est disponible sur http ://bioinfo.lifl.fr/TFMscan. Il intègre un modèle de Bernoulli avec pourcentage en GC variable par fenêtres (le modèle de texte évolue tout au long du texte). Il a déjà plusieurs utilisateurs identifiés, biologistes ou bio-informaticiens.

Ces travaux ouvrent plusieurs perspectives. Tout d'abord, il semble intéressant de poursuivre l'extension et l'évaluation des approches développées sur les mots exacts aux matrices score-position. Cela comprend l'algorithme de Boyer-Moore et la méthode Backward Dawg Matching. Concernant l'algorithme de Boyer-Moore, l'analyse faite pour le calcul du bord et le calcul du bord étiqueté dans le cadre de Knuth-Morris-Pratt doit facilement être adaptable au calcul de la table du Bon Suffixe. Concernant le Backward Dawg Matching, cela semble une alternative pertinente aux approches existantes basées sur l'automate de Aho-Corasick.

Une autre perspective de recherche est l'analyse de la distribution du nombre d'occurrences d'un motif, présentée au chapitre 2. Les calculs de bord pour la table des correspondances partielles de Knuth-Morris-Pratt est très proche de celui qui serait nécessaire pour calculer les auto-chevauchements entre matrices. Pour cela, il suffit quasiment de calculer la probabilité

des intersections de facteurs du motifs au lieu de simplement tester s'ils sont vides. À partir de là, on peut calculer le coefficient de corrélation nécessaire à la distribution.

Bibliographie

[1] Kel A and Al. Match : a tool for searching transcription factor binding sites in dna sequences. *Nucl. Acids Res.*, 31(13) :3576–3579, 2003.

[2] S. Aerts, G. Thijs, B. Coessens, M. Staes, Y. Moreau, and B. De Moor. Toucan : deciphering the cis-regulatory logic of coregulated genes. *Nucleic Acids Research*, 31(6) :1753–1764, 2003.

[3] H. Almagor. A markov analysis of dna sequences. *J.Theor. Biol.*, 104 :633–645, 1983.

[4] D. Angluin. Finding patterns common to set of strings. *J.Comput. System Sci.*, 21 :46–62, 1980.

[5] Dorohonceanu B. and Neville-Manning C.G. Accelerating protein classification using suffix trees. *ISMB*, 8 :128–133, 2000.

[6] F. Rodolphe et E. Turckheim B. Prum. Finding words with unexpected frequencies in dna sequences. *JRSS B*, 57 :205–220, 1995.

[7] R.A. Baaeza-Yates and G.H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10) :74–82, 1992.

[8] A. Bairoch. The prosite dictionary of sites and patterns in proteins, its current status. *Nucleic Acids Res*, 21 :3097–3103, 1993.

[9] Giegerich R. Beckstette M., Homann R. and Kurtz S. Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, 7 :389, 2006.

[10] P V Benos, M L Bulyk, and G D Stormo. Additivity in protein-dna interactions : how good an approximation is it ? *Nucleic Acids Res*, 30 :4442–51, 2002.

[11] P V Benos, A S Lapedes, and G D Stormo. Is there a code for protein-dna recognition ? probab(ilistical)ly... *Bioessays*, 24 :466–75, 2002.

[12] O.G. Berg and P.H. von Hippel. Selection of DNA binding sites by regulatory proteins. statistical-mechanical theory and application to operators and promoters. *Journal of Molecular Biology*, 193(4) :723–50, 1987.

[13] B.-E. Blaisdell. Markov chain analysis finds a significant influence of neighboring bases on the occurrence of a base in eucaryotic nuclear dna sequences both protein-coding and noncoding. *J. Mol. Evol.*, 21 :278–288, 1985.

[14] A Blumer, J Blumer, D Haussler, and R McConnell. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM (JACM)*, Jan 1987.

[15] P. Bucher. Weight matrix descriptions of four eukaryotic rna polymerase ii promoter derived from 502 unrelated promoter sequences. *J. Mol. Biol.*, 212 :563–578, 1990.

[16] Pasi Rastas Cinzia Pizzi and Esko Ukkonen. Fast search algorithms for position specific scoring matrices. *BIRD 2007*, 2007.

[17] J.M. Claverie and S. Audic. The statistical significance of nucleotide position-weight matrix matches. *Computer Applications in the Biosciences*, 12(5) :431–439, 1996.

[18] J.-H. Morris et V.-R. Pratt D.-E. Knuth. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(1) :323–350, 1977.

[19] I.-S. Mian et K. Sjölander D. Haussler, A. Krogh. Protein modeling using hidden markov models : Analysis of globins. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 1, pages 792–802. IEEE Computer Society Press, 1993.

[20] M Defrance and H Touzet. Predicting transcription factor binding sites using local over-representation and comparative .... *BMC Bioinformatics*, Jan 2006.

[21] M Dekker. *Handbook of fungal biotechnology.* Dilip K. Arora, 2003.

[22] G. Christian Overton E. A. Cheever and David B. Searls. Fast fourier transform-based correlation of dna sequences using complex plane encoding. *Computer Applications in the Biosciences*, 7 :143–154, 1991.

[23] S.-R. Eddy. Profile hidden markov models. *Bioinformatics Review*, 14(9) :755–63, 1998.

[24] C. Fondrat et A. Kalogeropoulos. Approaching the function of new genes by detection of their potential upstream activation sequences in saccharomyces cerevisiae : application to chromosome iii. *Curr Genet*, 25(5) :396–406, 1994.

[25] J.-T. Lecomte et C.-R. Matthews. Unravelling the mechanisme of protein folding : new tricks for an old problem. *Prot. Eng.*, 6 :1–10, 1990.

[26] G.-D. Stormo et D.-S. Fields. Specificity, free energy and information content in protein-dna interactions. *Biochemical Sciences*, 23(3) :109–13, 1998.

[27] S. Robin et J.-J. Daudin. Exact distribution of word occurrences in a random sequence of letters. *J. Appl. Prob.*, 36 :179–193, 1999.

[28] R.-S. Boyer et J.-S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10) :762–772, 1977.

[29] J.-L. Lassez et K. Marriot. Polynomial time inference of pattern languages and its applications. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations*, Computer Science, Mathematical Theory of Computations / The Complexity of Algorithms, pages 191–209, 1982.

[30] J.-L. Lassez et K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3) :301–318, 1987.

[31] M.-J. Fischer et M.-S. Paterson. String matching and other products. *Siam AMS Proc. on Complexity of Computation*, pages 113–126, 1974.

[32] Alfred V. Aho et Margaret J. Corasick. Efficient string matching : An aid to bibliographic search. *Communications of the ACM*, 18 :333–340, 1975.

[33] C.-O. Pabo et R.-T. Sauer. Transcription factors : structural families and principles of dna recognition. *Annu Rev Biochem.*, 61 :1053–95, 1992.

[34] T. Shinohara et S. Arikawa. Pattern inference. In *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 259–291. Springer, 1995.

[35] T. Yokomori et S. Kobayashi. Learning local languages and their application to dna sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10), 1998.

[36] E Roquain et S Schbath. Improved compound poisson approximation for the number of occurrences of multiple words in a stationary markov chain. *Appl. Proba*, 39 :1–13, 2007.

[37] G. Mengeritsky et T.-F. Smith. Recognition of characteristic patterns in sets of functionally equivalent dna sequences. *Comput. Appl. Biosci.*, 3 :223–227, 1987.

[38] G.-M. Landau et U. Vishkin. Introducing efficient parallelism into approximate string matching and a new serial algorithm. *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 220–230, 1986.

[39] J.-H. Morris et V.-R. Pratt. A linear pattern-matching algorithm. *Report 40, University of California, Berkeley*, 1970.

[40] J.-M. Chandonia et S.-E. Brenner G.-E. Crooks, G. Hon. A sequence logo generator. *Genome Research*, 14 :1188–1190, 2004.

[41] S. Robin et C. Baril G. Nuel. Predicting distances using a linear model : the case of varietal distinctness. *Journal of Applied Statistics*, 28(5) :607–21, 2001.

[42] Reinert Gesine, Schbath Sophie, and S. Waterman Michael. Probabilistic and statistical properties of words : An overview. *Journal of computationnal Biology*, 7(1-2) :1–46, 2000.

[43] Pietrokovski S Henikoff J, Greene E and Henikoff S. Increased coverage of protein families with the blocks database servers. *Nucleic Acids Research*, 28 :228–230, 2000.

[44] G.Z. Hertz and D. Stormo. Identifying DNA and protein patterns with statistically significant alignement of multiple sequences. *Bioinformatics*, 15(7-8) :563–77, 1999.

[45] Haiyan Huang, Ming-Chih J Kao, Xianghong Zhou, Jun S Liu, and Wing H Wong. Determination of local statistical significance of patterns in markov sequences with application to promoter element identification. *J Comput Biol*, 11(1) :1–14, 2004.

[46] B. Andre et J. Collado-Vides J. Van Helden. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J Mol Biol*, 281(5) :827–42, 1998.

[47] G. Soules et N. Weiss L.-E. Baum, T. Petrie. A maximization technique occuring in statistical analysis of probabilistic functions in markov chains. *The Annal of Mathematical Statistics*, 41(1) :164–171, 1970.

[48] G. Getz L. Hertzberg, O. Zuk and E. Domany. Finding motifs in promoter regions. *Journal of Computational Biology*, 12(3) :314–330, 2005.

[49] J. Ziv A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transaction on Information Theory*, 24(5), 1978.

[50] V.-I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8) :707–710, 1966.

[51] Ketil Malde and Robert Giegerich. Calculating pssm probabilities with lazy dynamic programming. *J. Funct. Program.*, 16(1) :75–81, 2006.

[52] U Manber and G Myers. Suffix arrays : a new method for on-line string searches. *Proceedings of the first annual ACM-SIAM symposium on ...*, Jan 1990.

[53] Udi Manber and Gene Myers. Suffix arrays : A new method for on-line string searches. *Siam Journal on Computing*, 22(5) :935–948, 1993.

[54] Hailong Meng, Arunava Banerjee, and Lei Zhou. Bliss : binding site level identification of shared signal-modules in dna regulatory sequences. *BMC Bioinformatics*, 7 :287, Jan 2006.

[55] S. Kurtz M.I. Abouelhoda and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2 :53–86, 2004.

[56] K Monostori, A Zaslavsky, and I Vajk. Suffix vector : A space-efficient suffix tree representation. *Proceedings of the 12th International Symposium on ...*, Jan 2001.

[57] I.-P. Ioshikhes N.-I. Gershenzon, G.-D. Stormo. Computational technique for improvement of the position-weight matrices for the dna/protein binding sites. *Nucleic Acids Research*, 33(7) :2290–2301, 2005.

[58] P. Nicodème, B. Salvy, and P. Flajolet. Motif statistics. In *Proc. European Symposium on Algorithms — ESA'99, Prague*, volume 1643 of *Lecture Notes in Computer Science*, pages 194–211. Springer-Verlag, 1999.

[59] K. Quandt, K. Frech, H. Karas, E. Wingender, and T. Werner. Matind and Matinspector - new fast and versatile tools for detection of consensus matches in nucleotide sequence data. *Nucleic Acids Research*, 23 :4878–4884, 1995.

[60] Jin X. Rajasekaran S. and Spouge J.L. The efficient computation of position-specific match scores with the fast fourier transform. *Journal of Computational Biology*, 9(1) :23–33, 2002.

[61] Mireille Regnier. A unified approach to word statistics. In *RECOMB*, pages 207–213, 1998.

[62] KARP R.M. and RABIN M.O. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2) :249–260, 1987.

[63] A. Sandelin, W. Alkema, P. Engstrom, W.W. Wasserman, and B. Lenhard. JASPAR : an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, Database issue :D91-4, 2004.

[64] Albin Sandelin and Wyeth W. Wasserman. Constrained binding site diversity within families of transcription factors enhances pattern discovery bioinformatics. *Journal of Molecular Biology*, 338(2) :207–215, 2004.

[65] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, 2000.

[66] S. Schbath. Compound poisson approximation of word counts in dna sequences. *ESAIM : Probability and Statistics*, 1 :116, 1995.

[67] S. Schbath. An overview on the distribution of word counts in markov chains. *J.Comp. Biol.*, 2000.

[68] E. D. Schones, P. Sumazin, and M.Q. Zhang. Similarity of position frequency matrices for transcription factor binding sites. *Bioinformatics*, 21(3) :307–13, 2005.

[69] Flower D.R. Scordis P. and Attwood T. Fingerprintscan : Intelligent searching of the prints motif database. *Bioinformatics*, 15(10) :799–806, 1999.

[70] David B. Searls. The linguistics of dna. *Am. Scient.*, 80 :579–591, 1992.

[71] David B. Searls. The computational linguistics of biological sequences. In *Artificial intelligence and molecular biology*, pages 47–120. American Association for Artificial Intelligence, 1993.

[72] Kielbasa SM, Gonze D, and Herzel H. Measuring similarities between transcription factor binding sites. *BMC Bioinformatics*, 6(237), 2005.

[73] Alona Sosinsky, Christopher P Bonin, Richard S Mann, and Barry Honig. Target explorer : An automated tool for the identification of new target genes for a specified set of transcription factors. *Nucleic Acids Res*, 31(13) :3589–92, Jul 2003.

[74] R Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res*, 12 :505–19, 1984.

[75] R. Staden. Methods for calculating the probabilities of finding patterns in sequences. *Computer Applications in the Biosciences*, 5 :89–96, 1989.

[76] R. Staden. Searching for patterns in protein and nucleic acid sequences. *Methods Enzymol*, 183 :193–211, 1990.

[77] G.-D. Stormo. Consensus patterns in dna. *Methods Enzymol*, 183 :211–221, 1990.

[78] G.-D. Stormo. Dna binding sites : representation and discovery. *Bioinformatics*, 16(1) :16–23, 2000.

[79] L. Gold et A. Ehrenfeucht T.-D. Schneider, G.-D. Stormo. Information content of binding sites on nucleotide sequences. *J Mol Biol.*, 188(3) :415–31, 1986.

[80] R.-M. Stephens T.-D. Schneider. Sequence logos : A new way to display consensus sequences. *Nucleic Acids Res*, 18 :6097–6100, 1990.

[81] Hélène Touzet and Jean-Stéphane Varré. Efficient and accurate p-value computation for position weight matrices. *Algorithms for Molecular Biology 2007 2 :15*, 2(15), 2007.

[82] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64 :100–118, 1985.

[83] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3) :249–260, 1995.

[84] Freschi V. and Bogliolo A. Using sequence compression to speedup probabilistic profile matching. *Bioinformatics*, 21(10) :2225–2229, 2005.

[85] E. Wingender, X. Chen, R. Hehl, I. Karas, I. Liebich, V. Matys, T. Meinhardt, M. Pruss, I. Reuter, and F. Schacherer. TRANSFAC : an integrated system for gene expression regulation. *Nucleic Acids Research*, 28(1) :316–319, 2000.

[86] Neville-Manning C.G. Wu T.D. and Brutlag D.L. Fast probabilistic analysis of sequence function using scoring matrices. *Bioinformatics*, 16(3) :233–244, 2000.

[87] Jing Zhang, Bo Jiang, Ming Li, John Tromp, Xuegong Zhang, and Michael Q. Zhang. Computing exact p-values for dna motifs. *Bioinformatics*, 23 :531–537, 2007.