Extraction de Motifs Communs dans un Ensemble de Séquences.Application à l'identification de sites de liaison aux protéines dans les séquences primaires d'ADN.

Alban Mancheron

École Centrale de Nantes **Université de Nantes** **École des Mines de Nantes**

ÉCOLE DOCTORALE STIM

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX »

Année 2006

N° attribué par la bibliothèque

Extraction de Motifs Communs dans un Ensemble de Séquences.

Application à l'identification de sites de liaison aux protéines dans les séquences primaires d'ADN.

THÈSE DE DOCTORAT

Discipline : INFORMATIQUE

Spécialité : INFORMATIQUE

Présentée
et soutenue publiquement par

Alban MANCHERON

Le 29 septembre 2006 à l'UFR Sciences & Techniques, Université de Nantes,
devant le jury ci-dessous

Président	:	Professeur Maxime CROCHEMORE	Université de Marne-La-Vallée
Rapporteurs	:	Alain DENISE, Professeur	Université de Paris-Sud
		Thierry LECROQ, Professeur	Université de Rouen
Examinateurs	:	Jérémie BOURDON, Maître de Conférence	Université de Nantes
		Richard NOCK, Professeur	Université des Antilles et de la Guyane

Directrice de thèse : Professeur Irena RUSU

Laboratoire: **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED 0366-269

EXTRACTION DE MOTIFS COMMUNS DANS UN ENSEMBLE DE SÉQUENCES.

APPLICATION À L'IDENTIFICATION DE SITES DE LIAISON AUX PROTÉINES DANS LES SÉQUENCES PRIMAIRES D'ADN.

Pattern Extraction from a Set of Sequences.

An application to proteins binding sites identification in DNA *primary sequences.*

Alban MANCHERON

⋈

favet neptunus eunti

Université de Nantes

Alban MANCHERON

Extraction de Motifs Communs dans un Ensemble de Séquences.

Application à l'identification de sites de liaison aux protéines dans les séquences primaires d'ADN.

IV+XII+274 p.

Ce document a été préparé avec LaTeX2e et la classe `these-LINA` version v. 2.7 de l'association de jeunes chercheurs en informatique LOGIN, Université de Nantes. La classe `these-LINA` est disponible à l'adresse : `http://login.irin.sciences.univ-nantes.fr/`.
Cette classe est conforme aux recommandations du ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche (circulaire n° 05-094 du 29 mars 2005), de l'Université de Nantes, de l'école doctorale « Sciences et Technologies de l'Information et des Matériaux » (ED-STIM), et respecte les normes de l'association française de normalisation (AFNOR) suivantes :

- AFNOR NF Z41-006 (octobre 1983)
  Présentation des thèses et documents assimilés ;
- AFNOR NF Z44-005 (décembre 1987)
  Documentation – Références bibliographiques – Contenu, forme et structure ;
- AFNOR NF Z44-005-2/ISO NF 690-2 (février 1998)
  Information et documentation – Références bibliographiques – Partie 2 : documents électroniques, documents complets ou parties de documents.

Impression : ma-these.tex – 13/11/2006 – 9:51.

Révision pour la classe : these-LINA.cls,v 2.7 2006-09-12 17:18:53 mancheron Exp

Verba secretorum Hermetis – Verum, sine mendacio, certum et verissimum : quod est inferius est sicut quod est superius ; et quod est superius est sicut quod est inferius, ad perpetranda miracula rei unius. Et sicut omnes res fuerunt ab uno, mediatione unius, sic omnes res natæ fuerunt ab hac unare, adaptatione. Pater ejus est Sol, mater ejus Luna ; portavit illud Ventus in ventre suo ; nutrix ejus Terra est. Pater omnis telesmi totius mundi est hic. Vis ejus integra est si versa fuerit in terram. Separabis terram ab igne, subtile a spisso, suaviter, cum magno ingenio. Ascendit a terra in cœlum, iterumque descendit in terram, et recipit vim superiorum et inferiorum. Sic habebis gloriam totius mundi. Ideo fugiet a te omnis obscuritas. Hic est totius fortitudine fortitudo fortis ; quia vincet omnem rem subtilem, omnemque solidam penetrabit. Sic mundus creatus est. Hinc erunt adaptationes mirabiles, quarum modus est hic. Itaque vocatus sum Hermes Trismegistus, habens tres partes philosophiæ totius mundi. Completum est quod dixi de operatione Solis.

— Hermetis TRISMEGISTI, Tabula Smaragdina.

Il est vrai, sans mensonge, certain et très véritable : ce qui est en bas est comme ce qui est en haut, et ce qui est en haut est comme ce qui est en bas ; par ces choses se font le miracle d'une seule chose. Et comme toutes les choses sont et proviennent d'un, par la médiation d'un, ainsi toutes les choses sont nées de cette chose unique par adaptation. Le Soleil en est le père, et la Lune la mère. Le vent l'a porté dans son ventre. La Terre est sa nourrice et son réceptacle. Le Père de tout, le Thélème du monde universel est ici. Sa force ou puissance est entière si elle est convertie en terre. Tu sépareras la terre du feu, le subtil de l'épais, doucement avec grande industrie. Il monte de la terre et descend du ciel, et reçoit la force des choses supérieures et des choses inférieures. Tu auras par ce moyen la gloire du monde, et toute obscurité s'enfuira de toi. C'est la force, forte de toute force, car elle vaincra toute chose subtile et pénétrera toute chose solide. Ainsi, le monde a été créé. De cela sortiront d'admirables adaptations, desquelles le moyen est ici donné. C'est pourquoi j'ai été appelé Hermes TRISMÉGISTE*, ayant les trois parties de la philosophie universelle. ce que j'ai dit de l'œuvre solaire est complet.*

— Hermes TRISMÉGISTE, La Table d'Émeraude (Traduction de FULCANELLI [49]).

Résumé

L'extraction de motifs ayant une signification biologique, et notamment l'identification de *sites de régulation* de la synthèse protéique dans les séquences primaires d'ADN, est un des enjeux de la recherche en bioinformatique. Une anomalie dans cette régulation peut avoir de graves conséquences sur la santé d'un organisme. Aussi, l'extraction de ces sites permet de mieux comprendre le fonctionnement cellulaire et de soigner certaines pathologies.

Les difficultés posées par ce problème sont le manque d'informations sur les motifs à extraire, ainsi que le volume important des données à traiter. Deux algorithmes polynomiaux – l'un déterministe et l'autre probabiliste – permettant de le traiter ont été conçus. Dans ce contexte, nous avons introduit une nouvelle famille de fonctions de score et étudié leurs propriétés statistiques. Nous avons également caractérisé le langage reconnu par la structure d'index appelée « Oracle », et proposé une amélioration la rendant plus efficace.

Mots-clés : bioinformatique, algorithmique, statistiques, automates, extraction de motifs, structures d'index, application à l'ADN.

Abstract

The extraction of significant biological patterns, and in particular the identification of regulation sites of proteinic synthesis in DNA primary sequences, is one of the major issues today in bioinformatics. Indeed any anomaly in proteinic synthesis regulation has detrimental damages on the well-being of certain organisms. Extracting these sites enables to better understand cellular operation or even to remove or cure pathology.

What is problematic is the lack of information on patterns to be extracted, as well as the large volume of data to mine. In this dissertation, we introduce two polynomial algorithms – the first one is deterministic and the other one is probabilist – to address the issue of pattern extraction. We introduce a new family of score functions and we study their statistical properties. We characterize the language which is recognized by the index structure named "Oracle", and we modify this structure in order to make it more efficient.

Keywords: bioinformatics, algorithmics, statistics, automata, pattern extraction, index structures, DNA application.

Classification ACM

Catégories et descripteurs de sujets : E.1 [**Data**]: Data Structures; F.1 [**Theory of Computation**]: Computation by Abstract Devices; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; G.3 [**Mathematics of Computing**]: Probability and Statistics; H.1.1 [**Information Systems**]: Models and Principles—*Systems and Information Theory*; H.3.1 [**Information Systems**]: Information Storage and Retrieval—*Content Analysis and Indexing*; H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*; H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*; I.5 [**Computing Methodologies**]: Pattern Recognition.

Termes généraux : Algorithms, Measurement, Performance, Theory.

Remerciements

Il est d'usage d'entamer la thèse par une série de remerciements. C'est essentiellement une bonne connaissance de l'usage qui permet précisément d'y faire quelques entorses. Aussi ne vais-je pas commencer ce manuscrit par des remerciements, mais en m'excusant auprès de ceux que j'ai entraîné dans cette aventure.

Je demande donc à mes filles Harmonie et Constance, ainsi qu'à ma femme Mireille, de me pardonner pour leur avoir imposé ce long périple qu'est la thèse. Je les remercie du plus profond de mon cœur pour m'avoir donné le courage et la motivation qui m'ont permis de la mener à terme.

Je remercie également toutes les personnes de mon entourage, à commencer par mon frère Gus pour son soutien inconditionnel. Je remercie également mes parents et beaux-parents, pour s'être intéressé à mon travail qui parfois (pour ne pas dire souvent) n'a pas dû leur sembler très concret.

Cette thèse (et pas seulement la phase de rédaction) n'a pas abouti sans souffrances ; c'est ce qui – je l'espère – la rend pour le moins intéressante. J'adresse donc également mes remerciements à Irena pour m'avoir guidé et formé à la recherche. Plus généralement, je tiens à signifier ma reconnaissance à toutes les personnes qui ont consacré de leur temps à lire et relire ce manuscrit ; notamment Alain DENISE et Thierry LECROQ qui ont accepté d'en être les rapporteurs, ainsi que Maxime CROCHEMORE qui a accepté de présider mon jury de thèse. Sans faire de publicité, un grand merci aussi à mon pote *Google*™ qui a su répondre à bon nombre de mes interrogations. Ce travail n'aurait pas vu le jour sans le financement du Conseil Régional des Pays de la Loire, et doit son aboutissement à l'Université de Nantes et à l'Université des Antilles et de la Guyane.

En plus de le remercier pour avoir lu ma thèse, je tiens à signifier toute ma gratitude à Jerry pour l'orientation qu'il a insufflé à ma recherche, pour sa Normandie, pour m'avoir supporté au quotidien pendant deux ans et pour avoir apporté des réponse aux questions existentielles d'un doctorant plongé dans le doute le plus profond.

Parmi mes collègues, je souhaite remercier tout particulièrement Jean BÉZIVIN pour sa gentillesse, et ce dès le DEA ; Didier ROBBES pour m'avoir fait découvrir et aimer l'informatique alors que j'étais un jeune étudiant ; Marie-Mad' TALLINEAU, Alain VAILLY et Mourad OUSSALAH pour m'avoir accordé leur confiance en tant qu'enseignant ; Christian ATTIOGBÉ, Laurent GRANVILLIERS, Frédéric GOUALARD, Laurent UGHETTO et Éric LANGUÉNOU pour leur sympathie à mon égard ; Richard NOCK, Jean-Émile SYMPHOR et Abdellatif MOUDAFI pour leur accueil et leur soutien.

J'ai également une pensée toute particulière pour mes collègues doctorants ou jeunes docteurs, principalement Erwan MOREAU, Lamiaa NAOUM, Lucas BORDEAUX, Benj' HABBEGGER, Nordine FOUROUR, ainsi que pour les dinosaures de LOGIN : Gaëtan GAUMER, Gyôm RASCHIA, Pascal POIZAT, et enfin Vincent ROSSIGNOL. *I Thank* François DAMBRE pour sa contribution au résumé en anglais. Je remercie ceux qui ont eu la malchance de me supporter plus que les autres, ceux qui furent mes collègues de bureau, notamment Christophe MOAN, Marco CHRISTIE, Amenel VOGLOSIN, Manu ALLAUD et Harry GROS-DÉSORMEAUX.

Merci beaucoup à mes amis pour avoir pris régulièrement des nouvelles de ma petite famille. Je pense en particulier à Jérôme BOUCHAUD, Nicolas SAVARY, Fabien LE GRAND, Sylvain DEMEY, Camille JOLY, Céline BARRÉ et ma filleule Marine.

Je remercie aussi Christine BRUNET, Régine CHAVIGNY DE LA CHEVROTIÈRE, pour tous les services qu'elles m'ont rendus avec le sourire, ainsi que Bertrand et Josette pour les moments de détente.

Pour finir, je tiens à remercier celui ou celle qui, un jour, a découvert le café [134], sans quoi il m'eût probablement fallu encore de longues années avant d'achever ce travail ; avec une spéciale dédicace pour Cécile GUICHARD...

Avant de quitter cette page, je renouvelle mes remerciements à ma femme et mes enfants, qui m'ont accompagné durant ce long périple, et qui ont – envers et contre tout – continué de me soutenir, de me préserver.

Merci.

Á Constance & Harmonie.

Sommaire

— *Corps du document* —

— *Annexes* —

— *Pages annexées* —

Introduction et cadre de la thèse

It is with great regret that we have to announce the death, on friday 18^{th} *July* 1952 *of* D.N.A. *helix (crystalline).*
Death follows a protracted illness which an intensive course of besselised injections has failed to relieve.
A memorial service will be held next monday or tuesday.
It is hoped that Dr. M. H. F. WILKINS *will speak in memory of the late helix.*

— Rosalind Elsie FRANKLIN [1920–1958], note manuscrite.

Rosalind Elsie FRANKLIN est née à Londres le 25 juillet 1920. Elle suit des études de physique et de chimie au Newham College de Cambridge, dont elle sort diplômée en 1941. C'est alors qu'elle rejoint le Laboratoire central des services chimiques de Jacques MÉRING, à Paris, où elle participe à divers travaux sur la structure du charbon et d'autres composés carbonés jusqu'en 1950. C'est également à cette occasion qu'elle se familiarise avec les techniques de diffraction par rayons X. Elle entre au King's College en 1951. Les clichés que Rosalind Elsie FRANKLIN réalise alors sur l'ADN lui permettent de proposer une structure en double hélice pour cette molécule [45]. C'est également en 1953 qu'elle entre au Laboratoire de cristallographie du Birkbeck College, où elle se consacra à l'étude du virus de la mosaïque du Tabac.

[1] Les données biographiques sont issues principalement de [34, 48, 92, 135].

Francis Harry Compton CRICK est né le 8 juin 1916 à Northampton. Il a commencé par étudier la physique à l'Université de Londres. Il interrompit ses études en 1939 en raison de la deuxième guerre mondiale. Il reprit des études en biologie en 1947, et obtint sa thèse en 1954. En 1961, l'équipe de recherche de Francis Harry Compton CRICK découvrit le codage génétique des protéines [32], et en 1962, Maurice Hugh Frederick WILKINS, James Dewey WATSON et Francis Harry Compton CRICK reçurent le prix NOBEL de Médecine pour « leur découverte concernant la structure des acides nucléiques, et leur signification dans le transfert d'informations chez les êtres vivants. » Rosalind Elsie FRANKLIN aurait très probablement été associée à ce prix si elle n'avait pas été prématurément emportée par la maladie, le prix NOBEL ne pouvant (selon le règlement) être accordé à titre posthume.

La découverte du code génétique fût une découverte capitale dans le domaine de la biologie. Elle a permis ensuite à Hamilton SMITH de découvrir en 1972 le premier site spécifique dans l'ADN [47][2]. Il s'agissait alors d'un problème très difficile, et il n'existait que peu de moyens pour extraire cette information. Pour reprendre une métaphore bien connue, « autant chercher une aiguille dans une botte de foin ». C'est pourquoi, avec l'apparition de l'informatique, des chercheurs, tant biologistes qu'informaticiens, se sont intéressés à la recherche et à l'extraction de motifs.

Ainsi, une communauté s'est formée afin de résoudre des problèmes liés à l'extraction d'informations des molécules biologiques telles que l'ADN, l'ARN ou encore les protéines, donnant naissance à la bioinformatique (*cf.* [27] pour une présentation sommaire de la bioinformatique). En effet, la masse de données à traiter est sans cesse grandissante, et bien trop importante pour être traitée par l'humain. Le principe est d'extraire grâce à l'informatique des « renseignements », qui seront ensuite utilisés par des

[2] Il s'agit d'un site de liaison formé par 6 nucléotides (*cf.* Section 1.1.1 – page 11) , dont le complémentaire inversé est identique à lui même : GTYRAC (*cf.* Annexe A.1)

biologistes. Depuis, de nombreux algorithmes ont vu le jour, basés sur des modèles probabilistes ou bien déterministes (*cf.* [30] pour un aperçu, et [19] pour plus de détails sur les algorithmes d'extraction de motifs), qui sont souvent complémentaires. De plus, avec les facilités qu'offre *internet*, de plus en plus de laboratoires de biologie utilisent ces algorithmes pour comparer, compléter ou valider leurs résultats. Depuis les années 1990, certaines bases de données, certains programmes sont devenus des références incontournables dans le monde de la biologie.

Un premier aperçu

C'est dans ce cadre que s'inscrit cette thèse. En effet, son objet est de développer de nouveaux algorithmes afin de mettre en évidence des segments d'ADN, d'ARN, ou d'acides aminés, qui sont susceptibles d'avoir un rôle biologique. La prédiction de zones appelées promoteurs ([107] présente un aperçu des techniques biologiques de prédiction des promoteurs chez les eucaryotes), et de manière plus générale la prédiction de sites dits de régulation est un des enjeux de la biologie, mais également de la bioinformatique. La connaissance de ces zones permet de mieux comprendre le fonctionnement des cellules de leur naissance à leur mort, et permet parfois d'expliquer certaines anomalies et certains dysfonctionnements cellulaires. La compréhension des mécanismes de vie des cellules peut alors parfois mener à l'élaboration de procédés biochimiques visant à modifier le comportement des cellules. Il est possible de vérifier la pertinence des prédictions, soit par le biais de statistiques [22], soit par expérimentation [114].

Au cours de ce travail de thèse, plusieurs algorithmes ont été étudiés ou imaginés. Trois grands axes de recherche ont été explorés : la conception d'algorithmes d'extraction de motifs présents dans des séquences biologiques, l'analyse statistique d'une nouvelle famille de fonctions de score utilisables comme heuristiques dans de tels algorithmes, et l'étude ainsi que l'amélioration d'une structure d'indexation de séquences. Ces trois axes s'intègrent tous dans le contexte de l'extraction de motifs communs dans un ensemble de séquences. Les motifs à extraire peuvent être vus comme de petits fragments de séquences, inconnus au départ, ayant de grandes similitudes. La première difficulté consiste donc à définir la ou les notions de similitudes. La seconde est de quantifier cette notion. Ces difficultés gérées, deux hypothèses se profilent : soit chaque séquence comporte au moins un motif, ou bien au contraire, certaines séquences peuvent ne pas en comporter. Le cas échéant, combien de séquences au minimum doivent comporter au moins un de ces motifs ? Il apparaît clairement que la première hypothèse est un cas particulier de la seconde. Le problème le plus général s'intitule donc : « extraction de motifs communs sous contrainte de quorum dans un ensemble de séquences » – abrégé par EM_q dans le manuscrit, la contrainte de quorum correspondant au pourcentage de séquences devant comporter le ou les motifs. Le problème le plus restreint correspond donc au cas particulier où la contrainte de quorum est 100%. Il existe des méthodes de résolution du problème EM_{100} intrinsèquement liées aux propriétés qu'apportent cette restriction. Il s'intitule : « extraction de motifs communs à un ensemble de séquences » – dénoté EM (*i.e.*, en l'absence de précision sur la contrainte de quorum, celui-ci est supposé égal à 100%). Qu'il s'agisse du problème général ou du cas particulier, il n'existe pas d'algorithme générique efficace. En effet, la notion de similitude entre motifs est traduite par une mesure de la similarité entre eux ; or les algorithmes efficaces pour une certaine définition de la similarité entre motifs ne le sont plus nécessairement lorsque cette définition change, voire ils deviennent inapplicables. De plus, l'utilisation de ces algorithmes nécessite souvent beaucoup d'informations *a priori* sur le type de motifs à extraire. Par exemple, certains algorithmes exigent en entrée de fixer la taille des motifs à extraire. D'autres peuvent imposer qu'il n'y ait pas de trous dans les motifs, ou bien n'accordent aucune importance à la forme (longueur, disposition, ...) de ces trous dans les motifs. Il est évident que plus la définition des motifs à extraire est précise, plus

cela permet de restreindre l'espace de recherche des solutions. Pourtant, il est rare que les utilisateurs de ces outils disposent de beaucoup d'informations *a priori* sur les motifs à extraire. Un autre inconvénient de certains outils réside dans la difficulté à fixer les paramètres en entrée, parce qu'ils ne sont pas explicites, ou parce qu'ils sont trop nombreux. Enfin, certaines méthodes renvoient trop de réponses, rendant leur résultats inexploitables. Ainsi, après avoir étudié bon nombre de ces algorithmes, le choix a été fait d'élaborer des méthodes offrant un maximum de souplesse quant au type de motifs à extraire, nécessitant le moins de paramètres possibles en entrées, lesquels devant être le plus aisé possible à fixer.

Pour chacun des problèmes d'extraction de motifs communs à un ensemble de séquences, sous contrainte de quorum ou sans, un algorithme a été conçu, en essayant de respecter ce choix. Les deux algorithmes autorisent l'utilisation de plusieurs mesures de la similarité (fonctions de scores variées et matrices de similarité). Ces outils ont été testés sur des jeux de séquences générées aléatoirement afin d'évaluer statistiquement leur pertinence. Ils ont également été testés sur des jeux de séquences d'ADN contenant des sites de régulation validés expérimentalement. Les résultats obtenus avec STAMS (problème EM) pour plusieurs fonctions de score sont encourageants, et ont mené à l'étude des propriétés statistiques de toute une famille de fonctions de score. Les résultats de cette étude ont permis de fournir un moyen de calculer la moyenne et la variance de ces fonctions en temps constant, et de mettre en évidence le caractère gaussien de leurs distributions. Ainsi, quelle que soit la fonction de score issue de cette famille, il est possible de lui associer un score « normalisé » appelé *Z-Score*. La valeur de ce score permet alors de déduire la probabilité d'obtenir un score normalisé au moins aussi élevé que cette valeur à partir de motifs générés aléatoirement. Cette probabilité est également connue sous le nom de P-valeur. Ce résultat est à la base du second algorithme : StatiSTAMS, lequel répond au problème EM_q .

Enfin, dans un souci d'optimisation du temps de calcul, il a été envisagé d'utiliser une structure d'indexation de motifs. Le principal objectif d'une telle structure est de déterminer « rapidement » la (ou les) occurrence(s) d'un motif dans une séquence. Parmi les structures existantes, l'« Oracle » introduit en 1999 par ALLAUZEN *& al.* a semblé le plus séduisant. Cette structure présente deux énormes avantages en comparaison des autres structures d'indexation existantes. Tout d'abord, l'algorithme est simple et rapide à comprendre et à implémenter. Ensuite et surtout, c'est la structure qui, en l'état actuel des choses, nécessite le moins de ressources pour sa construction, tant du point de vue mémoire que du point de vue temps d'exécution. Néanmoins, il pose un inconvénient que les autres structures n'ont pas : l'Oracle est un outil de reconnaissance faible de motifs, *i.e.*, l'Oracle d'un mot donné reconnaît au moins les facteurs de ce mot. En effet, lorsque l'Oracle d'une séquence reconnaît un motif comme appartenant à la séquence, il se peut que cela soit faux, et que le motif n'apparaisse pas. Toutefois, lorsque l'Oracle rejette un motif, alors c'est que le motif n'apparaît réellement pas dans la séquence. Aussi, avant de pouvoir utiliser cette structure a-t-il fallu caractériser le langage reconnu pour un mot donné. Fort de ce résultat, il apparaît que le nombre de mots acceptés à tort par l'Oracle est trop important pour l'utiliser en l'état. C'est pourquoi une légère modification a été apportée à la structure initiale, le rendant beaucoup plus fiable selon les premières expérimentations. Une étude statistique portant sur les propriétés des Oracles (initiaux et modifiés) est en cours afin de confirmer les observations expérimentales. Les résultats de cette étude s'avèrent nécessaires avant de pouvoir utiliser ces structures à bon escient dans l'un ou l'autre des algorithmes présentés dans ce manuscrit.

Organisation du manuscrit

Les problèmes traités durant toute la durée de la thèse sont issus de la communauté des biologistes. Avant d'essayer d'apporter une réponse informatique, il est donc nécessaire d'étudier les différentes for-

malisations existantes, et au besoin d'en proposer de nouvelles. De l'étude de ces formalisations découle généralement l'étude des solutions existantes, et la spécificité des problèmes posés par la biologie fait que les méthodes informatiques se proposant de les résoudre sont généralement soit trop génériques pour être efficaces, soit trop spécifiques (et difficiles à appréhender) pour être utilisées à bon escient. L'objectif de ce travail a donc été d'essayer de formaliser correctement le problème de l'extraction de motifs dans les séquences biologiques (tout en restant en adéquation avec la réalité biologique), puis d'élaborer de nouvelles solutions apportant une réponse satisfaisante, tant du point de vue informatique que du point de vue biologique.

Le premier chapitre est un rappel succinct des bases de biologie et de biochimie. Dans ce chapitre sont présentés les objets biologiques étudiés dans la suite, ainsi que quelques problématiques s'y rapportant. La formalisation de ces problèmes est donnée au second chapitre. Plusieurs approches y sont également détaillées. Cela permet d'introduire le chapitre suivant. Celui-ci aborde le problème de l'extraction de motifs communs à un ensemble de séquences biologiques de manière détaillée et propose un premier algorithme probabiliste polynomial permettant d'y répondre. Cet algorithme introduit une nouvelle famille de fonctions de score ; lesquelles seront l'objet d'une étude statistique dans le chapitre 4. Cette étude, basée sur l'emploi de séries génératrices, est le fondement d'un nouvel algorithme déterministe également polynomial répondant au problème de l'extraction de motifs communs sous contrainte de quorum dans un ensemble de séquences. Après avoir présenté ces deux algorithmes, la structure de données particulière appelée « Oracle » est étudiée au chapitre 5, avec comme objectif d'optimiser les phases de calcul des deux algorithmes précédents. Le contenu de ce chapitre apporte plusieurs connaissances supplémentaires au sujet de cette structure, ainsi qu'une version légèrement modifiée aux résultats très prometteurs. Enfin, après la synthèse des contributions présentées dans ce manuscrit, plusieurs problèmes et pistes de recherche seront proposés.

Nonobstant le fait que le présent manuscrit ait comme objectif de présenter les résultats obtenus durant la période de la thèse, il a été rédigé avec la volonté que le lecteur y trouve également un support didactique. Les annexes proposées à la fin du document ont également été élaborées dans ce sens.

CHAPITRE 1

Des molécules et des lettres

It is these chromosomes, or probably only an axial skeleton fibre of what we actually see under the microscope as the chromosome, that contain in some kind of code-script the entire pattern of the individual's future development and of its functioning in the mature state [···]. The chromosome structures are at the same time instrumental in bringing about the development they foreshadow. They are law-code and executive power – or, to use another simile, they are architect's plan and builder's craft – in one.

— Erwin SCHRÖDINGER [1887–1961], What is life ? (1944).

Ce chapitre constitue une introduction sommaire à la biologie (principalement dans le monde des eucaryotes). Après avoir présenté différents types de séquences biologiques du point de vue de la biochimie, ainsi que l'importance de ces séquences dans les mécanismes cellulaires, seront présentés quelques organismes vivants couramment utilisés dans le domaine de la biologie et de la bioinformatique. Alors seulement sera abordé le problème du choix de la représentation textuelle des séquences biologiques. Un premier parallèle sera ainsi établi entre les problèmes posés par la biologie et les problèmes d'analyse des textes.

1.1 Du génome au protéome

Chaque individu, de quelque espèce qu'il soit, possède ses caractéristiques propres, d'où la notion même d'individu. Le classement en espèce a longtemps posé problème, et le principe accepté aujourd'hui est celui de l'inter-fécondité entre individus (bien qu'il demeure quelques exceptions d'espèces différentes inter-fécondes et de populations non inter-fécondes d'une même espèce). Ce principe est assez logique si l'on considère que la pérennité d'une espèce dépend de sa capacité à transmettre son matériel génétique. Ce matériel génétique, ou *génome* (*cf.* Nomenclature & Biographies), est présent dans toutes les cellules d'un organisme vivant, en leur noyau, sous forme de chromatine (constituant organique des chromosomes). Depuis le milieu du XX$^{\text{ème}}$ siècle, il est connu que le vecteur de l'information génétique est l'acide désoxyribonucléique [80, 74], plus connu sous son acronyme : l'ADN. Cette structure permet (entre autres) la synthèse et la régulation des protéines au sein de chaque cellule, lui permettant ainsi de vivre et de se renouveler.

1.1.1 L'ADN

L'ADN est une *macromolécule, i.e.,* une molécule composée de plusieurs molécules. Les constituants de base de l'ADN sont les *nucléotides*. Ils sont au nombre de quatre et se distinguent selon la *base azotée* (ou *base*) qui les compose. Il est possible de classer les bases en deux catégories du point de vue de leur structure bio-chimique (*cf.* Figure 1.1) :

les bases puriques, ou *Purines*.	les bases pyrimidiques, ou *Pyrimidines*

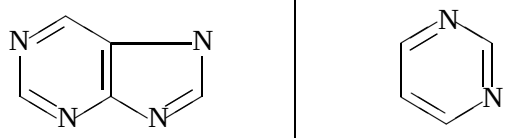


Figure 1.1 – Structure bio-chimique des nucléotides.

Les purines utilisées dans le codage de l'ADN sont l'*Adénine*, et la *Guanine*. Quant aux pyrimidines, il s'agit de la *Cytosine* et de la *Thymine*.

Liaisons Faibles

Les noyaux purines et pyrimidines sont complémentaires. Leurs propriétés physico-chimiques respectives permettent leur appariement, via des liaisons hydrogène, également connues sous le nom de liaisons WATSON & CRICK. Plus précisément, sont complémentaires l'Adénine et la Thymine, ainsi que la Cytosine et la Guanine. Ce sont ces propriétés qui confèrent à l'ADN sa structure particulière, appelée « double hélice », dont la découverte [44, 133] valut à WATSON et CRICK d'obtenir le prix NOBEL en 1962. La double hélice est constituée de deux macromolécules (ou *brins*) d'ADN complémentaires enlacés en spirale (*cf.* Figure 1.2).

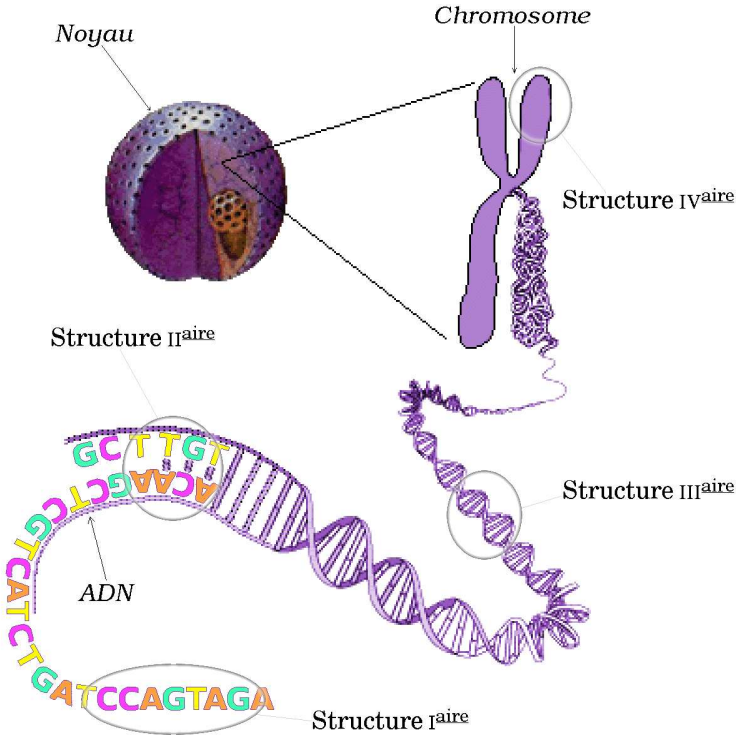


Figure 1.2 – Structures I^{aire}, II^{aire}, III^{aire} et IV^{aire} de l'ADN.

Liaisons Fortes

Si les Liaisons Faibles sont responsables de l'appariement des bases, les Liaisons Fortes sont responsables de l'agencement des bases sous forme de macromolécule. En effet, elles permettent de constituer le squelette, *i.e.*, la partie structurelle, de chacun des brins d'ADN de la double hélice. Ce squelette est semblable à un fil, il n'est pas ramifié, et permet donc de définir les structures primaire et secondaire de la macromolécule (la structure primaire est la représentation en une dimension de la macromolécule, la structure secondaire représente la macromolécule en deux dimensions, et enfin les structures tertiaire et quaternaire sont les représentations en trois dimensions avec un niveau de détail respectif de l'ordre de la macromolécule et du composant organique). Ces liaisons confèrent aux brins d'ADN une polarité, une orientation, dénotée 5'–3' (en raison de la position des atomes de carbone portant les liaisons fortes). Chaque brin possédant ainsi une orientation, deux brins complémentaires sont orientés en sens

contraire. L'appariement des brins correspond à la structure secondaire de cette macromolécule qu'est l'ADN, la double hélice correspond à sa structure tertiaire, sa structure quaternaire étant le chromosome. Ces différentes structures sont illustrées par la figure 1.2.

1.1.2 L'ARN

Les molécules d'ARN diffèrent des molécules d'ADN en quelques points seulement. Principalement, l'ARN est constitué d'une seule chaîne de nucléotides, plutôt que d'une chaîne double. En outre, la Thymine est remplacée par l'*Uracile*, qui est également une pyrimidine. La structure bio-chimique de l'Uracile est très proche de celle de la Thymine, ce qui lui permet de s'apparier à l'Adénine. Il existe plusieurs type d'ADN ; la distinction est faite selon le rôle de la molécule. Il est question d'ARN messager (ARN_m), d'ARN de transfert (ARN_t), ...

De même que l'ADN correspond au support du code génétique, l'ARN peut être décrit comme étant une copie locale de ce code. En effet, l'ARN est une réplication de l'ADN. Rappelons que l'ADN possède une structure en double hélice, dans laquelle les brins sont complémentaires. Par conséquent, la connaissance de l'un des deux brins, apporte la connaissance du second. Ce même principe de complémentarité est présent dans l'ARN. En effet, l'ARN est synthétisé à partir du complémentaire d'un brin d'ADN, en substituant juste l'uracile à la thymine. Ce phénomène est appelé la *transcription*. Cette copie est ensuite *épissée* (*i.e.*, certaines portions du brin d'ARN sont enlevées). Si le rôle de ce brin d'ARN porte l'information sur la synthèse d'une protéine, il est appelé ARN_m et est *traduit* sous forme de protéine (*cf.* Section 1.1.3 – page suivante). La figure 1.3 illustre les mécanismes de transcription, d'épissage et de traduction.

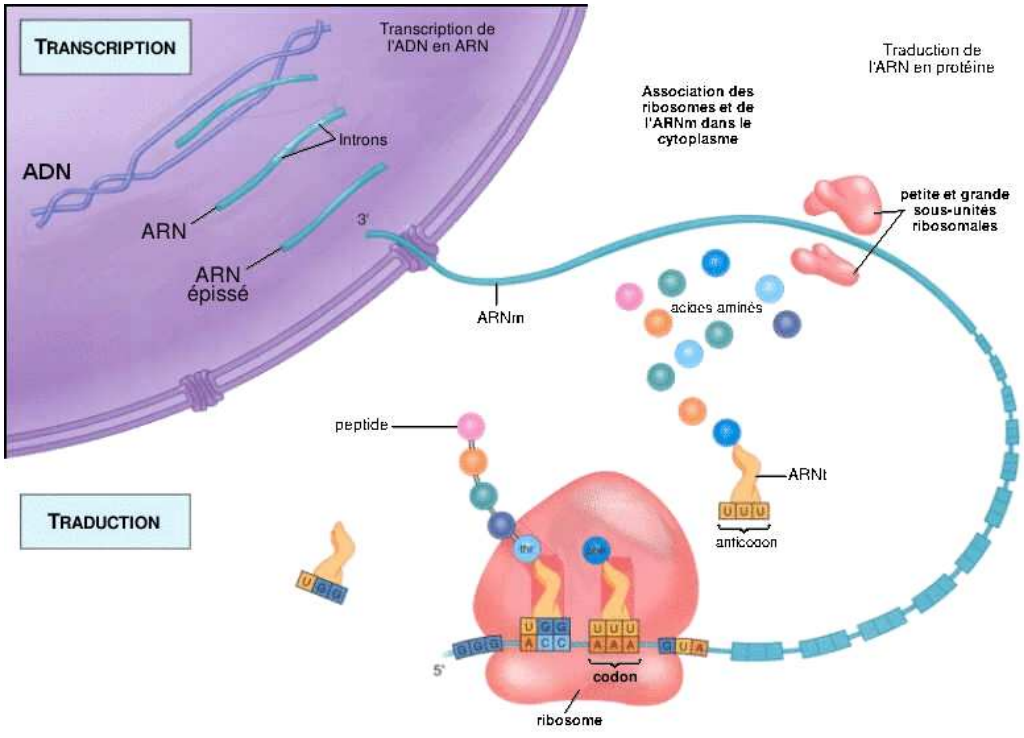


Figure 1.3 – Principe de la synthèse protéique.

Comme pour l'ADN, plusieurs niveaux de représentations de l'ARN sont possibles (*cf.* Figure 1.4).

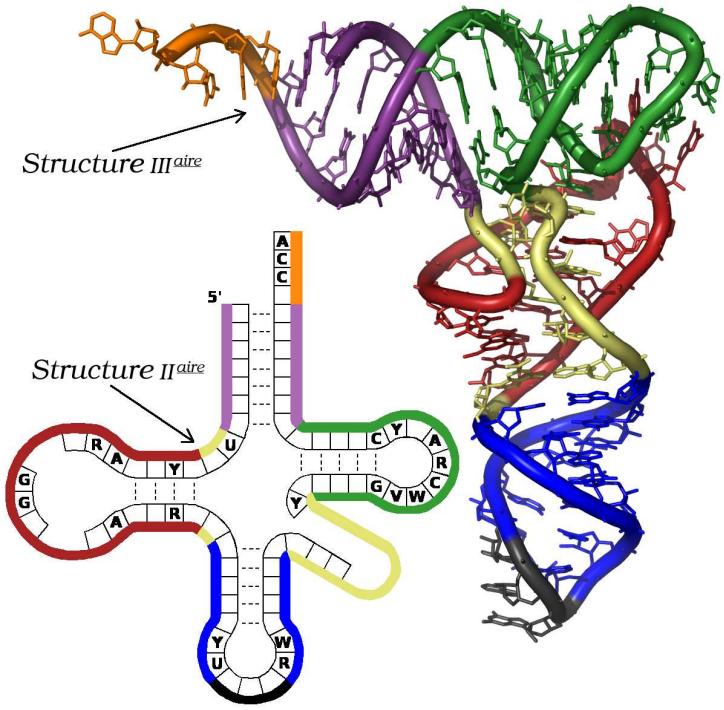


Figure 1.4 – Structures II$^{\text{aire}}$ et III$^{\text{aire}}$ de l'ARN$_t$.

1.1.3 Les Protéines

Les protéines sont aussi des macromolécules. Elles sont synthétisées par les cellules à partir de l'information contenue dans le génome et véhiculée par l'ARN. Les constituants de base des protéines sont les *acides aminés*.

Les acides aminés

À ce jour, 22 sont connus (*cf.* Annexe A.3). Ils sont obtenus à partir de l'information génomique (donc de l'ADN transcrit en ARN) par un procédé, dit de *traduction* (*cf.* Figure 1.3).

En effet, les acides aminés sont codés directement à partir des bases de l'ARN, par groupement de trois. Chaque groupement de trois bases, appelé *codon*, détermine soit un acide aminé, soit la fin de la transcription, et plusieurs codons peuvent engendrer le même acide aminé (*cf.* Annexe A.2). De fait, il y a $4^3 = 64$ codons différents. Ainsi, s'il est possible, à partir d'un brin d'ADN, de déduire son complémentaire, il n'en va pas de même concernant la relation entre les séquences d'ARN et les séquences d'acides aminés. En effet, il n'est pas possible de reconstituer intégralement la séquence d'ARN source de la transcription d'une protéine donnée.

En outre, si le génome d'un individu est immuable tout au long de sa vie et présent dans toutes les cellules de son organisme, le protéome (*cf.* Nomenclature & Biographies) lui, varie en fonction des conditions de vie de l'organisme, et n'est pas intégralement présent dans toutes les cellules.

Comme l'illustre la figure 1.3, la structure tridimensionnelle d'une protéine joue un rôle important dans la vie cellulaire. Il est d'usage de distinguer les acides aminés ayant un rôle structurel dans la conformation de la protéine des acides aminés ayant un rôle fonctionnel (*i.e.*, qui peuvent agir par contact avec d'autres molécules).

Facteurs de transcriptions

Selon les besoins, une cellule va être capable de synthétiser (*i.e.*,, fabriquer) certaines protéines, ou bien au contraire, inhiber leur synthèse. Ce mécanisme de régulation est effectué en amont de la synthèse, à savoir au niveau de l'ADN. En effet, en plus de contenir l'information nécessaire à la synthèse protéique, certaines portions de l'ADN permettent d'activer, ou bien d'inhiber la transcription. Il est alors question de sites de liaisons ADN-protéines spécifiques aux *facteurs de transcriptions*. Ces derniers sont des protéines qui agissent comme des interrupteurs. Les sites de liaisons sont situées en amont des *gènes* ; les *gènes* étant l'information. La figure 1.5 illustre la fixation de la protéine *LexA Repressor* sur le double site appelé *recA operator* [26].

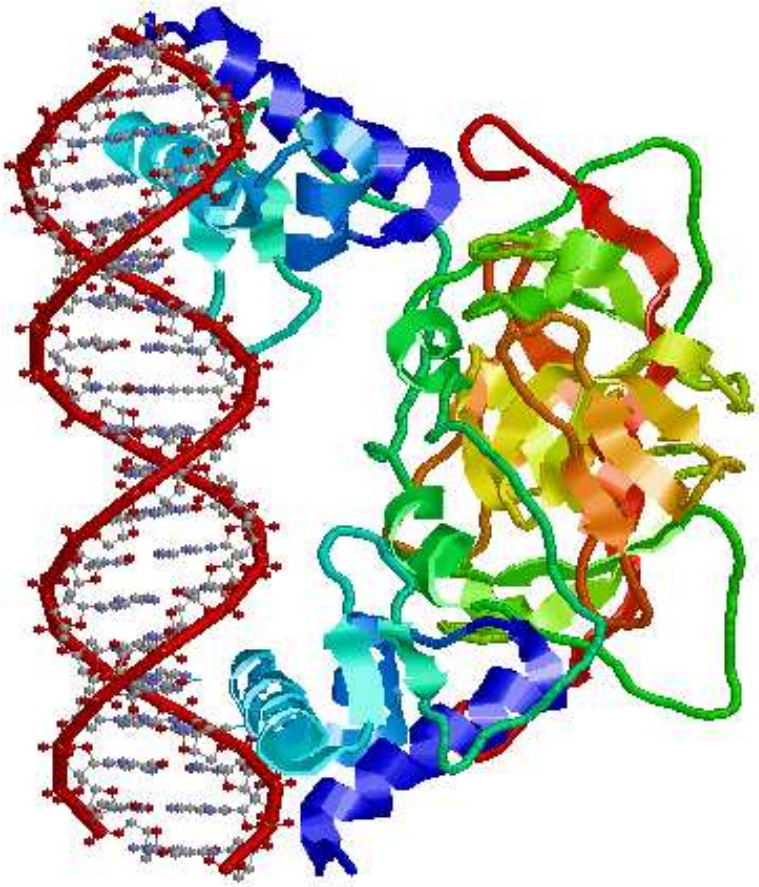


Figure 1.5 – Exemple de liaison ADN/protéine.

La classification [89] effectuée en 1735 par Carl VON LINNÉ (ou LINNÉ) (1707–1778) concernait les plantes, et avait pour objectif premier la production d'un « guide » des espèces. Dans un premier temps, il effectua sa classification sur des critères d'apparence. Il semble que LINNÉE considérait la nature comme figée. Aussi il établit sa classification afin de standardiser la nomenclature. Plus tard, il adopta comme critère les modes de reproduction des espèces. Sa classification fût ensuite reprise et adaptée aux autres organismes vivant. La hiérarchie retenue actuellement est structurée[1] en *règne*, *domaine*, *phylum*, [*sous-phylum*,] *classe*, *ordre*, [*sous-ordre*,] *famille*, *genre* et enfin en *espèce* [*sous-espèce*]. La langue adoptée fût (et demeure) le latin, qui était la langue internationale de la science au XVIII$^{\text{ème}}$ siècle (*cf.* [24] pour de plus amples détails sur l'apparition des systèmes de classification). Ainsi, tous les lecteurs de ce manuscrit appartiennent à la branche des *Eucarya Animalia Chordata [Vertebrata] Mammalia Primates [Anthropoidea] Hominidæ Homo sapiens [sapiens]*.

1.2.1 Les origines de l'évolution

Chaque espèce possède son propre code génétique, ses propres structures moléculaires. Mais au sein même d'une même espèce, il existe aussi beaucoup de différences. Celles-ci sont dues à plusieurs facteurs. Sans être exhaustif, citons la reproduction, qui permet un brassage du matériel génétique, ainsi que les étapes de transcription et de traduction (*cf.* Section 1.1.2 – page 13). En effet, lors de ces étapes, il peut se produire des erreurs, et il arrive que celles-ci n'engendrent pas (ou peu) de conséquences sur les molécules produites. Ces erreurs, lorsqu'elles sont bénignes peuvent être transmises, et contribuent à la diversité des espèces, et à l'évolution. Dans le cas contraire, les molécules produites sont généralement détruites par la cellule. Ces phénomènes sont également à l'origine de diverses pathologies.

Dans le cas de l'ADN, de l'ARN et des protéines, ces erreurs sont appelées *mutations*. Il est possible de distinguer trois types de mutations : les *substitutions*, les *suppressions*, et les *insertions* de bases. En général, les substitutions sont moins dommageables que les insertions ou les suppressions [56]. Elles sont aussi les plus nombreuses ; et parmi les substitutions, il est assez fréquent que celles-ci conservent les propriétés physico-chimiques de la macromolécule. Il arrive néanmoins que de petites mutations aient des conséquences importantes. Citons pour exemple le cas de la mucoviscidose [56] ; la mutation la plus fréquente[2] est une suppression de trois nucléotides aboutissant à l'élimination de la Phénylalanine en position 508 de la protéine CFTR (de l'anglais « Cystic Fibrosis Transmembrane Conductance Regulator »).

Les mutations se produisent principalement lors de la fabrication d'un organisme (protéine, cellules, ...). Quand cela se produit, le plus souvent, la mutation induit alors la destruction de cet organisme, par des mécanismes de retro-contrôle (ce phénomène est extrêmement fréquent). Ces mutations sont alors qualifiées comme étant *non-viable*. Cependant, les mutations qui portent sur les 85% de gênes « inutiles », généralement désignées comme des régions non codantes, sont souvent viables. Ce qui induit que dans l'ADN, les régions codantes[3] pour des fonctions importantes (voire vitales) pour l'organisme sont souvent beaucoup mieux conservées d'un individu à l'autre que des régions non codantes...

[1] Les niveaux entre crochets correspondent à des subdivisions de « transitions ».
[2] Cette mutation concerne 70% des sujets atteints et est appelée DF508.
[3] Par opposition aux régions non codantes.

1.2.2 *E. coli* et autres organismes

L'étude du génome et du protéome des organismes permet d'appréhender le fonctionnement de certaines maladies. De plus en plus d'organismes sont séquencés, notamment dans le règne des procaryotes (organismes unicellulaires sans noyau individualisé). En effet, leur génome est plus petit que celui des eucaryotes (organismes unicellulaires ou multicellulaires à noyau individualisé). En particulier, citons la bactérie *Escherichia coli* (*cf.* Figure 1.6). Il s'agit de parasites du tube digestif présents chez l'homme et chez l'animal. Ces bactéries sont le plus souvent inoffensives, mais ont la faculté de muter et d'acquérir naturellement des gènes les rendant virulentes pour les différentes espèces qui leur servent d'hôtes. Ces phénomènes permettent l'émergence régulière de nouvelles variétés de bacilles pathogènes adaptés à leurs hôtes et responsables de pathologies spécifiques [116] (diarrhées, infections urinaires, septicémies, méningites, ...). Cette bactérie est l'une des plus étudiées, et est souvent utilisée comme modèle afin de vérifier la pertinence de tel ou tel algorithme. D'autres organismes sont également étudiés et utilisés comme références, tels que la levure *Saccharomyces cerevisiæ* (*cf.* Figure 1.6), le nématode transparent *Cænorhabditis elegans*, la mauvaise herbe *Arabidopsis thaliana*, le diptère *Drosophila melanogaster* ou encore la souris *Mus musculus*. Tous ces organismes offrent l'avantage d'être faciles à reproduire et en grand nombre. En outre, leur cycle de vie est bref et leur génome est de petite taille (à l'exception de la souris).

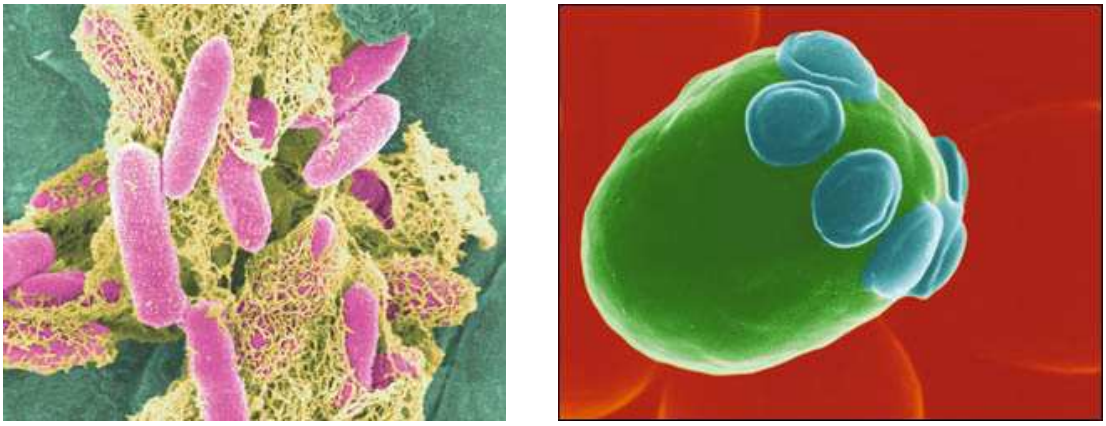


Figure 1.6 – Vues de *Escherichia coli* (à gauche) et de *Saccharomyces cerevisiæ* (à droite).

1.3 De la compréhension des macromolécules au problèmes de texte

L'analogie entre les macromolécules et du texte est évidente dès que l'on essaie de représenter leurs structures primaires. Un brin d'ADN est une succession orientée (5'–3', *cf.* Section 1.1.1 – page 12) de bases que les biologistes ont pris l'habitude de représenter par leurs initiales. Leur représentation sous forme textuelle (initiales des bases lues de gauche à droite) ne pose donc pas de problème. La représentation des protéines sous forme de texte est plus compliquée. En effet, les acides aminés sont plus nombreux, et certains partagent une même initiale. Toutefois, il est possible d'associer un symbole distinct par acide aminé. La difficulté s'accroît lorsque l'on essaie de représenter une macromolécule dont on ne connaît pas avec précision tous les composants. Le problème devient insurmontable en l'absence de conventions.

1.3.1 Le standard IUPAC

Il a fallu attendre 1970 avant que soit adoptée la nomenclature des acides nucléiques [69], et 1983 pour ce qui est des acides aminés [70] ; dates auxquelles l'INTERNATIONAL UNION OF BIOCHEMISTRY AND MOLECULAR BIOLOGY, conjointement à l'INTERNATIONAL UNION OF PURE AND APPLIED CHEMISTRY ont défini les conventions de représentation des acides nucléiques et aminés (*cf.* Annexe A ; [68]). Ces normes, du fait de leur apparition tardive, ne sont pas toujours respectées. D'autres standards existent (GCG, *staden*, ...), mais n'intègrent pas toujours de notion sémantique dans leurs choix de représentation. En effet, le principe de ces standards est de représenter les macromolécules (ADN, ARN et protéines) sous forme de séquences de lettres ou d'abréviations, en gérant dans la mesure du possible la possibilité de trouver plusieurs acides nucléiques ou aminés à certaines positions données (*e.g.* dans le cas d'une représentation consensuelle d'une macromolécule). La norme IUPAC est la plus complète, la plus documentée et la plus justifiée des trois normes citées ci-dessus. C'est pourquoi les travaux présentés dans ce manuscrit, se basent sur le respect de cette norme.

1.3.2 Problématiques

L'analogie entre les macromolécules et du texte peut être menée plus loin qu'une simple représentation desdites macromolécules. En effet, en reprenant l'exemple des facteurs de transcription (*cf.* Section 1.1.3 – page 15 – et Figure 1.5), la caractérisation des sites de liaisons concerne une information présente au niveau de la structure primaire des séquences d'ADN. De manière générale, les génomes sont très structurés. Deux brins d'ADN codant pour la même fonctionnalité biologique sont structurellement souvent proches. Les recherches sur la structure primaire des protéines sont basées sur les mêmes principes que pour la recherche sur la structure primaire de l'ADN, si ce n'est que l'alphabet est plus étendu. En revanche, les structures tertiaire et quaternaire des protéines revêt un caractère important pour son fonctionnement. Aussi, la recherche peut alors être basée uniquement sur les propriétés physico-chimiques des acides aminés composant une protéine. La représentation textuelle des séquences primaires doit alors pouvoir permettre d'établir ces propriétés. Parmi les problèmes inhérents à l'analyse des séquences biologiques, certains sont exclusivement focalisés sur l'analyse des séquences primaires des macromolécules, et sont transposables à des problèmes d'algorithmique du texte, tels la recherche ou l'extraction de motifs répétés, ou encore l'alignement de séquences.

Extraction de Motifs *vs.* Recherche de Motifs

La recherche de ressemblances entre deux ou plusieurs séquences biologiques permet de mettre en évidence certaines fonctionnalités cellulaires. Plusieurs problèmes apparaissent alors, en fonction des informations disponibles. Selon que la recherche de ressemblance est basée sur la connaissance préalable d'un « motif » dans les séquences, ou au contraire qu'il s'agisse de découvrir un ou plusieurs « motifs » ressemblants dans les séquences, il est possible de distinguer deux problématiques complémentaires. Le premier problème, appelé recherche de motifs dans un ensemble de séquences, consiste à rechercher dans les séquences les positions d'un (ou plusieurs) motif « ressemblants » aux motifs fournis *a priori*. La notion de ressemblance traduit les phénomènes biologiques d'altération des séquences (*cf.* Section 1.2.1 – page 16). Le second problème, appelé « extraction de motifs dans un ensemble de séquences », consiste à extraire le (ou les) motif(s) commun(s) (*i.e.,* qui se ressemblent ou qui ressemblent à un motif consensuel) à l'ensemble de séquences, sans connaissance précise de ce(s) motif(s). Dans les deux cas, la première difficulté rencontrée lors de l'élaboration de méthodes de résolution de ces problèmes, est de définir

l'ensemble des critères permettant d'affirmer si deux motifs sont ressemblants ou non. S'en suit généralement le problème de la délimitation de l'espace de recherche, et enfin de l'évaluation de la pertinence des résultats fournis par lesdites méthodes.

Alignement de séquences

La ressemblance de séquences peut également être étudiée au niveau de la séquence entière, et non pas au niveau de quelques motifs. Dans le cas de la recherche ou de l'extraction de motifs, les comparaisons se font localement sur les séquences. Il est question de ressemblance (ou similitude) locale. Dans le cas de l'étude, non plus de la présence d'un motif biologique particulier, mais d'une structure globale commune à plusieurs séquences, il est alors question de ressemblance globale. La majorité des méthodes recherchant des similitudes globales entre deux ou plusieurs séquences est basée sur un mécanisme d'alignement des séquences (cette notion est reprise et illustrée dans le chapitre suivant). L'alignement se lit en deux dimensions. Les lignes correspondent aux séquences, tandis que les colonnes correspondent aux positions dans chacune des séquences. Dans une même colonne, deux acides nucléiques ou aminés qui ne se correspondent pas sont alors des substitutions. Selon les choix de représentations, il est possible ou non d'insérer un « trou » dans la colonne, représentant alors une suppression d'un acide dans la séquence (ou une insertion dans les séquences ne contenant pas de trou). La lecture de l'alignement vise à mettre en évidence les parties communes à toutes les séquences. À l'instar des problèmes de la recherche et de l'extraction de motifs, la principale difficulté consiste à modéliser la notion de ressemblance globale, puis à délimiter l'espace des possibles. Ce problème est néanmoins très proche – à plusieurs égards – du problème précédent, et les modélisations de la ressemblance sont souvent les mêmes.

CHAPITRE 2

Exploration des séquences biologiques

*Énumérer les termes « algorismi », « alchorismi », « algoritmi », « algorithmus », « algorithme », « algorithm » est intéressant, certes, mais trouver la loi qui détermine et explique leur succession serait plus fondamental. Cette loi, bien entendu, est elle-même un algorithme. [· · ·] Induire l'algorithme régissant l'évolution du nom d'*AL-KHWÂRÎZMÎ *au fil des siècles [· · ·] permettrait de prévoir dès aujourd'hui la prochaine transformation qui [lui] sera imposée.*

— Didier NORDON, Pour la Science [Bloc-Notes] n° 307 (mai 2003).

Dans le chapitre précédent ont été introduites informellement plusieurs problématiques issues de la biologie, notamment les problèmes de la recherche et de l'extraction de motifs (*cf.* Section 1.3.2 – page 18). Cette dernière problématique est l'essence même du sujet de cette thèse. Dans ce chapitre, les problèmes de la recherche et surtout de l'extraction de motifs dans les séquences primaires d'ADN sont formalisés. Les principales méthodes de résolution de ces problèmes sont sommairement décrites, afin de mettre en évidence leurs avantages et inconvénients respectifs. Sur la base de cette synthèse, la problématique générale de cette thèse est présentée sous ses aspects formels et informels, ainsi qu'une présentation succincte des contributions proposées en vue de répondre au problème de l'extraction de motifs.

L'analyse de séquences biologiques est souvent réalisée *in vitro*, par les chercheurs en biologie moléculaire, en physiologie ou encore en biochimie. L'apparition de l'informatique a permis d'aborder les études sous un aspect plus formel, permettant une étude *in silico*. Certaines théories développées dans le cadre de l'algorithmique du texte trouvent une application pour les problèmes de bioinformatique, au même titre que les techniques d'indexation. Il ne s'agit pas simplement d'appliquer des théories aux problèmes biologiques, mais de les adapter, d'en développer de nouvelles plus spécifiques.

2.1 Définitions préliminaires

Les séquences biologiques sont des macromolécules qui sont considérées comme du texte (*cf.* Section 1.3.1 – page 18). Qu'il s'agisse d'ADN, d'ARN ou de protéines, ces séquences peuvent être modélisées formellement. Ainsi, il est nécessaire de définir quelques notions élémentaires.

2.1.1 Alphabet et facteurs

Définition 2.1. Étant donné un ensemble fini de symboles Σ, appelé alphabet, l'ensemble de toutes les suites (non vides) ordonnées de symboles de l'alphabet qu'il est possible d'engendrer à partir de Σ est défini par $\Sigma^+ = \{x_1 \cdots x_n \,|\, x_i \in \Sigma, 1 \leq i \leq n, n \in \mathbb{N}\}$. Étant donné un entier $k \in \mathbb{N}$, le sous-ensemble de Σ^+ des suites non vides de longueur inférieure ou égales à k est défini par $\Sigma^k = \{x_1 \cdots x_k \,|\, x_i \in \Sigma, 1 \leq i \leq k\}$. La suite vide est dénotée par ε et $\Sigma^+ \cup \{\varepsilon\} = \Sigma^*$.

Pour exemple, dans le cas de l'ADN, l'alphabet considéré est $\Sigma = \{A, C, G, T\}$ (*cf.* Annexe A).

Définition 2.2. Étant donné un alphabet Σ, une suite finie et ordonnée de symboles de Σ est appelée chaîne. Lorsque la chaîne représente une séquence biologique, la dénomination séquence est utilisée. Le terme mot est utilisé pour désigner une chaîne dans un cadre abstrait. Par opposition, le terme motif est utilisé pour exprimer une connotation biologique. Étant données deux chaînes s et f telles que $s = u\,f\,v$ (avec $u, v \in \Sigma^*$), le mot f est appelé facteur de s. Dans ce cas, f est dit présent dans s. L'ensemble des facteurs de s est noté $Fact(s)$.

Définition 2.3. Étant donnée une chaîne $s \in \Sigma^k$ ($k \geq 0$), la longueur de s, notée $|s|$ est égale à k. Étant donné un facteur f de s, tel que $s = u\,f\,v$ (avec $u, v \in \Sigma^*$), la position de f dans s est égale à $|u| + 1$.

Définition 2.4. Étant donnés une chaîne s et un facteur f de $s = u\,f\,v$ (avec $u, v \in \Sigma^*$), f est un préfixe (respectivement suffixe) de s si $|u| = 0$ (respectivement $|v| = 0$). Le facteur (respectivement préfixe, suffixe) f est un facteur (respectivement préfixe, suffixe) propre si f est différent de s et de ε. L'ensemble des préfixes (respectivement suffixes) d'une chaîne s est noté $Pref(s)$ (respectivement $Suff(s)$).

2.1.2 Motifs & Couverture

En biologie, il se peut que certaines substitutions n'aient pas d'importance. Par exemple, la substitution d'une purine par une autre purine, dans un contexte donné, peut n'avoir aucune incidence grave (*cf.* Section 1.2.1 – page 16). Auquel cas, il est souhaitable de pouvoir manipuler des alphabets dits « dégénérés ». C'est la raison pour laquelle les notions de *classes* et de *couverture* d'un alphabet sont introduites.

Définition 2.5. Étant donné un alphabet Σ, une classe est un sous-ensemble non vide de Σ.

Définition 2.6. Étant donnés un alphabet Σ et $C_1, \ldots, C_n$ un ensemble de classes distinctes de Σ, la collection $\{C_1, \ldots, C_n\}$ est appelée une couverture de Σ si $\bigcup_{i=1}^{n} C_i = \Sigma$.

Définition 2.7. Étant donné un alphabet Σ, la couverture $\mathcal{CI}(\Sigma) = \{\{\alpha\} \mid \alpha \in \Sigma\}$ est appelée couverture identité.

À partir de cette dernière définition, il est effectivement possible de représenter un alphabet dégénéré, comme illustré par l'exemple 2.1.

Exemple 2.1 (Alphabets dégénérés & couverture).
Selon le standard IUPAC *(cf. Annexe A), l'ensemble des purines est représenté par le code à trois lettres Pur et le code à une lettre R. Il permet donc de représenter soit l'adénine – représentée par le code Ade ou A, soit la guanine – représentée par le code Gua ou G (cf. Section 1.1.1 – page 11). Pour représenter cette possibilité de dégénérescence de l'alphabet, il suffit d'inclure dans la couverture de l'alphabet les collections $\{R, A\}$ et $\{R, G\}$ – ou $\{Pur, Ade\}$ et $\{Pur, Gua\}$ pour le codage sur trois lettres. Ainsi, la couverture de l'*ADN *permettant de représenter l'alphabet dégénéré présenté dans la table A.1 – page 185 – est elle*

$$\left\{\begin{array}{cccccc} \{A\}, & \{C\}, & \{G\}, & \{T\}, & & \\ \{R,A\}, & \{R,G\}, & \{Y,C\}, & \{Y,T\}, & & \\ \{M,A\}, & \{M,C\}, & \{K,G\}, & \{K,T\}, & & \\ \{S,C\}, & \{S,G\}, & \{W,A\}, & \{W,T\}, & & \\ \{H,A\}, & \{H,C\}, & \{H,T\}, & \{B,C\}, & \{B,G\}, & \{B,T\}, \\ \{V,A\}, & \{V,C\}, & \{V,G\}, & \{D,A\}, & \{D,G\}, & \{D,T\}, \\ & \{N,A\}, & \{N,C\}, & \{N,G\}, & \{N,T\} & \end{array}\right\}.$$

Définition 2.8. Étant donné un alphabet Σ, ainsi qu'une couverture $\mathcal{C}(\Sigma)$, alors deux symboles α et β de l'alphabet sont en « correspondance » s'il existe un ensemble X dans $\mathcal{C}(\Sigma)$ tel que $\alpha, \beta \in X$; cette assertion est notée $\alpha =_{\mathcal{C}(\Sigma)} \beta$ le cas échéant. Dans le cas contraire ils sont dits en « non correspondance », et la notation est alors $\alpha \neq_{\mathcal{C}(\Sigma)} \beta$.

2.1.3 Mutations

Les définitions suivantes permettent de formaliser les trois types de mutations identifiées dans le chapitre précédent (*cf.* Section 1.2.1 – page 16).

Définition 2.9. Étant donnés une séquence s telle que $s = u\,v$ ($u, v \in \Sigma^*$) et un symbole $\alpha \in \Sigma$, l'insertion de α à la position $|u| + 1$ produit la séquence $s' = u\,\alpha\,v$.

Définition 2.10. Étant donnée une séquence s telle que $s = u\,\alpha\,v$ (avec $u, v \in \Sigma^*$ et $\alpha \in \Sigma$), la suppression de α à la position $|u| + 1$ produit la séquence $s' = u\,v$.

Définition 2.11. Étant donnés une séquence s telle que $s = u\,\alpha\,v$ (avec $u, v \in \Sigma^*$ et $\alpha \in \Sigma$) et un symbole $\beta \in \Sigma$, la substitution de α à la position $|u|+1$ par le symbole β produit la séquence $s' = u\,\beta\,v$.

2.2 Notions de similarité et fonctions de score entre deux mots

La mesure de la similarité entre deux motifs est un aspect capital de l'analyse de séquences (biologiques ou autres…). Cette mesure est intrinsèquement liée à l'efficacité des méthodes d'analyses. En effet, une mesure très restrictive de la similarité permettra très vraisemblablement de réduire le temps de calcul d'un algorithme, en contrepartie, cela nécessite d'avoir beaucoup d'informations sur le type de motifs recherchés. *A contrario,* une mesure peu restrictive de l'espace de recherche risque fort d'être responsable d'un temps d'exécution élevé. Notons au passage que si l'on admet comme postulat de départ que plus deux éléments sont similaires, moins ils sont dissimilaires (ce qui semble raisonnable), une mesure de la similarité est également une mesure de la dissimilarité, et réciproquement.

Définition 2.12. Étant donnés un ensemble de n objets de $\mathcal{E}$ à comparer, toute application de $\mathcal{E}^n \to \mathbb{R}$ est un score.

Selon la définition précédente, un score peut être utilisé comme mesure de la similarité entre deux ou plusieurs objets de $\mathcal{E}$.

Les mesures de la similarité sont parfois (voire souvent) utilisées afin de définir un notion de similarité. La notion de similarité permet d'affirmer ou au contraire d'infirmer que deux mots sont similaires. Pour cela, il suffit de définir un score seuil, en deça duquel les paires de motifs évaluées ne sont pas considérées comme similaires. La notion de similarité permet de définir la notion d'occurrence de motifs.

Définition 2.13. Étant donnés un alphabet Σ, une séquence s, une notion de similarité $\Re$, un motif m et un facteur m' de s, le facteur m' est une occurrence de m dans s si et seulement si $m\,\Re\,m'$. En l'absence de notion de similarité explicite, la relation $\Re$ considérée est l'égalité.

2.2.1 Distance & Similarité

Les applications utilisées afin de mesurer la similarité, ou fonction de score, ont été classées par GOLDSTONE [52] en quatre catégories : les applications géométriques, les applications transformationnelles, les applications basées sur la notion d'alignement, et enfin les applications basées sur la notion d'attribut.

Notation 2.14. Les applications données ci-après respectent la notation des similarités et des distances définie dans l'ouvrage de LEGENDRE et LEGENDRE [86]. Les distances sont notées $D_?$ et les mesures de similarité $S_?$. Lorsque le texte en indice est un entier, il s'agit du même numéro que celui figurant dans [86].

Il existe une étroite relation entre les notions de similarité et de distance comme l'indique la remarque suivante.

Remarque 2.15. Il est souvent possible, étant donné une distance D de définir une similarité S par un simple processus de transformation :

$$\begin{aligned} S &\equiv D_{max} - D \qquad \text{où } D_{max} \text{ est une borne supérieure de la distance.} \\ S &\equiv 1/D \\ &\vdots \end{aligned}$$

Applications géométriques :

Rappelons tout d'abord quelques propriétés mathématiques usuelles. Une application $f : \mathcal{E} \times \mathcal{E} \rightarrow \mathbb{R}^+$ vérifie :

- l'axiome de symétrie si et seulement si $f(x, y) = f(y, x)$ pour tout $x, y \in \mathcal{E}$;
- l'axiome de séparation si et seulement si $f(x, y) = 0 \Leftrightarrow x = y$ pour tout $x, y \in \mathcal{E}$;
- l'axiome de l'inégalité triangulaire si et seulement si $f(x, y) \leq f(x, z)+f(z, y)$ pour tout $x, y, z \in \mathcal{E}$.

Définition 2.16 (Espace métrique)**.** Un espace métrique est un ensemble $\mathcal{E}$ non vide tel qu'il existe une application $f : \mathcal{E} \times \mathcal{E} \rightarrow \mathbb{R}^+$ vérifiant les axiomes de symétrie, de séparation et d'inégalité triangulaire. Une telle application est alors une métrique, également appelée distance.

Les mesures de la similarité géométriques sont des métriques. Elles permettent de mesurer la similarité entre deux objets d'un même ensemble lorsque il est possible de les projeter dans un espace métrique.

La distance la plus courante est la distance **euclidienne**, ou chaque élément de $\mathcal{E}$ est projeté dans un espace euclidien à p dimensions :

$$D_1 \ : \ \boldsymbol{x}, \boldsymbol{y} \mapsto \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}, \tag{2.1}$$

où $\boldsymbol{x} = (x_1, \cdots, x_p)$ et $\boldsymbol{y} = (y_1, \cdots, y_p)$ sont les vecteurs associés aux objets x et y.

Citons également la distance de **MANHATTAN**[1] (également connue sous le nom de « *city-bloc* ») :

$$D_7 \ : \ \boldsymbol{x}, \boldsymbol{y} \mapsto \sum_{i=1}^{p} |x_i - y_i|. \tag{2.2}$$

Ces deux distances sont des cas particuliers de la distance de **MINKOWKSI** d'ordre m :

$$D_6 \ : \ \boldsymbol{x}, \boldsymbol{y} \mapsto \sqrt[m]{\left(\sum_{i=1}^{p} |x_i - y_i|^m \right)}. \tag{2.3}$$

[1] Cette distance tient son nom de l'analogie avec une ballade dans MANHATTAN.

Applications transformationnelles :

Une opération de transformation est une application $f : \mathcal{E} \rightarrow \mathcal{E}$. Il est possible d'associer un coût à chaque opération de transformation. Ainsi la somme des coûts des opérations nécessaires pour transformer un élément d'un ensemble $\mathcal{E}$ vers un autre élément de cet ensemble permet de déterminer le coût global de la transformation. Les applications transformationnelles sont les applications qui renvoient le coût d'une séquence d'opérations permettant de transformer un élément de $\mathcal{E}$ en un élément de $\mathcal{E}$. Le plus petit coût de transformation d'un élément de $\mathcal{E}$ vers un autre élément de $\mathcal{E}$ représente alors la plus petite « distance » entre ces deux éléments. Les applications géométriques suivent donc le principe de parcimonie défini par Guillaume d'OCKHAM au XIV$^{\text{ème}}$ siècle : « *pluralitas non est ponenda sine necessitate* » (*les pluralités ne doivent être posées sans nécessité*).

Cette catégorie d'application comporte notamment les distances de HAMMING [62], INDEL [88], de **LEVENSHTEIN** [88] (également connue sous le nom de distance d'**édition**) et de NEEDLEMAN-WUNSCH [104].

Définition 2.17. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$ de même longueur, la distance de HAMMING est le nombre minimum de substitutions nécessaires pour transformer w_1 en w_2. Cette distance est notée $D_H(w_1, w_2)$.

Définition 2.18. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$, la distance INDEL est le nombre minimum d'insertions et de suppressions (en anglais *deletion*) nécessaires pour transformer w_1 en w_2. Cette distance est notée $D_{I/D}(w_1, w_2)$.

Définition 2.19. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$, la distance de LEVENSHTEIN est le nombre minimum de mutations nécessaires pour transformer w_1 en w_2. Cette distance est notée $D_L(w_1, w_2)$.

Définition 2.20. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$, la distance de NEEDLEMAN-WUNSCH est la distance de LEVENSHTEIN, où le coût des mutations est pondéré. Ainsi, cette distance correspond au coût minimum des opérations permettant de transformer w_1 en w_2. Cette distance est notée $D_{NW}(w_1, w_2)$.

L'utilisation de l'une de ces quatre distances dans le calcul d'un score de similarité confère généralement à ce score une loi de distribution dite « des valeurs extrêmes » [6] (*cf.* Annexe E.3). Ces lois sont également connues sous le nom de lois de « GUMBEL » [58, 59, 60].

Ces distances, bien qu'étant des mesures transformationnelles, peuvent toutefois se calculer en utilisant la distance de MANHATTAN. Par conséquent, elles peuvent également être classées dans la catégorie des mesures géométriques.

Applications basées sur la notion d'alignement :

Ces applications permettent d'exprimer les similarités au niveau structurel, en appliquant un principe de « superposition » des objets comparés. Lorsque les objets étudiés sont des mots, leur superposition est effectuée en les alignant (*cf.* Section 1.3.2 – page 19).

La mesure de l'**entropie** définie par SHANNON en 1948 [121] illustre cette catégorie. Cette mesure permet d'évaluer la pertinence d'un motif en fonction du contexte. Elle fût développée afin d'optimiser les transmissions de signaux discrets (*e.g.* le télégraphe) en adaptant leurs codages en fonction de la fréquence d'émission des symboles composant lesdits signaux. Cette mesure est la différence entre l'information portée *a priori* par chaque lettre d'un mot pour chacune des positions (*i.e.*, en fonction

de la probabilité d'apparition des symboles de l'alphabet pour chaque position) et l'information portée *a posteriori* pour cette position (*i.e.*, en fonction de la fréquence observée des symbole de l'alphabet pour chaque position). D'où son autre appellation de mesure du « contenu d'information ». En effet, en considérant le cas de l'ADN (alphabet à quatre symboles), il faut *a priori* au moins deux *bits* pour coder chaque symbole de manière unique. Chaque *bit* peut être considéré comme une question dont la réponse (1 pour *oui*, 0 pour *non*) permet de définir les possibilités d'être de chacun des symboles – ce procédé dichotomique permet ainsi d'éliminer une moitié des symboles encore disponibles à chaque nouvelle réponse, d'où leur dénomination de *bits* d'information. Si la fréquence d'apparition d'un symbole particulier (*e.g.* A) est très supérieure à la fréquence d'apparition des autres symboles (information *a posteriori*), il peut en pratique s'avérer plus rentable (en nombre de *bits* émis) de ne coder le symbole le plus fréquent que sur un seul *bit*, un autre sur deux *bits* et les deux autres sur trois *bits*. La théorie développée par SHANNON a pour objectif de déterminer le nombre de *bits* d'information portés *a posteriori* par chaque symbole, et de calculer ainsi le gain d'information qu'il est possible de réaliser en utilisant un tel codage.

Soit $H_{prior}(i)$ l'information nécessaire *a priori* pour coder (en binaire) le symbole situé à la position i dans un mot. Sans aucune autre information à disposition, alors :

$$\begin{aligned} H_{prior}(i) &= \log_2 |\Sigma| \\ &= -\log_2(\frac{1}{|\Sigma|}). \end{aligned} \tag{2.4}$$

Si la distribution des symboles est uniforme ($\forall\, \alpha \in \Sigma, p_\alpha^i = p = \frac{1}{|\Sigma|}$, avec p_α^i la probabilité d'obtenir le symbole α à la position i), alors

$$H_{prior}(i) = -\log_2 p. \tag{2.5}$$

Si la distribution n'est pas uniforme, alors SHANNON a montré [121] que

$$H_{prior}(i) = -\frac{1}{|\Sigma|} \sum_{\alpha \in \Sigma} \log_2 p_\alpha^i. \tag{2.6}$$

Afin de mesurer l'entropie, il est également nécessaire de connaître l'information portée *a posteriori* par un symbole à une position donnée. Pour une position i, cette information – notée $H_{post}(i)$ – est calculée par rapport à une expérience passée, *i.e.*, à partir d'une collection de mots. Soit $\{w_1, \cdots, w_t\}$ une collection de mots de longueur au moins égale à i, SHANNON a également montré que

$$\begin{aligned} H_{post}(i) &= \frac{1}{t} \sum_{j=1}^{t} -\log_2 p_{w_j[i]}^i \\ &= -\sum_{\alpha \in \Sigma} f_i(\alpha)\ \log_2 p_\alpha^i, \end{aligned} \tag{2.7}$$

$f_i(\alpha)$ étant la fréquence du symbole α à la position i des mots de $\{w_1, \cdots, w_t\}$.

La différence entre l'information portée *a priori* et celle portée *a posteriori* pour chaque position traduit le gain de codage binaire qui aurait pu être effectué pour représenter la collection $\{w_1, \cdots, w_t\}$. Ce gain pour la position i est donné par :

$$R(i) = H_{prior}(i) - H_{post}(i). \tag{2.8}$$

Il est possible d'approcher les p_α^i, par les fréquences des symboles α indépendamment de leur positions (p_α) dans la collection $\{w_1, \cdots, w_t\}$.

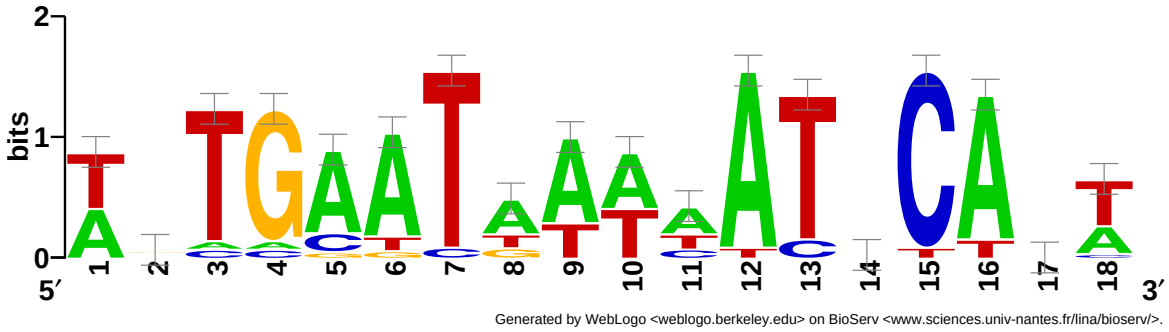
Auquel cas, le gain d'information devient :

$$R(i) = H_{prior}(i) - H_{post}(i), \tag{2.9}$$

$$\text{avec} \quad H_{post}(i) = -\sum_{\alpha \in \Sigma} f_i(\alpha) \log_2 p_\alpha. \tag{2.10}$$

Cette mesure est utilisée dans l'outil de visualisation d'alignement « *Sequence Logo* » développé par SCHNEIDER *& al.* [118, 120]. Cet outil permet de représenter un alignement sans trous d'une collection de séquences $\mathcal{S}$ par un histogramme à 2 dimensions, où l'axe des abscisses étiquette les différentes positions de la séquence consensuelle, et l'axe des ordonnées est gradué par le contenu d'information (mesurée en *bits*). Les barres de l'histogramme sont remplacées par les symboles de l'alphabet correspondants, et leurs couleurs permettent de coder un attribut. En considérant par exemple le cas des acides aminés, le code de couleur est représentatif de leurs propriétés physico-chimiques (*cf.* Annexe A.3.2). La hauteur des *pseudo*-barres est définie par $f_i(\alpha) \times R(i)$ (elles sont empilées par valeurs croissantes).

Exemple 2.2 (Illustration du programme « *Sequence Logo* »).



Applications basées sur la notion d'attribut :

Ces applications permettent de mesurer la similarité entre deux objets d'un même ensemble lorsqu'il est possible de les projeter dans un espace quasi-métrique, semi-métrique ou pré-métrique.

Définition 2.21 (Espaces « pseudo »-métriques)**.** Un espace quasi-métrique est un ensemble $\mathcal{E}$ non vide tel qu'il existe une application $f : \mathcal{E} \times \mathcal{E} \to \mathbb{R}^+$ vérifiant les axiomes de séparation et d'inégalité triangulaire ; un espace semi-métrique est un ensemble $\mathcal{E}$ non vide tel qu'il existe une application $f : \mathcal{E} \times \mathcal{E} \to \mathbb{R}^+$ vérifiant les axiomes de séparation et de symétrie ; et un espace pré-métrique est un ensemble $\mathcal{E}$ non vide tel qu'il existe une application $f : \mathcal{E} \times \mathcal{E} \to \mathbb{R}^+$ vérifiant l'axiome de séparation. Les applications sont alors respectivement appelées une quasi-distance, une semi-distance et une pré-distance.

Dans ce contexte, la mesure de la similarité entre deux objets est alors basée sur la quantification des caractéristiques communes aux deux objets, par rapport à celles spécifiques à chacun ; d'où leur appartenance à la catégorie des applications basées sur la notion d'attribut. La formule d'une telle application peut être, par exemple,

$$S_A : x, y \mapsto \lambda\, f(Attr(x) \cap Attr(y)) - \alpha\, f(Attr(x) \setminus Attr(y)) - \beta\, f(Attr(y) \setminus Attr(x)),$$

ou encore :

$$S_B : x, y \mapsto \frac{\lambda\, f(Attr(x) \cap Attr(y))}{\alpha\, f(Attr(x) \setminus Attr(y)) + \beta\, f(Attr(y) \setminus Attr(x))},$$

où $Attr(x)$ (respectivement $Attr(y)$) représente l'ensemble des caractéristiques de x (respectivement y), où f est une application (d'un ensemble de caractéristiques) à valeurs dans $\mathbb{R}$ et où λ, α et β sont des réels.

Cette catégorie d'applications comporte notamment les indices de simple concordance, de JACCARD ou encore de SØRENSEN, décrits ci-après.

Définition 2.22. Une application $f : \mathcal{E} \times \mathcal{E} \to \mathbb{R}$ est un indice de similarité si pour tout $x, y \in \mathcal{E}$, $f(x, y) \leq f(x, x) = f(y, y)$.

Parmi les indices de similarité binaires, citons l'indice (symétrique) de **simple concordance** :

$$S_1 : x, y \mapsto \frac{v_{00} + v_{11}}{v_{00} + v_{10} + v_{01} + v_{11}}, \tag{2.11}$$

où v_{00}, v_{01}, v_{10} et v_{11} représentent respectivement le nombre d'attributs absents pour x et de y, absents pour x et présents pour y, présent pour x et absents pour y et enfin présents pour x et pour y.

Citons également l'indice (asymétrique) de communauté, également appelé indice de **JACCARD** :

$$S_7 : x, y \mapsto \frac{v_{11}}{v_{11} + v_{10} + v_{01}}. \tag{2.12}$$

Cet indice est utile lorsque le traitement de la double absence (v_{00}) pose problème. Citons également l'indice (asymétrique) de **SØRENSEN** :

$$S_8 : x, y \mapsto \frac{2\, v_{11}}{2\, v_{11} + v_{10} + v_{01}}, \tag{2.13}$$

qui accentue le poids de la double présence.

Équivalence entre applications :

Il est souvent possible de transformer une application non géométrique en une application géométrique [31] (c'est le cas par exemple de la distance de HAMMING). De plus, plusieurs applications différentes peuvent fournir une information très similaire. En effet, la mesure de la similarité permet de quantifier le degré de similarité entre deux ou plusieurs éléments, mais elle permet également d'établir une relation d'ordre entre les ensembles d'éléments comparés. Ainsi, il est possible de discriminer les ensembles d'objets les plus similaires. C'est pourquoi il est intéressant de pouvoir classer les applications en fonction des relations d'ordre qu'elles établissent. BATAGELJ et BREN ont établi, dans le cadre de la psychologie cognitive, un schéma de classification des applications géométriques [12], ce qui leur a permis de proposer une relation d'ordre partiel entre les différentes classes d'applications. Selon cette classification, deux applications d'une même classe sont censées avoir un comportement voisin. Une étude empirique en traitement du langage naturel [83] a permis de mesurer et de classifier (l'étude a été menée indépendamment des travaux précédents) la « performance » de plusieurs fonctions de similarité. Les résultats obtenus vont dans le même sens que ceux obtenus par BATAGELJ et BREN. En outre, bien que spécifiques à un domaine différent de la bioinformatique, ces résultats confirment l'importance que

revêt la connaissance – statistique ou empirique – du comportement d'une application. Ainsi, comme le préconisent JONASSEN *& al.* [72], les fonctions de scores doivent être pour le moins représentatives 1) de la similarité des motifs, mais aussi 2) de la diversité des séquences. Si cette seconde condition n'est pas vérifiée, il devient difficile d'accréditer (*i.e.,*, d'interpréter correctement) les résultats fournis par un quelconque algorithme.

2.2.2 Matrices de Distances/Similarités

Une vision plus générale de la notion de la distance de HAMMING, est d'associer un coût à chaque possibilité de substitution. Il est possible de représenter ces coûts par une matrice carrée de dimension égale à la taille de l'alphabet (non dégénéré) utilisé. Le terme de « matrice de distances » est également utilisé même lorsque la matrice n'est pas symétrique. À l'instar de la transformation des similarités en distances, il est également possible de transformer une distance en similarité, et par conséquent une matrice de distances en matrice de similarités.

Ces matrices peuvent être définies algébriquement. Par exemple, il est possible d'utiliser la matrice de l'exemple ci-dessous pour calculer la distance de HAMMING.

Exemple 2.3 (Matrice de distance de « HAMMING » pour un alphabet Σ).
La distance de HAMMING *entre deux mots w_1, w_2 de même longueur construits sur un alphabet $\Sigma = \{\alpha_1, \ldots, \alpha_{|\Sigma|}\}$ correspond (cf. Définition 2.17 – page 27) au nombre minimum de substitutions qu'il est nécessaire d'opérer sur w_1 pour obtenir w_2. En affectant un coût unitaire à la substitution d'un symbole α_i par un symbole α_j lorsque $i \neq j$ et un coût nul lorsque $i = j$, la distance de* HAMMING *s'obtient alors en sommant les coûts des substitutions des symboles $w_1[k]$ ($1 \leq k \leq |w_1|$) par les symboles $w_2[k]$. Il est ainsi possible d'utiliser une matrice afin de déterminer les coûts de toutes les substitutions possibles, comme illustré ci-après.*

	α_1		$\cdots$		$\alpha_{\lvert\Sigma\rvert}$
α_1	0	1	$\cdots$		1
	1	0			
$\vdots$			$\ddots$		$\vdots$
	$\vdots$		$\ddots$	0	1
$\alpha_{\lvert\Sigma\rvert}$	1		$\cdots$	1	0

Il est également possible de construire des matrices à partir d'observations biologiques. Notons qu'il n'est pas obligatoire que les matrices soient symétriques. En effet, certaines mutations surgissant au niveau de la phase de transcription de l'ADN engendrent des substitutions viables au niveau de la séquence d'acides aminés, qui sont probablement à l'origine des phénomènes d'asymétrie observés. Ces mécanismes sont complexes et ne sont pas encore totalement élucidés (plusieurs explications et hypothèses sont détaillées dans [115, chapitre 5]). La principale difficulté relative à l'usage des matrices est l'interprétation qui sera donnée du résultat. C'est la raison pour laquelle il est nécessaire de connaître et de comprendre les mécanismes ayant abouti à l'élaboration d'une matrice avant de l'utiliser.

Les observations peuvent être un simple comptage des fréquences des mutations pour une même famille de séquences biologiques. Il peut également s'agir d'un processus de reconstruction phylogénétique, visant à estimer sur des courts intervalles de temps – quelques milliers d'années tout de même – l'évolution supposée d'une famille de séquences. Ainsi, la similarité entre deux séquences calculée avec ce type de matrice est assimilable au degré de parenté existant entre ces deux séquences. Chaque

matrice représente alors un fragment de théorie de l'évolution. Une autre façon de construire les matrices consiste à tenir compte des propriétés physico-chimiques des molécules.

L'emploi de matrices prend tout son essor lors de l'étude du protéome, en partie du fait du nombre d'acides aminés. Parmi les matrices existantes, focalisons l'analyse sur les trois familles de matrices les plus connues et les plus utilisées par les biologistes.

Les matrices de type PAM (*cf.* Annexe C.2.1) [35] ont été construites sur la base de 71 familles de protéines relativement similaires (environ 1 300 séquences). L'alignement produit par ces séquences a permis de calculer toutes les probabilités de substitution des acides aminés entre eux. Cette matrice de probabilité de substitution est également appelée 1PAM ; en effet, elle traduit l'existence d'une substitution (qui n'altère pas la fonction biologique de la protéine) pour 100 sites dans un temps d'évolution t. Elle tire son nom de l'anglais *1 Point Accepted Mutations*. Si cette matrice est élevée à la puissance x, on en déduit une matrice xPAM, représentant les probabilités de substitutions pour un temps d'évolution $x\,t$. En considérant les logarithmes des probabilités, ces matrices deviennent alors des matrices de DAYHOFF [35], dénotées PAMx. Ces matrices ont été réactualisées en 1992 [73] sur la base de 2 621 familles de protéines (environ 16 000 séquences) issues de la base de données SWISSPROT.

Les matrices de type BLOSUM (*cf.* Annexe C.2.2) [64] sont, quant à elles, obtenues à partir des fréquences des substitutions observées sur des alignements de sous-séquences (ou blocs) de séquences protéiques variées. Sont comparés entre eux, les blocs ayant un pourcentage d'identité x donné. La matrice logarithmique obtenue sur la base de ces fréquences est alors appelée BLOSUMx (de l'anglais BLO*cks* SU*bstitution* M*atrix*).

Les matrices de type GONNET (*cf.* Annexe C.2.3) [53] sont construites à partir de l'intégralité des données contenues dans un ensemble de bases de séquences protéiques. Chaque séquence est comparée à toutes les autres, et tous les alignements significatifs (dont le score obtenu en utilisant initialement une matrice PAM250 a été supérieur à un seuil fixé, *cf.* Table C.2 – page 199) sont considérés. Une matrice de distance est alors calculée à partir de ces alignements, selon le même principe que les matrices PAM250. Celle-ci est alors utilisée afin de raffiner les alignements. Le processus est alors réitéré jusqu'à stabilisation de la matrice.

Le choix des matrices dépend du type de problème traité. Ainsi, les matrices de type BLOSUM sont les plus adaptées à l'extraction de motifs, tandis que les matrices de type PAM ou GONNET sont plus adaptées à l'alignement de séquences (les matrices de type GONNET (*cf.* Annexe C.2.3) donnant en général de meilleurs résultats). Néanmoins, il est possible d'établir des équivalences. Ainsi, l'usage d'une matrice PAM100 donnera des résultats similaires à ceux obtenus avec une matrice BLOSUM90. De même en ce qui concerne les résultats obtenus en utilisant une matrice PAM120 ou BLOSUM80, une matrice PAM160 ou BLOSUM60, une PAM200 ou BLOSUM52, ou encore une PAM250 (*cf.* Annexe C.2.1) ou BLOSUM45 (*cf.* Annexe C.2.2).

2.3 Notions de similarité et fonctions de score entre plusieurs mots

Définir une mesure de similarité entre plusieurs mots est une tâche plus ardue que de mesurer une similarité entre seulement deux mots. En effet, il est nécessaire de proposer préalablement une représentation d'un ensemble de mots. Cette représentation doit permettre l'interprétation de la mesure de

la similarité. Dans le contexte de l'analyse de séquences biologiques, une représentation fréquemment rencontrée consiste à aligner les mots sur une grille à deux dimensions, ou chaque ligne correspond à un mot et chaque colonne à une position d'au moins un mot. Les mots sont alors dits alignés. Toutefois, dans le contexte plus spécifique de l'analyse de séquences primaire d'ADN, il est possible de contraindre la représentation, en fixant que chaque colonne correspond à une position dans tous les mots. Cette condition, bien qu'assez restrictive, n'est pas déraisonnable. En effet, du point de vue biologique, les substitutions ont une fréquence d'apparition très supérieure à celles cumulées des insertions et des suppressions (*cf.* Section 1.2.1 – page 16). Les mots sont alors dits alignés sans trous (*cf.* Figure 2.1). Cela présuppose également que tous les mots soient de la même longueur.

a	c	g	a	t	c	g	a	c	g	a	c
g	a	t	a	g	c	t	a	c	g	a	g
g	g	a	g	c	a	c	g	c	t	a	g
g	a	t	c	g	a	c	t	a	g	c	t
a	g	c	a	g	c	a	c	t	a	g	c
a	g	c	t	a	g	c	a	t	c	g	a

Figure 2.1 – Alignement sans trou d'un ensemble de mots.

2.3.1 Quelques fonctions basées sur la mesure de l'entropie

La mesure de l'entropie a été initialement élaborée par SHANNON en 1948 [121] dans le cadre du traitement du signal (*cf.* Section 2.2.1 – page 25). Cette mesure correspond à la différence entre la quantité d'information nécessaire *a priori* (*cf.* Équation 2.4 – page 28) pour coder chaque symbole d'une séquence à une position donnée et la quantité d'information nécessaire à ce même codage *a posteriori* pour cette même position (*cf.* Équation 2.7 – page 28). Ainsi, cette mesure est calculée sur la base des probabilités d'obtenir chaque symbole de Σ à la position i ainsi que sur leurs fréquences à cette même position *a posteriori* dans les séquences d'un ensemble $\mathcal{S}$ (resp. $p^i_\alpha, f_i(\alpha), \alpha \in \Sigma$). La mesure de l'entropie à la position i est définie (*cf.* Équation 2.8 – page 28) par

$$R(i) = \sum_{\alpha \in \Sigma} f_i(\alpha)\ \log_2 p^i_\alpha - \frac{\sum_{\alpha \in \Sigma} \log_2 p^i_\alpha}{|\Sigma|}.$$

Il est possible de donner une approximation de cette formule (*cf.* Équation 2.9 – page 29).

Cette mesure peut être utilisée afin d'associer un score à un alignement local sans trous, par exemple en sommant l'entropie engendrée à chaque position i de cet alignement (*i.e.*, $\sum_i R(i)$). Cette mesure a été utilisée dans PRATT [18, 71] (*cf.* Section 2.5.2 – page 38). Il été montré que ce score suivait une loi Gamma $G(n, \lambda)$, où les paramètres n et λ dépendent essentiellement des probabilités d'apparition des symboles de l'alphabet (quelques exemples de la distribution en fonction des paramètre n et λ sont donnés à l'annexe E.2).

Plusieurs variantes peuvent être développées, par exemple en considérant la moyenne (arithmétique, géométrique, harmonique, quadratiques, ...) de l'entropie sur l'ensemble des positions. Dans le contexte de la comparaison d'alignements sans trous de longueur variables – ce qui est le cas dans le contexte de cette étude – ces variantes ont pour effet d'atténuer le biais induit par la longueur des alignements, ce qui n'est pas nécessairement opportun. En effet, si deux alignements de longueurs différentes sont

considérés égaux par l'utilisation d'une moyenne, il peut s'avérer préférable de considérer de prime abord l'alignement le plus grand.

L'utilisation de la mesure de l'entropie pour chaque position dans un alignement de plusieurs mots donné permet de surcroît d'induire une relation d'ordre entre les symboles de l'alphabet (pour chacune des positions). Cette mesure permet alors de déterminer un motif consensuel (qui se veut le plus représentatif, eu égard à la fonction de score utilisée) de l'alignement, voire même d'utiliser le contenu d'information en vue de l'élaboration d'un « *Sequence Logo* » (*cf.* Équation – page 29). Nonobstant l'usage d'autres fonctions de score, l'usage de « *Sequence Logo* » demeure pleinement justifié pour représenter de tels consensus.

2.3.2 Utilisation de matrices de similarité

Permettre l'usage des matrices de similarité est important (*cf.* Section 2.2.2 – page 31). En effet, elles sont le reflet d'une réalité biologique. Les possibilités de scores offertes par ces matrices sont nombreuses. De la même façon que pour la mesure de l'entropie, il est possible de sommer ou de déterminer un coût moyen par position dans un alignement, et de calculer, à partir des valeurs obtenues pour chaque position, une somme, un produit, une moyenne, ...

2.3.3 Mesures composites

Une des manières de calculer un score global à l'ensemble des mots de l'alignement est d'opérer un calcul (moyenne, somme, produit, minimum, ...) sur les scores des mots pris deux à deux. Si la collection de mots est alignée sur un modèle consensuel, alors il est probablement plus judicieux de considérer les scores de chaque mot par rapport au modèle (cette méthode peut également être utilisée pour construire un modèle consensuel, permettant de maximiser/minimiser le score global). La figure 2.2 illustre la différence entre ces deux approches. La partie gauche de la figure (en étoile) représente le calcul d'un score par rapport à un modèle consensuel (en gris clair), tandis que la partie droite correspond au calcul d'un score entre les motifs pris deux à deux. Les boîtes grises correspondent aux motifs et un trait entre deux boîtes signifie que les deux motifs représentés sont similaires. Dans le cas de l'approche en étoile, le motif central (en gris clair) n'appartient pas nécessairement à la collection de motifs.

De même que dans le cadre de la mesure de la similarité de deux mots, il est possible d'utiliser un score seuil afin de déterminer si les mots d'un ensemble sont (globalement) similaires. Il n'y a pas nécessairement d'implication forte entre la similarité globale d'un ensemble de mots et la similarité des mots pris deux à deux (similarité locale). Dans le cas de l'approche en étoile par exemple, plusieurs mots considérés comme globalement similaires peuvent tout à fait n'être pas similaires deux à deux.

2.4 La recherche de motifs

Les mesures de similarité permettent d'évaluer la proximité (ou la distance) de deux ou plusieurs mots pris dans un texte ; qu'il s'agisse d'un texte biologique ou non. Le problème de la recherche de motifs dans un texte est souvent traité en utilisant une mesure de la similarité afin de définir une notion de similarité. Dans sa forme la plus simple (*i.e.*, la notion de similarité utilisée est l'égalité lexicale), ce problème peut-être formulé comme suit :

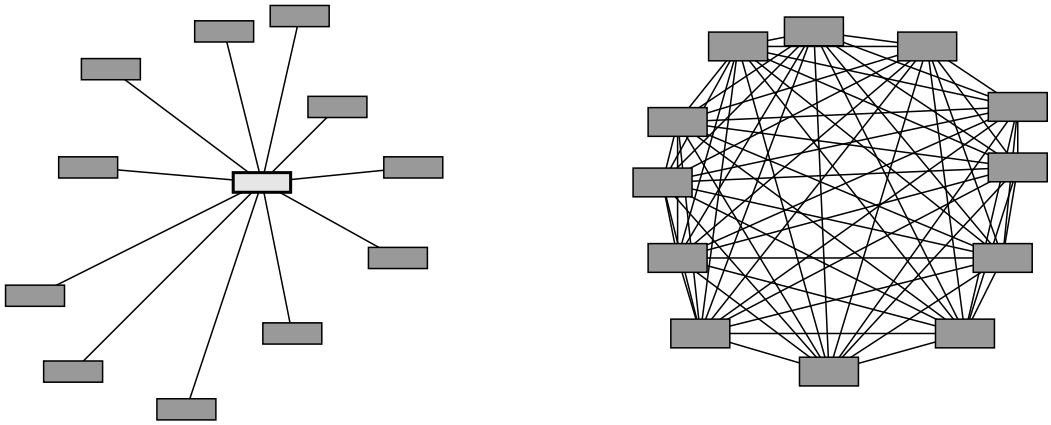


Figure 2.2 – Similarité par rapport à un modèle externe *vs.* similarité entre motifs.

Recherche exacte d'un motif dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), ainsi qu'un motif $m \in \Sigma^*$.

Problème: trouver toutes les occurrences du motif m dans les séquences $s_1, \ldots, s_t$.

Ce problème a été très largement étudié [25], et de nombreux algorithmes permettent d'y répondre. La meilleure complexité en nombre d'accès aux séquences dans le meilleur des cas est $\Theta(n/p)$ (où n est la somme des longueurs des séquences et p la taille du motif). À titre d'exemple, cette borne est atteinte par l'algorithme de BOYER-MOORE [16]. Dans le pire des cas, la meilleure complexité est en $\Theta(n)$, et est atteinte notamment par l'algorithme *Forward* DAWG *Matching* [33]. Enfin, dans le cas moyen, elle est en $\Theta(n \log_{|\Sigma|}(p)/p)$. Cette complexité est atteinte dans le cas moyen entre autres algorithmes par *Reverse Factor* (ou *Backward* DAWG *Matching*) [82].

Une variante de ce problème a également été étudiée [61].

Recherche exacte de plusieurs motifs dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), ainsi qu'un ensemble de motifs $\mathcal{M} = \{m_1, \ldots, m_q\}$, ($m_i \in \Sigma^*, 1 \leq i \leq q$).

Problème: trouver toutes les occurrences des motifs $m_1, \ldots, m_q$ dans les séquences $s_1, \ldots, s_t$.

Là aussi, de nombreux algorithmes ont été élaborés afin de répondre à ce problème [102], et les complexités théoriques optimales sont connues [103]. La meilleure complexité dans le meilleur des cas est atteinte par une variante de BOYER-MOORE. Sa complexité est alors en $O(q\,n/p)$, où n désigne toujours la somme des longueurs des séquences, et p désigne cette fois la taille du plus petit motif. L'algorithme de AHO-CORASICK [1] est optimal dans le pire des cas, avec une complexité en $\Omega(n)$.

Enfin, le meilleur algorithme dans le cas moyen est en $\Theta(n \log_{|\Sigma|}(p\,q)/p)$; il s'agit de *Multi-Backward* DAWG *Matching* [109].

Néanmoins, ces deux premiers problèmes ne sont que peu exploitables dans le cadre de l'exploration de séquences biologiques. En effet, les motifs biologiques sont extrêmement rarement conservés intégralement. Du fait des mutations, il est utile d'y intégrer une notion de similarité.

Recherche approchée d'un motif dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), un motif $m \in \Sigma^*$, ainsi qu'une notion de similarité.

Problème: trouver tous les motifs similaires à m dans les séquences $s_1, \ldots, s_t$.

La complexité des algorithmes répondant à ce problème est dépendante de la notion de similarité utilisée. Les complexités dans le meilleur des cas et dans le cas moyen ont été étudiées pour plusieurs notions de similarité. Les résultats présentés ici correspondent au cas où deux motifs sont considérés comme similaires s'ils sont à distance de HAMMING, INDEL ou de LEVENSHTEIN, au plus k, avec $k \in \mathbb{N}$. Dans le meilleur des cas, le meilleur algorithme est en $\Theta(n/p)$; il s'agit toujours de l'algorithme BOYER-MOORE. La complexité optimale dans le pire des cas demeure inconnue. Elle est bornée par $O(k\,n)$. Dans le cas moyen, la complexité optimale est en $\Omega(n\,(k + \log_{|\Sigma|} p)/p)$ [23]. Pour $k \leq p/2$, la complexité optimale est alors en $\Theta(n\,(k + \log_{|\Sigma|} p)/p)$.

Comme pour le problème de la recherche exacte d'un motif, il est possible de définir une variante du problème énoncé ci-dessus.

Recherche approchée d'un ensemble de motifs dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), un ensemble de motifs $m_1, \ldots, m_q \in \Sigma^*$, ainsi qu'une notion de similarité.

Problème: trouver tous les motifs similaires à l'un des motifs $m_1, \ldots, m_q$ dans les séquences $s_1, \ldots, s_t$.

La complexité optimale en réponse à ce problème pour les distances de HAMMING, INDEL et de LEVENSHTEIN dans le meilleur des cas est la même que dans sa version exacte, à savoir $O(q\,n/p)$. Elle est encore une fois atteinte par une variante de l'algorithme de BOYER-MOORE. Le pire des cas pour ces distances est bien évidemment inconnue[2], et la complexité optimale demeure bornée par $O(k\,q\,n)$. Enfin, le cas moyen a été également étudié [103], et la meilleure complexité est en $\Omega(n(k + \log_{|\Sigma|}(p\,q)/p))$. Pour $k \leq p/2$, elle est en $\Theta(n(k + \log_{|\Sigma|}(p\,q)/p))$.

2.5 Le problème de l'extraction de motifs

Si les problèmes de la recherche de motifs sont polynomiaux, ceux d'extraction sont très souvent NP-durs. Les premières approches de ce problème concernent la découverte de motifs exactement répétés sur une séquence [76] ou deux séquences [123]. La première formalisation du problème de la découverte

[2] La complexité optimale dans le pire des cas du problème plus simple de la recherche approchée d'un motif dans un ensemble de séquences étant lui même inconnu.

de motifs dans un ensemble de séquences introduisant une notion de similarité est issue des recherches en fouilles de données [39]. La problématique est introduite en bioinformatique d'abord de manière très restreinte et informelle en 1987 [55], puis de manière générique et formelle en 1992 [117]. Un historique des techniques d'extraction de motifs en bioinformatique a été établi par RIGOUTSOS *& al.* [113] en 2000.

Avant de donner une formalisation du problème, revenons sur les motivations biologiques. L'extraction de motifs similaires présents dans plusieurs séquences biologiques (ADN, ARN et protéines) permet avant tout de mieux comprendre les mécanismes de la vie cellulaire (*cf.* Section 1.3.2 – page 18). La mise en évidence de certaines régions dans les séquences permettent parfois d'isoler les causes de certaines maladies (cancers, contaminations virales, ...) mais également les mécanismes d'immunologie à ces maladies [19]. De plus, la découverte de ces régions peuvent également conduire à l'élaboration de tests et de médicaments [56]. Une autre application liée à l'extraction de motifs est la classification des séquences en familles ; les motifs sont alors utilisés comme des signatures. Dans certains cas, ces classifications peuvent mener à l'élaboration de modèles d'évolution (*cf.* Section 1.2 – page 16). Enfin, les similarités locales entre les séquences sont souvent la conséquence d'un rôle structurel ou biologique commun à ces séquences, tels que la présence de sites de liaisons (*cf.* Section 1.1.3 – page 15), de régions régulatrices, ...

Jusqu'en 1987, les similarités locales entre les séquences biologiques étaient essentiellement détectées par des méthodes d'alignement multiple (*i.e.*, plusieurs séquences). Ces méthodes sont efficaces lorsque les séquences sont globalement assez proches. En revanche, lorsque les séquences sont assez éloignées les unes des autres, ces méthodes ne permettent plus d'isoler les similarités locales. C'est précisément en essayant d'extraire des motifs similaires entre des séquences très éloignées que GRIBSKOV *& al.* [55] ont introduit le problème de l'extraction de motifs pour la première fois.

2.5.1 Formalisation du problème

Dans sa forme la plus simple, le problème de l'extraction de motifs, noté EM, se définit ainsi :

Extraction de motifs communs à un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), ainsi qu'une notion de similarité globale.

Problème: trouver toutes les collections de motifs $(m_1 \in s_1, \ldots, m_t \in s_t)$ telles que $m_1, \ldots, m_t$ soient similaires.

La notion de similarité globale (*i.e.*, entre plusieurs motifs) peut être définie de plusieurs manières. Il peut s'agir d'une notion basée sur un score d'alignement desdits motifs, d'une notion définie en comparant les motifs deux à deux, d'une notion basée sur la similarité entre les motifs et un motif externe ou consensuel (*cf.* Figure 2.2), ...

Il est possible de définir une variante de ce problème, notée EM_q, en intégrant une notion de quorum.

Extraction sous contrainte de quorum de motifs communs dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), un quorum q ($0 \leq q \leq 100\%$), ainsi qu'une notion de similarité globale.

Problème: trouver toutes les collections de $R \geq \lceil q\,t \rceil$ motifs $(m_1 \in s'_1, \ldots, m_R \in s'_R)$ telles que $\{s'_1, \ldots, s'_R\} \subseteq \mathcal{S}$ ($s'_i \neq s'_j$) et que $m_1, \ldots, m_R$ soient similaires.

Remarque 2.23. Le problème EM correspond au cas particulier du problème EM_q lorsque $q = 100\%$, soit EM_{100}.

2.5.2 Solutions existantes

Depuis 1987, de nombreux algorithmes ont vu le jour pour tenter de répondre à ces problèmes. Plusieurs formulations en ont été données, et plusieurs notions de similarité ont été étudiées. BRAZMA *& al.* ont proposé [18] une classification des algorithmes en fonction de la structure des motifs qu'ils sont capables d'extraire (*cf.* Annexe B.3). Il est également possible de proposer une classification basée sur les principes de fonctionnement des algorithmes.

Classification

Les algorithmes peuvent être classé en quatre catégories, selon qu'ils sont « déterministes » ou « probabilistes », et selon qu'ils sont basés sur une structure d'index ou sur une fouille des données. Avant de classer les principaux algorithmes d'extraction de motifs, il est nécessaire d'expliciter chaque catégorie.

Algorithme déterministe vs. Algorithme probabiliste :

Un algorithme est déterministe si étant données ses entrées, il renvoie toujours les mêmes résultats. Il est probabiliste dans le cas contraire. Un algorithme probabiliste emploie généralement un générateur de nombres [pseudo-]aléatoires, en vue de sélectionner un chemin vers un résultat. Toutefois, ceci n'est pas une condition suffisante pour que l'algorithme soit probabiliste. Par exemple, l'algorithme de tri rapide peut-être réalisé en choisissant le pivot aléatoirement, et pourtant le résultat de l'algorithme est toujours le même. Il est donc déterministe.

Base d'index vs. Base de Fouille :

Une structure d'indexation permet de représenter les données (dans le cas présent les séquences) afin d'extraire les informations contenues (ici les facteurs, suffixes ou préfixes) rapidement. Les structures utilisées en algorithmique du texte sont principalement les *Trie* , les PATRICIA *Trie* , les Automates des Suffixes, les Arbres des Suffixes [33] ou encore les tables des suffixes [93]. Les algorithmes qui ne se basent pas sur de telles structures retrouvent donc les informations dont ils ont besoin en fouillant les données.

Les algorithmes d'extraction de motifs

Il est ambitieux de vouloir être exhaustif à propos des méthodes d'extraction existantes, néanmoins certains algorithmes sont devenus des références dans ce domaine, tant du point de vue de leur efficacité que des modélisations qu'ils ont donné de ce problème.

Chaque algorithme présenté dans la table 2.1 est une réponse au problème EM ou EM_q. Chacun d'entre eux est ici présenté succinctement, en précisant le type de motifs qu'ils permettent de découvrir, et en insistant sur leurs avantages et leurs inconvénients.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Algorithmes</td></tr>
<tr><td>Déterministes</td><td>Probabilistes</td></tr>
<tr><td rowspan="2">À base</td><td>d'index</td><td>MOTIF [122]
PRATT [71]
SMILE [99]</td><td>DISCOVER [129]</td></tr>
<tr><td>de fouille</td><td>MEME [9]
SPLASH [22]
TEIRESIAS [112]
WINNOWER [108]
CONSENSUS [66]</td><td>Meta-MEME [57]
GIBBS [81]
PROJECTION [21]</td></tr>
</table>

Table 2.1 – Algorithmes d'extraction de motifs.

BRAZMA *& al.* [18] ont élaboré une classification des algorithmes selon le type de motifs qu'ils permettent d'extraire (*cf.* Annexe B.3). Cette classification est élaborée en suivant la notation PROSITE (*cf.* Annexe B.2). Afin de faciliter la lecture, les différentes classes de motifs sont également donnés dans la table 2.2.

classe	motifs	trous	illustration
A	simples	non	T-C-T-T-G-A
B	simples	longueur fixe	D-R-C-C-x(2)-H-D-x-C
C	dégénérés	non	G-G-G-T-F-[ILV]-[ST]-[ILV]
D	dégénérés	longueur fixe	V-x-P-x(2)-[RQ]-x(4)-G-x(2)-L-[LM]
E	simples	longueur bornée	G-C-x(1,3)-C-P-x(8,10)-C-C
F	dégénérés	longueur bornée	C-x(2,4)-C-x(3)-[ILVFYC]-x(8)-H-x(3,5)-H
G	simples	longueur non bornée	D-T-A-G-Q-E-*-L-V-G-N-K
H	dégénérés	longueur non bornée	D-T-A-G-[NQ]-*-L-V-G-N-[KEH]
I	dégénérés	longueur bornée ou non	D-T-A-x(2,5)-G-[NQ]-*-L-V-G-N-[KEH]

Table 2.2 – Classification des algorithmes d'extraction de motifs.

Les algorithmes déterministes à base d'index :

MOTIF [122] est exclusivement dédié à l'extraction de motifs dans les séquences protéiques (problème EM_q). L'algorithme utilise une matrice afin de représenter l'ensemble des motifs contenant exactement trois acides aminés, séparés par un nombre fixe de jokers (le nombre maximum de joker est fixé à l'avance par l'utilisateur). Cette description le classe dans la catégorie *B*. Chaque motif est quantifié par un score obtenu à partir d'une matrice PAM250. Les motifs contigus (sur les séquences) sont ensuite concaténés, et leur score mis à jour. Plusieurs paramètres doivent être fixés par l'utilisateur, afin de restreindre l'espace de recherche, tels que le nombre d'occurrences minimum d'un motif sur l'ensemble des séquences, ou encore le nombre de motifs à renvoyer.

PRATT [71] apporte une réponse au problème EM_q. Cet algorithme utilise un index afin de représenter les motifs présents dans tout ou partie des séquences. À chaque motif de cet index (représenté par un nœud) sont également associées plusieurs informations, dont le nombre de séquences dans lesquelles il est présent, ainsi que son score. Le type de motifs renvoyés par cet algorithme le classent dans la catégorie *F*. Par défaut, PRATT utilise la mesure de l'entropie (*cf.* Section 2.2.1 – page 25) afin d'assigner un score aux motifs, il est toutefois possible de choisir une autre fonction de score parmi les quatre autres fonctions proposées. De plus, cet algorithme intègre la notion de *quorum*. Les paramètres en entrée de l'algorithme sont peu nombreux, et relativement explicites. Ils requièrent assez peu de connaissances *a priori* sur le(s) motif(s) à extraire. Toutefois, il est nécessaire (en raison de la structure d'index utilisée) de fixer une borne supérieure de la longueur des motifs à extraire. Dans sa version courte, l'algorithme met en évidence des petits motifs très conservés. Ils sont en général trop petits pour être significatifs. L'algorithme propose une phase de « raffinement » afin d'étendre les motifs, mais les motifs obtenus alors sont souvent trop dégénérés dans le cas de séquences nucléiques pour être exploitables. Cet outil est donc réellement efficace lorsque les motifs à extraire sont petits et très bien conservés, ou alors dans le cas de séquences protéiques.

SMILE [99] est basé sur un arbre des Suffixes [127] (*cf.* Section 5.1.2 – page 142). Comme PRATT, cette méthode intègre la notion de *quorum* (problème EM_q), et appartient à la classe *F*. La fonction de score utilisée est ici la distance de HAMMING dans un premier temps, puis l'estimation d'un Z-score (score centré réduit – la notion de Z-score est détaillée à la section 4.3.1 – page 110). Le paramétrage est, comme précédemment, assez réduit, toutefois il réside une ambiguïté, puisqu'il faut fixer à la fois le nombre de substitutions maximum autorisées et le nombre maximum de jokers. Ainsi, certaines substitutions sont considérées comme des jokers et d'autres non. De plus, il est nécessaire de fixer les bornes inférieure et supérieure de la longueur des motifs à extraire. Enfin, l'estimation du Z-score est réalisée de manière très empirique, et n'apporte pas nécessairement autant de pertinence et de crédit qu'il n'y paraît de prime abord (*cf.* [38] pour une discussion à ce propos).

Un algorithme Probabiliste à base d'index :

DISCOVER [129] apporte une réponse au problème EM. L'algorithme construit un arbre des suffixes à partir d'un sous-ensemble des séquences sélectionné au hasard. De cet arbre, sont extraits les motifs les plus similaires (le score utilisé alors est la distance de LEVENSHTEIN). Ces motifs sont ensuite recherchés sur l'ensemble des séquences. Les résultats obtenus subissent ensuite un *post*-traitement permettant de fournir des motifs appartenant à la classe *G*. Plus la taille de l'échantillon de séquences choisi initialement est importante, plus les résultats sont fiables. En contrepartie, la première phase de l'algorithme est la plus gourmande en temps de calcul. Ainsi, la grande difficulté avec cette méthode est de trouver le juste équilibre entre le temps de calcul et la pertinence des résultats.

Les algorithmes Déterministes à base de fouille :

MEME [9] est basé sur la maximisation d'un critère statistique appelé la vraisemblance. Il s'agit en fait de la probabilité qu'un événement (l'existence d'un motif par exemple) se produise en fonction des données qui ont déjà été traitées et des données restant à traiter. L'algorithme fabrique un modèle de longueur fixée *a priori* qui possède la propriété de maximiser la vraisemblance (ou son logarithme, ce qui revient au même). Le modèle est affiné par itérations successives ; jusqu'à stabilisation du maximum de vraisemblance (à un ε près), ou jusqu'à ce qu'un nombre maximum d'itérations aient été effectuées. La valeur de ε et du nombre maximal d'itérations peut être fixé par l'utilisateur. Si les résultats obtenus

par cette méthode sont dans l'ensemble fiables (les motifs trouvés sont valides), cet algorithme présente l'inconvénient d'en manquer un certain nombre. Cette méthode appartient à la catégorie *C* et répond au problème EM_q.

SPLASH [22] et TEIRESIAS [112] renvoient tous deux des motifs de la classe *B*, et répondent au problème EM_q. En plus d'un paramétrage compliqué, le nombre trop important de motifs fournis en résultats rendent ces méthodes peu exploitables dans le cadre de la découverte de sites biologiques particuliers. Toutefois, TEIRESIAS a été utilisé par IBM dans un tout autre contexte, celui de la lutte contre les SPAM.

WINNOWER [108] utilise la distance de HAMMING afin d'extraire les motifs similaires deux à deux, et répond au problème EM. L'algorithme nécessite que la longueur des motifs soit connue *a priori*. Outre cet inconvénient, la seule garantie apportée par cette méthode est que s'il existe une collection de motifs de longueur fixée et deux à deux à distance de HAMMING bornée (classe *A/C*), alors elle figurera dans l'ensemble des collections de motifs renvoyés.

Les algorithmes Probabilistes à base de fouille :

GIBBS [81] utilise un principe d'échantillonnage afin d'évaluer la pertinence et de raffiner des motifs sélectionnés au hasard. Cette méthode traite le problème EM_q. Bien que la longueur des motifs doit être fixée *a priori*, sa bonne vitesse d'exécution permet d'exécuter l'algorithme plusieurs fois en faisant varier la longueur. Les résultats obtenus par cette méthode sont généralement très bons, surtout lorsque les séquences sont globalement éloignées. Toutefois si les séquences sont globalement toutes très proches les unes des autres, alors la qualité et la pertinence des résultats risquent fort d'en souffrir. Les motifs renvoyés classent cette méthode dans la catégorie *C*.

PROJECTION [21] a été créé en vue de résoudre à un problème (énoncé dans [108]) et de concurrencer WINNOWER. Ce problème consiste à extraire tous les motifs de même longueur fixée *a priori* et à distance de HAMMING également fixée *a priori* dans un ensemble de séquences (EM). L'algorithme se classe donc dans la catégorie *A/C*. Il s'appuie sur l'élaboration d'une fonction de hachage associée à chaque motif à extraire. La fonction de hachage est établie en choisissant aléatoirement un certain nombre (paramètre en entrée, dont les bornes conseillées sont fonction de la distance maximale, du nombre et de la taille des séquences) de positions dans les motifs présents dans les séquences. Lorsque pour une même clé de hachage, le nombre de motifs associé est suffisamment important, la collection de motifs est analysée en vue d'en extraire les résultats. Le processus est réitéré plusieurs fois (sur la base d'une estimation statistique de ce nombre). Cet algorithme offre des performances théoriques quasi optimales selon les auteurs. Pourtant, il semble qu'il soit inadapté à l'extraction de motifs dans les séquences biologiques (toujours selon les auteurs).

Meta-MEME [57] utilise les résultats de MEME pour construire un modèle de MARKOV caché (*cf.* [79, chapitre 10] pour un complément d'information au sujet de cette structure). Cet outil répond donc au problème EM_q. Il est ensuite utilisé pour chercher des motifs consensuels dans des banques de séquences. Les résultats obtenus par cette méthode sont fortement dépendants de la qualité des bases de données utilisées, et demeurent assez difficiles à interpréter.

Un algorithme à base de fouille :

CONSENSUS [66] propose en option une version déterministe ou bien probabiliste pour répondre au problème EM. En effet, l'algorithme est basé sur le choix de motifs candidats dans les séquences. Ces motifs (ou graines) sont utilisés ensuite pour élaborer les motifs consensuels résultats de type *C*. Le choix

de ces graines est effectué soit linéairement sur la première séquence, ce qui confère à l'algorithme son caractère déterministe (auquel cas les résultats dépendent fortement de l'ordre des séquences) ; soit les graines sont choisies arbitrairement dans les séquences, rendant l'algorithme probabiliste. Le principal inconvénient de cette méthode réside dans le fait que la longueur du motif à extraire doit être connue à l'avance.

2.5.3 Analyse des besoins

Les méthodes précédemment décrites sont loin d'être universelles. En effet, elles sont généralement dédiées à l'extraction d'une catégorie de motifs : les promoteurs et autres sites de liaisons. Celles qui se veulent plus généralistes sont en général peu efficaces en pratiques. Pourtant, malgré cette spécialisation sur le type de motifs recherchés, les inconvénients de chaque méthode sont nombreux : motifs de longueur fixée à l'avance, schéma de score figé, classe de motif trop restrictive, réponses inexploitables, ..., quand il ne s'agit pas tout simplement du paramétrage qui par sa complexité rend la méthode inutilisable. La plupart des algorithmes d'extraction sont efficaces lorsque l'utilisateur est très bien renseigné sur ce qu'il faut découvrir. Dès que les algorithmes sont utilisés « en aveugles », ils deviennent purement et simplement inexploitables.

Au constat de ces lacunes, il apparaît évident qu'il est nécessaire d'apporter une solution au problème de l'extraction de motifs qui propose à la fois une paramétrisation simplifiée et une grande modularité dans le type de motifs à découvrir. Le chapitre suivant décrit un nouvel algorithme d'extraction de motifs (EM) : STAMS [is a **T**ool for **A**nalysis & **R**esearch in **S**equences]. Le cahier des charges de cet algorithme a été élaboré en vue d'offrir cette simplicité d'utilisation, ainsi que cette souplesse dans la description des motifs à extraire. Ceci constitue le principal axe de recherche de cette thèse. Il s'agit d'un algorithme probabiliste à base de fouille, renvoyant des motifs de la classe *D*. Dans le cadre de l'étude des propriétés de cette nouvelle méthode, un second algorithme est proposé au chapitre 4, répondant au problème EM_q : StatiSTAMS [uses **S**tatistical **T**echniques for **A**nalysis & **R**esearch in **S**equences]. Cette méthode est également classée dans la catégorie *D*. Afin d'améliorer le traitement des séquences en entrée, nous nous sommes intéressé à une structure d'indexation introduite par ALLAUZEN *& al.* en 1999. Cette structure (la plus économique à ce jour) a donné des résultats très prometteurs, mais d'obscurs détails demeurent (ensemble des mots reconnus par la structure, dénombrement de cet ensemble, ...). L'utilisation de cette structure pourrait être très intéressante, pour le stockage et la comparaison des séquences deux à deux dans STAMS et StatiSTAMS, sous réserve que les points obscurs ne le soient plus. Dans ce contexte, une étude approfondie de cette structure a été menée mettant en évidence l'ensemble de mots reconnus par cet index, ainsi que la pertinence statistique de ses résultats. Les conclusions de cette étude ne permettent pas de légitimer l'utilisation de cet index en l'état. Afin d'améliorer l'efficacité de cette structure, une légère modification est présentée à la fin du chapitre 5. Le nouvel index ainsi proposé demeure aussi séduisant par sa simplicité que la structure initiale, mais semble nettement plus fiable et efficace selon les premières constatations. Une étude plus théorique de la pertinence de cette nouvelle structure est en cours, mais nécessitera un temps d'étude probablement assez long. La suite de cette étude s'inscrit naturellement dans les recherches ultérieures aux travaux présentés dans ce manuscrit.

CHAPITRE 3

Extraction de motifs et STABS

Si quid sine probatione affirmatur, sine probatione negare potest
Ce qui est affirmé sans preuve, peut être nié sans preuve.

— Euclide de MÉGARE [450–380 av. JC].

L'objet de ce chapitre est la présentation d'un nouvel algorithme d'extraction de motifs communs à un ensemble de séquences. Le principe de cette méthode a été présenté dans [97] ; elle utilise plusieurs heuristiques afin de réduire de manière drastique l'espace des solutions et donner une réponse approchée au problème de l'extraction de motifs en temps polynomial.

Ainsi, sur la base de la distance de HAMMING, sont définies plusieurs notions de similarité, permettant ainsi de délimiter l'espace de recherche des solutions. Cet espace est ensuite restreint au moyen d'une première heuristique basée sur une mesure de la similarité. Plusieurs mesures usuelles et incontournables sont rappelées (matrices de similarité, mesure de l'entropie, ...), et replacées dans le contexte de cette étude. Après ce rappel, sont introduites trois nouvelles mesures de la similarité. Une fois donné l'ensemble des définitions, l'algorithme est détaillé pas à pas. Sa complexité spatiale et temporelle est calculée ensuite. Ce chapitre se termine par l'analyse de l'algorithme sur des jeux de séquences générées aléatoirement, ainsi que sur des jeux de séquences biologiques. Les tests sur les séquences aléatoires offrent une indication statistique de la qualité des résultats. Les tests sur les séquences biologiques permettent d'une part d'accréditer les résultats statistiques, et d'autre part de positionner cet algorithme par rapport à plusieurs méthodes faisant office de références (*cf.* Section 2.5.2 – page 38).

3.1 Analyse du problème

Le problème de l'extraction de motifs a été présenté dans la section 2.5 – page 36. Rappelons le succinctement :

Extraction de motifs communs à un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), ainsi qu'une notion de similarité globale.

Problème: trouver toutes les collections de motifs $(m_1 \in s_1, \ldots, m_t \in s_t)$ telles que $m_1, \ldots, m_t$ soient similaires.

Plusieurs approches de ce problème peuvent être distinguées. Parmi celles-ci, il est possible d'en dégager deux : l'une consiste à effectuer un parcours exhaustif de l'espace de recherche, afin de renvoyer toutes les solutions, tandis que l'autre cherche les solutions dans un sous-ensemble de l'espace des possibles. Dans le cas d'un parcours exhaustif de l'espace de recherche, pour une même notion de similarité, les résultats seront les mêmes quelle que soit la méthode d'investigation de cet espace ; l'évaluation de la qualité (du point de vue de l'informaticien) d'une méthode portera essentiellement sur la complexité de l'algorithme. L'évaluation de la qualité de la méthode (du point de vue du biologiste) sera au contraire intrinsèquement liée au choix de la notion de similarité. Dans le second cas, le choix des heuristiques permettant de restreindre l'espace des solutions est déterminante dans la fiabilité de la méthode. Ainsi, la qualité de l'algorithme (tant du point de vue du biologiste que de l'informaticien) sera appréciée principalement en fonction du rapport entre sa complexité[1] et sa fiabilité. Le choix entre ces deux approches dépend du type de réponse attendu. En effet, si la première approche permet de garantir que toutes les réponses et seulement celles-ci sont trouvées, cela se produit au détriment du temps de calcul (l'espace de recherche et parfois de solutions étant le plus souvent exponentiel en la taille des données). *A contrario*, une réponse basée sur des heuristiques n'offre généralement ni la certitude d'avoir toutes les réponses, ni la garantie que les réponses proposées soient toutes exactes. Le principal avantage des heuristiques

[1] Du point de vue du biologiste, il s'agit plutôt du temps d'exécution.

consiste à faire chuter la complexité de la méthode. Le choix qui est effectué dans la conception d'un algorithme sera donc basé sur un compromis « acceptable » entre la fiabilité des résultats et le temps de calcul nécessaire à leur obtention.

Dans tous les cas, la première démarche consiste à déterminer le type de motifs que l'on souhaite extraire d'un ensemble de séquences, ce qui revient à déterminer la (ou les) notion(s) de similarité(s) à considérer.

3.2 Modélisation du problème

Plusieurs mesures de similarité ont déjà été présentées dans le précédent chapitre (*cf.* Sections 2.2 à 2.3 – pages 25 à 32). La plupart sont des mesures de similarité permettant de comparer deux objets (*e.g.*, des motifs) d'un même ensemble. Il est alors question de similarité deux-à-deux. Dans certain cas, il s'agit d'une similarité pouvant être calculée sur plus de deux objets (*e.g.* mesure de l'entropie, *cf.* Section 2.3.1 – page 33). Cela permet de quantifier une mesure de similarité globale. Il est toutefois possible d'utiliser des mesures de similarités deux-à-deux pour calculer une similarité globale, par exemple en mesurant la similarité moyenne entre chaque paire d'objets ou bien de chaque objet par rapport à un modèle externe (*cf.* Figure 2.2). Les avantages et inconvénients des deux approches sont décrits à la section 3.3 – page 57.

3.2.1 Notions de similarité

Il est fréquent que la notion de similarité soit masquée par la mesure de la similarité. En effet, dans le chapitre précédent ont été données plusieurs mesures, dont la valeur permet de décider si les objets comparés sont similaires ou non. Dans l'approche proposée ici, le choix a été fait de distinguer clairement d'une part le critère de similarité, et d'autre part la mesure de la similarité. Ce choix est motivé par l'envie de pouvoir modéliser l'espace de recherche des solutions (définition de la similarité), tout en se gardant la possibilité de changer l'ordonnancement des solutions (mesure de la similarité).

STAMS utilise une nouvelle mesure de similarité deux-à-deux par rapport à un modèle externe pour élaguer l'espace de recherche, et une mesure de similarité globale calculée sur la base de la notion de similarité deux-à-deux, pour ordonner les solutions trouvées. À l'origine de la mesure de la similarité utilisée, est la distance de HAMMING. En effet, cette distance est à la fois simple à mettre en œuvre, et elle correspond à une réalité biologique : les motifs (*e.g.* sites de liaisons) « biologiquement » similaires sont généralement de même longueur et diffèrent principalement par quelques substitutions (*cf.* Section 1.2.1 – page 16). Après avoir proposé plusieurs modélisations de l'espace de recherche des solutions au problème d'extraction de motifs, seront proposées plusieurs mesures permettant de doter l'ensemble des solutions potentielles d'une relation d'ordre.

Retour sur la distance de HAMMING

La distance de HAMMING entre deux mots de même longueur est définie comme étant le nombre minimum de substitutions nécessaires pour passer d'un mot à l'autre (*cf.* Définition 2.17 – page 27).

Il est possible de donner une autre formalisation de cette distance au moyen des fonctions définies sur les couples de symboles α, β de l'alphabet Σ utilisé (couvert par l'ensemble $\mathcal{C}(\Sigma)$, *cf.* Définitions 2.5

à 2.8 – page 24) :

$$match(\alpha, \beta) = \begin{cases} 1 & \text{si } \alpha =_{\mathcal{C}(\Sigma)} \beta, \\ 0 & \text{si } \alpha \neq_{\mathcal{C}(\Sigma)} \beta. \end{cases}$$

$$mismatch(\alpha, \beta) = 1 - match(\alpha, \beta) = \begin{cases} 1 & \text{si } \alpha \neq_{\mathcal{C}(\Sigma)} \beta, \\ 0 & \text{si } \alpha =_{\mathcal{C}(\Sigma)} \beta. \end{cases}$$

Ainsi la distance de HAMMING pour deux mots w_1 et w_2 de même longueur n est définie par (*cf.* Définition 2.17 – page 27) :

$$D_H(w_1, w_2) = \sum_{i=1}^{n} mismatch(w_1[i], w_2[i]).$$

Un usage fréquent de cette distance en bioinformatique est de sélectionner les couples de mots w_1, w_2 de même longueur fixée à l'avance, et de ne considérer comme paires similaires que celles ayant une distance de HAMMING inférieure à un seuil donné également au préalable (*e.g.* [108]).

Selon la remarque 2.15 – page 26, il est possible de transformer ce calcul de distance en calcul de similarité :

$$S_H(w_1, w_2) := n - D_H(w_1, w_2) \left(= \sum_{i=1}^{n} match(w_1[i], w_2[i]) \right),$$

où w_1 et w_2 sont deux mots même longueur n.

Cette mesure est généralement utilisée à la fois pour définir l'espace de recherche (en ne considérant comme similaires que les couples de mots ayant une distance inférieure ou égale à un seuil préalablement fixé), tout en établissant une relation d'ordre sur l'ensemble des paires de mots similaires. Si son utilisation afin de délimiter l'espace de recherche est globalement acceptée par la communauté, il n'en est pas de même pour la relation d'ordre qu'elle engendre. C'est pourquoi le choix est fait d'utiliser cette distance dans la formalisation des définitions de la similarité présentées ci-après, en ne tenant pas compte de la mesure qu'elle apporte.

Similarité entre mots de même longueur

La définition de la similarité permet de délimiter l'espace de recherche des solutions. Dans cette section, la progression des notions de similarité proposées a pour objectif de faciliter la perception de l'aspect sémantique lié à chacune des définitions. Les définitions pourront ensuite être appliquées à toutes les paires de mots de même longueur afin de déterminer s'ils sont similaires ou non.

Remarque 3.1. Posons comme prérequis que pour tout alignement de deux séquences, ne sont considérées comme candidates que les paires de motifs de même longueur et dont le premier et le dernier symboles correspondent. Cette restriction sur le premier et le dernier symbole tient essentiellement à une considération algorithmique. En effet, cela permet d'éliminer certains effets de bords, tout en permettant plus de souplesse dans la formalisation des définitions qui suivent.

La première définition de la similarité permet de représenter l'ensemble des couples de mots de longueur et à distance de HAMMING fixées au préalable.

Définition 3.2. Étant donnés deux entiers l et e ($0 \leq e < l$), ainsi que deux mots w_1 et w_2 de même longueur $k \geq l$, w_1 et w_2 sont $\langle l, e \rangle_H$-similaires si les mots $w_1[i..i+l-1]$ et $w_2[i..i+l-1]$ sont à distance de HAMMING au plus e, pour tout i, $1 \leq i \leq k-l+1$.

La fonction associée à cette notion de similarité est alors :

$$\langle l, e \rangle_H\text{-similaire}(w_1, w_2) = \prod_{i=1}^{k-l+1} \left[\!\!\left[D_H(w_1[i..i+l-1], w_2[i..i+l-1]) \leq e \right]\!\!\right],$$

où $[\![\cdots]\!]$ est la notation d'IVERSON. Pour un prédicat P donné, $[\![P]\!]$ vaut 1 si P est vérifiée, 0 sinon. (*cf.* Nomenclature & Biographies).

Il est également possible d'utiliser une notion de similarité ne nécessitant pas de connaître la longueur (même minimale) des motifs cherchés. Cette limitation sur la connaissance de la longueur des motifs à chercher nécessite une connaissance préalable des motifs à extraire généralement trop importante. Or l'usage de la distance de HAMMING nécessite de fixer *a priori* la longueur des motifs recherchés, ce qui constitue le principal grief opposé à l'usage de cette distance de HAMMING.

Définition 3.3. Étant donnés un entier $e \geq 0$, ainsi que deux mots w_1 et w_2 de même longueur k, w_1 et w_2 sont e-similaires si le nombre maximum de « non correspondances » consécutives dans l'alignement sans trous de w_1 et de w_2 est inférieur ou égal à e.

Comme précédemment, il est possible d'associer une fonction à cette définition.

$$e\text{-similaire}(w_1, w_2) = \prod_{i=1}^{k-e} \left[\!\!\left[D_H(w_1[i..i+e], w_2[i..i+e]) \leq e \right]\!\!\right].$$

Propriété 3.1. *Pour un e fixé, si deux mots w_1 et w_2 de même longueur $k \geq l$ sont $\langle l, e \rangle_H$-similaires, alors ils sont également e-similaires (la relation inverse n'est pas vérifiée).*

La e-similarité peut toutefois sembler trop peu restrictive de l'espace de recherche c'est pourquoi il peut être souhaitable d'introduire un taux de « non correspondances » maximal acceptable.

Définition 3.4. Étant donnés un entier $e \geq 0$, un réel ρ ($0 \leq \rho \leq 1$), ainsi que deux mots w_1 et w_2 de même longueur k, w_1 et w_2 sont $\langle e \rangle_\rho$-similaires s'ils sont e-similaires, et si le ratio entre le nombre total de « non correspondances » et leur longueur k est inférieur ou égal au paramètre ρ.

À l'instar des définitions précédentes, à cette définition il est possible d'associer la fonction suivante :

$$\langle e \rangle_\rho\text{-similaire}(w_1, w_2) = e\text{-similaire}(w_1, w_2) \times \left[\!\!\left[\frac{D_H(w_1, w_2)}{k} \leq \rho \right]\!\!\right].$$

Par définition, la propriété 3.1 peut être adaptée en substituant la $\langle e \rangle_\rho$-similarité à la $\langle l, e \rangle_H$-similarité.

Propriété 3.2. *Pour e et ρ fixés, si deux mots w_1 et w_2 de même longueur sont $\langle e \rangle_\rho$-similaires, alors ils sont également e-similaires (la relation inverse n'est pas vérifiée).*

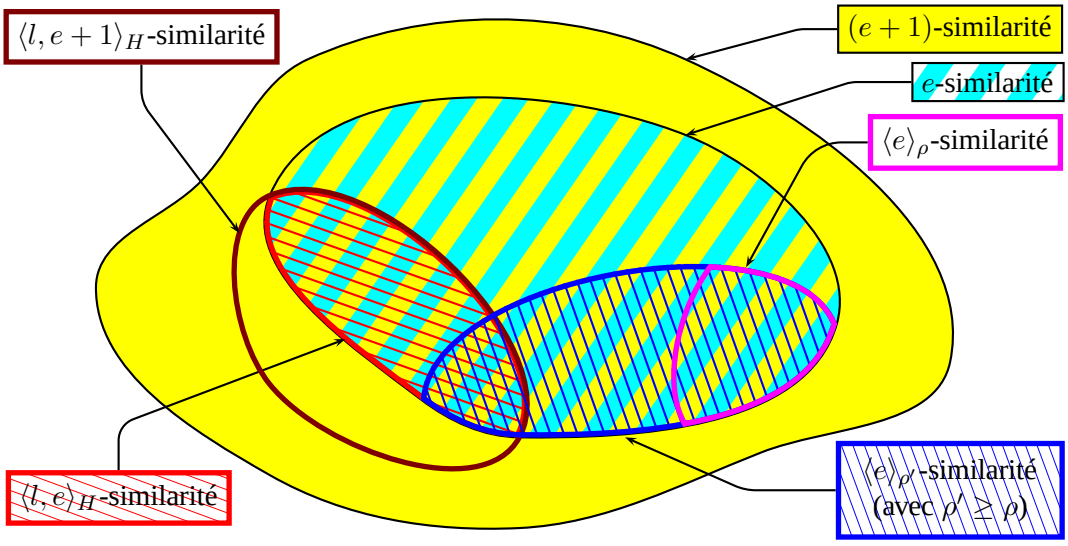


Figure 3.1 – Relations entre les notions de similarité.

De surcroît, pour tout entier e, deux mots e-similaires sont également $(e+1)$-similaires ; deux mots $\langle l, e\rangle_H$-similaires sont également $\langle l, e+1\rangle_H$-similaires. Si deux mots sont $\langle l, e\rangle_H$-similaires avec $l > e+2$, alors ils sont aussi $\langle l-1, e\rangle_H$-similaires. Enfin, pour tout $\rho' \geq \rho$ deux mots $\langle e\rangle_\rho$-similaires sont nécessairement $\langle e\rangle_{\rho'}$-similaires (*cf.* Figure 3.1).

Ces trois notions sont illustrées sur l'exemple ci-dessous.

Exemple 3.1 (Notions de $\langle l, e\rangle_H$-similarité, e-similarité et de $\langle e\rangle_\rho$-similarité).
Soit le couple $(w_1 = abacbcabac, w_2 = abccaaabcc)$. *Les mots* w_1 *et* w_2 *diffèrent aux positions* 3, 5, 6 *et* 9. *Ils sont* 2*-similaires mais ne sont pas* 1*-similaires ; ils sont également* $\langle 3, 2\rangle_H$*-similaires, mais ne sont pas* $\langle 4, 2\rangle_H$*-similaires ; enfin, ils sont* $\langle 2\rangle_{1/2}$*-similaires mais ne sont pas* $\langle 2\rangle_{1/3}$*-similaires*.

3.2.2 Fonctions *Block-Based*

Les différentes définitions de la similarité données précédemment permettent chacune de délimiter un espace de recherche des solutions. Une solution est – selon la définition du problème EM – une collection de motifs – un sur chaque séquence en entrée – globalement similaires. La similarité globale d'une collection de motifs est établie à partir d'une des notions de similarité locale présentée ci-avant ; et conformément à la remarque 3.1 – page 47, tous les motifs d'une même collection sont de même longueur. La représentation d'une solution adoptée dans ce chapitre consiste donc en l'alignement sans trous de la collection de motifs (*cf.* Section 2.3 – page 32).

Si les définitions de la similarité permettent de délimiter l'espace des solutions, elles ne permettent toutefois pas de distinguer – au sein de cet ensemble – les « bonnes » des « mauvaises » solutions. Il apparaît donc nécessaire de munir l'espace de recherche d'un critère de discrimination des solutions. Il devient même souhaitable de le doter d'une relation d'ordre, ce qui est réalisable en associant un score à chaque solution (*cf.* Section 2.2.1 – page 25).

Parmi l'ensemble des fonctions de score existantes (*cf.* Sections 2.2 – page 25 – et 2.3 – page 32), un grand nombre d'entre elles sont applicables à tout ou partie des motifs composant une solution. Une nouvelle famille de fonctions de score est introduite ci-après, enrichissant ainsi la panoplie de fonctions utilisables : les fonctions basées sur un découpage en blocs consécutifs de « correspondances » et de « non correspondances » ou fonctions *Block-Based*.

Bien qu'il soit écrit à la section 3.2.1 – page 46 – que la distance de HAMMING n'est pas nécessairement le meilleur choix pour mesurer la similarité, il n'en demeure pas moins le souhait de pouvoir appliquer cette distance afin de munir l'ensemble des solutions d'une relation d'ordre. Il est possible de définir un cadre plus général de calcul de score qui permette de définir entre autres les fonctions D_H et S_H décrites plus haut. Ce calcul peut être effectué en introduisant une notion de découpage en blocs consécutifs de « correspondances » et de « non correspondances » des symboles dans l'alignement sans trous entre deux mots w_1 et w_2 de même longueur.

Définition 3.5. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$ de même longueur n, il est possible de construire le mot alignement $A(w_1, w_2) \in \{0,1\}^n$ tel que $A(w_1, w_2)[i] = match(w_1[i], w_2[i])$ pour tout i. Ainsi, le mot $A(w_1, w_2)$ est une suite de blocs de 0 et de 1. Il peut donc s'écrire :

$$A(w_1, w_2) = 0^{k_0}1^{\ell_1}0^{k_1}\cdots 1^{\ell_t}0^{k_t}1^{\ell_{t+1}},$$

avec $t \geq 0$, $\forall\, i$ $(1 \leq i \leq t)$, $k_i, \ell_i > 0$ et $k_0, \ell_{t+1} \geq 0$.

Cette représentation du mot alignement traduit le découpage en blocs consécutifs de « correspondances » et de « non correspondances ». En effet, un mot de $\{0,1\}^+$ débute soit par un bloc de 0 (de longueur $k_0 > 0$), soit par un bloc de 1 (*i.e.*, $k_0 = 0$). De même, un tel mot se termine soit par un bloc de 0 (*i.e.*, $\ell_{t+1} = 0$), soit par un bloc de 1 (de longueur $\ell_{t+1} > 0$). Outre le bloc éventuel de 0 en début de mot et le bloc éventuel de 1 en fin de mot, le mot est également constitué soit d'une alternance de blocs de 1 et de blocs de 0 ($t > 0$), soit du mot vide ($t = 0$).

Exemple 3.2 (Construction du mot alignement).
Soient les deux mots $w_1 = PCVEPIUYGF$ et $w_2 = UCVRTIUPGS$, le mot alignement qui leur est associé est :

w_1	=	*P*	*C*	*V*	*E*	*P*	*I*	*U*	*Y*	*G*	*F*
w_2	=	*U*	*C*	*V*	*R*	*T*	*I*	*U*	*P*	*G*	*S*
$A(w_1, w_2)$	=	0	1	1	0	0	1	1	0	1	0

Le mot alignement est parfois désigné dans la littérature par le terme « similarité ». Toutefois, afin d'éviter toute confusion, seule le vocable « mot alignement » est utilisé dans ce document.

Définition 3.6. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$ de même longueur n, et leur mot alignement $A(w_1, w_2)(= 0^{k_0}\, 1^{\ell_1}\, 0^{k_1} \cdots 1^{\ell_t}\, 0^{k_t}\, 1^{\ell_{t+1}})$, nous définissons la suite de « correspondances » par

$$l\_matches(w_1, w_2) \equiv l\_matches(A(w_1, w_2)) := (\ell_1, \ldots, \ell_{t+1});$$

et de façon similaire la suite de « non correspondances » par

$$l\_mismatches(w_1, w_2) \equiv l\_mismatches(A(w_1, w_2)) := (k_0, \ldots, k_t).$$

Exemple 3.3 (Construction du mot alignement).
En reprenant l'exemple de la page précédente, les listes de suites de « correspondances » et de suites de « non correspondances » associées au mot alignement $A(w_1, w_2) = 0110011010 = 0^1 1^2 0^2 1^2 0^1 1^1 0^1 1^0$ *sont donc :*

$$l\_matches(w_1, w_2) = (2, 2, 1, 0)$$
$$l\_mismatches(w_1, w_2) = (1, 2, 1, 1).$$

Sur la base de ce découpage, il est possible de définir une famille de fonctions de scores.

Définition 3.7. Étant donnés un mot m défini sur $\{0,1\}^*$, et deux fonctions $f^=$ et $f^\neq$ définies de $\mathbb{N}$ dans $\mathbb{R}$, appelées respectivement fonctions de présence et fonctions d'absence, la fonction de score S : $\{0,1\}^* \to \mathbb{R}$ est définie par

$$S(m) = \sum_{i=1}^{t+1} f^=(\ell_i) + \sum_{i=0}^{t} f^\neq(k_i), \tag{3.1}$$

où $(\ell_1, \ldots, \ell_{t+1}) = l\_matches(m)$ et $(k_0, \ldots, k_t) = l\_mismatches(m)$.

Ainsi, étant donnés deux mots w_1 et w_2 de même longueur n, il est possible de définir un score de $\Sigma^* \times \Sigma^* \to \mathbb{R}$, en utilisant le score de leur mot alignement. Par abus de notation et pour simplifier, le score de l'alignement de w_1 et w_2 sera également noté S :

$$S(w_1, w_2) = S(A(w_1, w_2)).$$

Dans la suite de ce document, les fonctions de présence et d'absence sont également appelées fonctions composantes du score. Les fonctions de score sont quant à elle appelées fonctions basées sur un découpage en blocs, ou fonctions *Block-Based* (*cf.* Exemple 3.4).

Exemple 3.4 (Distance de HAMMING et fonctions *Block-Based*).
La distance de HAMMING *se calcule en prenant* $f^=(k) = 0$ *et* $f^\neq(k) = k$ *; tandis que le calcul de* S_H *(cf. page 47) s'obtient en choisissant* $f^=(k) = k$ *et* $f^\neq(k) = 0$*. Pour reprendre l'exemple ci-dessus,*

$$\begin{array}{lcccccccccccccccl}
S(0110011010) & = & f^=(2) & + & f^=(2) & + & f^=(1) & + & f^=(0) & + & f^\neq(1) & + & f^\neq(2) & + & f^\neq(1) & + & f^\neq(1) & \\
D_H(w_1, w_2) & = & 0 & + & 0 & + & 0 & + & 0 & + & 1 & + & 2 & + & 1 & + & 1 & = 5, \\
S_H(w_1, w_2) & = & 2 & + & 2 & + & 1 & + & 0 & + & 0 & + & 0 & + & 0 & + & 0 & = 5.
\end{array}$$

Cette famille de fonctions de score est applicable à toute paire de mots de même longueur. Il est donc possible, comme pour la distance de HAMMING, de fixer un seuil pour le score, en dessus ou en deçà duquel les mots ne sont pas considérés comme similaires. Les scores permettent alors de discriminer les meilleurs candidats parmi un ensemble de paires de mots. Les algorithmes utilisant la distance de HAMMING en vue de délimiter l'espace de recherche nécessitent généralement de déterminer la longueur des mots et la valeur de seuil *a priori*. Si la valeur du seuil peut être raisonnablement fixée pour une longueur des mots donnée, il reste le problème de choix de cette longueur. Ce prérequis impose donc soit de connaître la longueur des motifs à extraire, ou alors de relancer les algorithmes d'extraction en faisant varier la longueur à chaque exécution. L'algorithme présenté par la suite a été conçu de sorte qu'il n'y ait pas besoin de donner une longueur fixée, ni même un intervalle de valeur pour cette longueur.

Le choix des fonctions composantes de score est vaste, aussi seules quelques fonctions sont présentées dans la table 3.1 afin d'illustrer l'utilisation de ce score. Les fonctions choisies sont celles implémentées dans l'algorithme STAMS présenté à la section 3.3 – page 57. L'exemple choisi est le couple $(w_1 = accagtacga, w_2 = agcagaacct)$. Ainsi le mot alignement formé à partir de w_1 et de w_2 est $A(w_1, w_2) = 1011101100$, et donc $l\_matches(w1, w2) = (1, 3, 2, 0)$ et $l\_mismatches(w1, w2) = (0, 1, 1, 2)$.

		$f^{=}$				
		$k \mapsto k$	$k \mapsto k^2$	$k \mapsto k\,(k+1)/2$	$k \mapsto \log k$	$k \mapsto k\sqrt{k}$
$f^{\neq}$	$k \mapsto 0$	6,000	14,000	10,000	2,585	9,025
	$k \mapsto k$	2,000	10,000	6,000	−1,415	5,025
	$k \mapsto k^2$	0,000	8,000	4,000	−3,415	−3,025
	$k \mapsto k\,(k+1)/2$	1,000	9,000	5,000	−2,415	4,025
	$k \mapsto \log k$	5,000	13,000	9,000	1,585	8,025
	$k \mapsto k\sqrt{k}$	1,172	9,172	5,172	−2,243	4,197

Table 3.1 – Exemples d'utilisation des fonctions de présence et d'absence.

La motivation sous-jacente au choix d'une mesure de similarité est que celle-ci doit permettre d'évaluer une solution au problème EM (*i.e.*, mesurer la similarité d'une collection de motifs qui sont vus comme un alignment sans trous de plusieurs mots de même longueur). À l'instar de l'utilisation des matrices, dans le cadre du calcul d'un score sur un alignement de plusieurs mots, il est envisageable d'effectuer un calcul sur les scores des mots pris deux à deux, ou par rapport à un modèle consensuel.

L'utilisation des scores des mots (dans un alignement) deux à deux pour calculer un score global de l'alignement présente deux inconvénients majeurs. Tout d'abord, le calcul de tous les scores des mots deux à deux est coûteux. Ensuite, il n'apparaît en aucune manière de motif consensuel associable à ce score ; la notion de motif consensuel étant rendue nécessaire par l'usage établi dans la communauté des biologistes. Ainsi, plusieurs méthodes utilisent une approche en étoile pour l'établissement du score global d'un alignement directement associé à un modèle consensuel, tels PRATT [71] ou encore GIBBS [81]. Outre les bons résultats de ces méthodes, il apparaît clairement que le calcul du score s'en voit plus simple que dans la première approche. C'est principalement pour ces raisons que STAMS utilise une approche en étoile pour le calcul des scores des solutions. Toutefois, le score global obtenu par ce type d'approche est surtout pertinent lorsqu'il est associé à un modèle consensuel « parfait » (dans le sens où il est factice). S'il existe des imperfections dans le modèle consensuel, alors il faut pouvoir les répercuter sur le calcul du score global. C'est pourquoi une nouvelle notion permettant de refléter les imperfections du consensus sur le score est rendue obligatoire. Cette nouvelle notion est basée sur l'idée que si un symbole à une position donnée du modèle consensuel est bien conservé dans l'alignement, alors ce symbole est bien choisi. Si à l'inverse, il n'est pas bien conservé dans l'alignement, alors ce symbole est une erreur dans le consensus.

Définition 3.8. Étant donnés un pourcentage Q et un alignement $(w_0, \ldots, w_i)$, tel que tous les w_k $(0 \leq k \leq i)$ soient de longueur l, l'ensemble des positions significatives du mot w_0 dans l'alignement $(w_0, \ldots, w_i)$ est l'ensemble des positions j telles que $w_0[j]$ est en « correspondance » dans $\lceil Q\,i \rceil$ motifs à la position j.

Cette dernière notion permet de distinguer les positions du modèle w_0 qui sont significatives dans l'alignement.

La proposition suivante concerne surtout le cas où le modèle w_0 est figé. Supposons que le calcul du score d'un alignement $(w_0, w_1, \ldots, w_i)$, tel que tous les mots w_k soient de longueur l, avec w_0 un modèle consensuel (éventuellement fictif) soit déjà effectué. Cela implique que les scores des paires $\{(w_0, w_k) \,|\, 1 \leq k \leq i\}$ sont également déjà calculés. Pour obtenir le score de l'alignement $(w_0, \ldots, w_{i+1})$, avec w_{i+1} de longueur l, il suffit de calculer le score de la paire (w_0, w_{i+1}) et de mettre à jour le score global. Cependant l'alignement $(w_0, \ldots, w_i)$ donne déjà de l'information sur la validité des positions de w_0 dans $(w_0, \ldots, w_i)$. La proposition consiste donc à tenir compte de cette information dans le calcul du score de la paire (w_0, w_{i+1}), en relativisant l'importance des « correspondances » et des « non correspondances » selon qu'elles se produisent sur des positions significatives ou non. En pratique, ne seront considérées pour former le mot alignement entre w_0 et w_{i+1} que les « correspondances » sur les positions significatives et les « non correspondances » sur les positions non significatives de w_0 dans $(w_0, \ldots, w_i)$, les autres situations étant tout simplement ignorées (*cf.* Table 3.2). En effet, le mot w_0 est considéré comme étant un modèle représentatif de l'alignement $(w_0, \ldots, w_i)$. L'objectif étant que le score local entre w_0 et w_{i+1} soit le reflet du score global de l'alignement $(w_0, \ldots, w_{i+1})$. Étant donnée une position k de ce modèle, si $\lceil Q\, i \rceil$ mots de l'alignement sont en « correspondance » avec le mot w_0 à cette position, celle-ci est alors significative (*i.e.*, le modèle w_0 définit un consensus dans l'alignement pour cette position). *A contrario*, si la position k n'est pas significative (*i.e.*, le modèle w_0 ne définit pas de consensus dans l'alignement pour cette position). Si le mot w_{i+1} est en « correspondance » avec le modèle à une position non significative, le score global de l'alignement $(w_0, \ldots, w_{i+1})$ ne saurait tenir compte de cette position, attendu que celle-ci n'est pas consensuelle dans l'alignement. De même, si le mot w_{i+1} est en « non correspondance » avec le modèle à une position significative, le score global de l'alignement $(w_0, \ldots, w_{i+1})$ ne doit pas être dévalué du fait de cette « non correspondance », vu que plus de $Q\%$ des mots de l'alignement admettent un consensus sur cette position.

	position significative ($\geq \lceil Q\, i \rceil$)	position non significative ($< \lceil Q\, i \rceil$)
« correspondance »	1	–
« non correspondance »	–	0

Table 3.2 – Construction du mot alignement lors du calcul d'un score Q-dépendant.

Définition 3.9. Étant donnés deux mots $w_1, w_2 \in \Sigma^*$ de même longueur n, et la liste des positions significatives ℓ de w_1, il est possible de construire le mot $\mathscr{A}_\ell(w_1, w_2) \in \{0, -, 1\}^n$ tel que

$$\begin{aligned} \mathscr{A}_\ell(w_1, w_2)[i] &= 1 && \text{si } (i \in \ell) \wedge match(w_1[i], w_2[i]), \\ \mathscr{A}_\ell(w_1, w_2)[i] &= 0 && \text{si } (i \notin \ell) \wedge mismatch(w_1[i], w_2[i]), \\ \mathscr{A}_\ell(w_1, w_2)[i] &= - && \text{autrement.} \end{aligned}$$

Le mot alignement restreint est alors la sous-séquence maximale (non nécessairement contiguë) appartenant à $\{0, 1\}^*$ de $\mathscr{A}_\ell(w_1, w_2)$. Ce mot est noté $A_{|\ell}(w_1, w_2)$.

Le mot alignement restreint permet également l'utilisation des fonctions *Block-Based*. Les fonctions de score utilisant le découpage en blocs consécutifs de 1 et de 0 du mot alignement restreint $A_{|\ell}$ sont alors appelées fonctions Q-dépendantes et sont notées S_Q. Celles basées sur le découpage en blocs consécutifs de 0 et de 1 du mot alignement A sont (par opposition) appelées fonctions Q-indépendantes. L'avantage des fonctions Q-dépendantes est que le score donné par le dernier mot aligné avec le modèle peut également être utilisé comme score de l'alignement.

Exemple 3.5 (Calcul d'un score Q-dépendant).
Soient $f^{=}(k) = 2\,k$, $f^{\neq}(k) = -1$ et $Q = 70\%$. Alors le score de la paire (w_0, w_{10}) en fonction de l'alignement $(w_0, \ldots, w_9)$ suivant :

w_0	=	*P*	*C*	*V*	*E*	*P*	*I*	*U*	*Y*	*G*	*F*
w_1	=	*U*	*M*	*T*	*C*	*P*	*V*	*N*	*P*	*G*	*K*
w_2	=	*P*	*L*	*E*	*T*	*D*	*I*	*U*	*H*	*Y*	*F*
w_3	=	*P*	*I*	*U*	*Q*	*P*	*I*	*E*	*P*	*G*	*F*
w_4	=	*V*	*L*	*V*	*R*	*P*	*I*	*U*	*M*	*G*	*S*
w_5	=	*P*	*E*	*U*	*E*	*V*	*C*	*U*	*P*	*G*	*F*
w_6	=	*S*	*A*	*N*	*I*	*P*	*I*	*U*	*P*	*G*	*F*
w_7	=	*P*	*C*	*E*	*N*	*P*	*S*	*U*	*F*	*V*	*F*
w_8	=	*S*	*D*	*L*	*E*	*P*	*I*	*U*	*P*	*G*	*F*
w_9	=	*D*	*C*	*V*	*R*	*S*	*I*	*C*	*P*	*G*	*S*
w_{10}	=	*U*	*C*	*V*	*R*	*T*	*I*	*U*	*P*	*G*	*S*
$A(w_0, w_{10})$	=	0	1	1	0	0	1	1	0	1	0
ℓ	=					*	*	*		*	*
$\mathscr{A}_\ell(w_0, w_{10})$	=	0	–	–	0	–	1	1	0	1	–
$A_{\mid\ell}(w_0, w_{10})$	=	0			0		1	1	0	1	

est donné par le score de $A_{\mid\{5,6,7,9,10\}}(w_0, w_{10}) = 001101$, donc dans le cas présent $S_{70}(w_0, w_{10}) = 4$. Dans le cas de la fonction Q-indépendante homologue (i.e., mêmes fonctions de présence et d'absence), $S(w_0, w_{10}) = 6$.

Les fonctions *Block-Based* ont pour objectif de proposer des mesures de la similarité, offrant l'avantage d'être paramétrables (*via* les fonctions composantes de score). Outre la modularité offerte par ce schéma de représentation, le choix des fonctions composantes permet de définir intuitivement la « forme » des motifs à extraire (importance relative des blocs de « correspondances » par rapport aux blocs de « non correspondances », influence de la longueur des blocs de « correspondances » ou de « non correspondances », ...) Enfin, les résultats obtenus par l'algorithme décrit à la section 3.3 – page 57 – avec cette famille de fonctions sont plutôt prometteurs. Ils laissent à penser que ces fonctions sont bien adaptées à la mesure de la similarité entre motifs biologiques, notamment pour ce qui est des sites de liaisons ADN-protéines.

3.2.3 Présélection des candidats

Étant donné un ensemble $\mathcal{S}$ de séquences, l'espace de recherche initial représente l'ensemble des combinaisons possibles en choisissant un motif quelconque sur chaque séquence. Afin de réduire cet espace, la première condition posée est que tous les motifs d'une solution aient la même longueur (non fixée *a priori*, *cf.* Section 2.3 – page 32). Ainsi, chaque solution potentielle peut-être vue comme un alignement local sans trous des séquences de $\mathcal{S}$. Chaque alignement induit donc une notion de décalage (*cf.* Figure 3.2).

Définition 3.10. Étant données deux séquences $s_1, s_2 \in \Sigma^*$ disposées de sorte que chaque symbole $s_1[i]$ soit aligné avec un unique symbole $s_2[j]$, alors la différence $\delta = j - i$ est constante, et est appelée décalage de s_2 par rapport à s_1.

Propriété 3.3. *Le décalage δ entre deux séquences s_1 et s_2 est tel que $1 - |s_1| \leq \delta \leq |s_2| - 1$. En outre, pour tout i tel que $\max(1, 1 - \delta) \leq i \leq \min(|s_1|, |s_2| - \delta)$, les symboles $s_1[i]$ et $s_2[i + \delta]$ sont alignés.*

–	–	–	c	t	a	c	g	a	t	c	g	a	c	g	a	c	t	a	c	g	–	–	–
–	–	a	t	c	g	a	t	a	g	c	t	a	c	g	a	g	–	–	–	–	–	–	–
–	–	–	–	–	g	g	a	g	c	a	c	g	c	t	a	g	c	a	c	g	a	g	c
a	c	a	t	c	g	a	t	c	g	a	c	t	a	g	c	t	a	g	–	–	–	–	–
–	–	g	c	t	a	g	c	a	g	c	a	c	t	a	g	c	a	g	c	–	–	–	–
–	–	–	t	g	a	g	c	t	a	g	c	a	t	c	g	a	t	c	a	g	t	c	–

Figure 3.2 – Alignement sans trou d'un ensemble de séquences.

Dans la mesure où la relation entre l'alignement de deux séquences et le décalage induit par cet alignement est bijective, les décalages seront également utilisés pour identifier (sans ambiguïté) les alignements. Il faut maintenant définir le principe du chevauchement de motifs sur une séquence.

Définition 3.11. Étant donnés une séquence $s \in \Sigma^*$, et deux mots w_1, w_2 de cette séquence tels que $w_1 = s[i_1..j_1]$ et $w_2 = s[i_2..j_2]$, les motifs w_1 et w_2 sont dits chevauchants dans s lorsque $\max(i_1, i_2) \leq \min(j_1, j_2)$. Nous dénoterons la relation de chevauchement par $w_1 \between w_2$. Cette relation est réflexive ($w_1 \between w_1$) et symétrique ($w_1 \between w_2 \Leftrightarrow w_2 \between w_1$).

Il est intéressant de remarquer d'une part que la notion de chevauchement comprend également le cas de l'inclusion, et d'autre part qu'étant donnés un alignement δ entre deux séquences s_1 et s_2, ainsi que deux paires de mots alignés $\left(w_1^{[1]}, w_2^{[1]}\right)$ et $\left(w_1^{[2]}, w_2^{[2]}\right)$ telles que $w_1^{[1]} \between w_1^{[2]}$, nous avons $w_2^{[1]} \between w_2^{[2]}$ (*cf.* Figure 3.3).

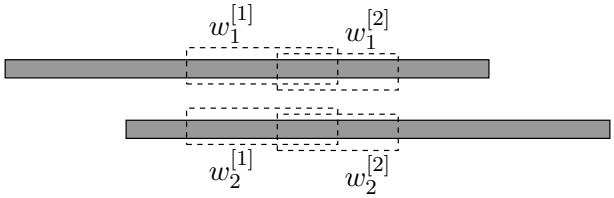


Figure 3.3 – Illustration de chevauchements dans un alignement de deux séquences.

Définition 3.12. Étant donnés un alignement de deux séquences $s_1, s_2 \in \Sigma^*$, et deux paires de mots alignés $\left(w_1^{[1]}, w_2^{[1]}\right)$ et $\left(w_1^{[2]}, w_2^{[2]}\right)$ dans cet alignement, telles que $w_1^{[1]} \between w_1^{[2]}$, le mot $w_1^{[1]}$ est dit extensible par $w_1^{[2]}$ (et de manière similaire pour $w_1^{[2]}$, $w_2^{[1]}$ et $w_2^{[2]}$). Par abus de langage, la paire $(w_1^{[1]}, w_2^{[1]})$ (et $(w_1^{[2]}, w_2^{[2]})$) est également dite extensible. Soit δ le décalage entre les séquences s_1 et s_2 et soient i_1, i_2, j_1 et j_2 les positions dans s_1 telles que $w_1^{[1]} = s_1[i_1..j_1]$ et $w_2^{[1]} = s_1[i_2..j_2]$ (par définition $w_1^{[2]} = s_2[i_1 + \delta..j_1 + \delta]$ et $w_2^{[2]} = s_2[i_2 + \delta..j_2 + \delta]$). Le mot $s_1[\min(i_1, i_2).. \max(j_1, j_2)]$ est le résultat de l'extension de $w_1^{[1]}$ par $w_2^{[1]}$ (et réciproquement de $w_2^{[1]}$ par $w_1^{[1]}$). Le mot $w_1^{[1]}$ (respectivement $w_2^{[1]}$) est alors également dit étendu par $w_2^{[1]}$ (respectivement $w_1^{[1]}$). Si $w_1^{[1]} \neq s_1[\min(i_1, i_2).. \max(j_1, j_2)]$ (respectivement si $w_2^{[1]} \neq s_1[\min(i_1, i_2).. \max(j_1, j_2)]$), alors le mot $w_1^{[1]}$ (respectivement $w_2^{[1]}$) est dit strictement étendu.

Définition 3.13. Étant donnés une notion de similarité, un alignement de deux séquences $s_1, s_2 \in \Sigma^*$, et une paire de mots alignés (w_1, w_2) dans cet alignement, la paire est dite candidate si les mots w_1 et w_2 sont similaires. Par abus de langage, les mots w_1 et w_2 sont alors appelés des candidats.

L'idée sous-jacente à la notion d'extensibilité est que si deux paires de motifs candidates se chevauchent dans un alignement donné (*cf.* Figure 3.3), alors lors de la phase de traitement des candidats (*e.g.* lorsque les candidats sont cherchés sur l'ensemble des séquences), la partie commune aux deux candidats sera traitée deux fois. En éliminant cette redondance dans les calculs, il est donc possible de diminuer la complexité de la phase de traitement. Ceci peut être réalisé en définissant un critère de « maximalité » des candidats.

Définition 3.14. Étant donnés un alignement de deux séquences $s_1, s_2 \in \Sigma^*$, et une paire de mots (w_1, w_2) dans cet alignement, alors w_1 est dit maximal si et seulement s'il n'existe pas de mot candidat dans cet alignement qui puisse étendre strictement w_1.

Cette dernière définition mène à la remarque suivante :

Remarque 3.15. Si une paire de mots (w_1, w_2) dans un alignement de deux séquences est telle que le mot w_1 soit maximal, alors le mot w_2 est également maximal.

De même que précédemment, nous utiliserons également le terme « maximal » concernant la paire (w_1, w_2) si w_1 et w_2 sont maximaux. La figure 3.4 illustre cette notion de maximalité. Les séquences alignées sont représentées par les rubans gris. Les couples de motifs similaires y sont repérés par les boîtes en pointillé. Les paires maximales sont alors les boîtes en trait plein.

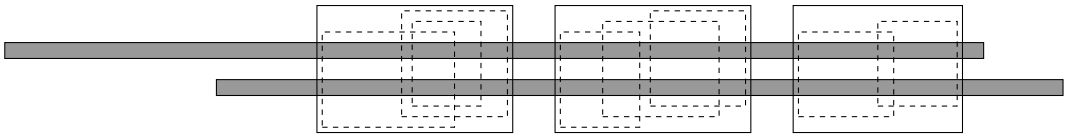
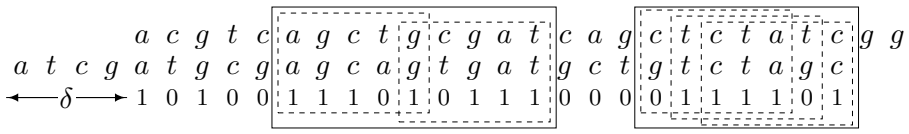


Figure 3.4 – Candidats présents dans un alignement de deux séquences.

Ce dernier concept étant établi, il est envisageable de réduire l'espace des solutions en ne considérant, pour une notion de similarité deux-à-deux donnée, que les mots maximaux obtenus à partir des motifs candidats. Il est à noter que les mots composant une paire maximale (pour un alignement donné) ne sont pas nécessairement similaires. Pour que cela soit le cas, il faut que l'extension préserve la notion de similarité. Par exemple, si nous considérons la distance de HAMMING pour un alignement donné, il est possible de créer l'ensemble des paires candidates maximales à partir des paires de motifs de longueur fixée l tels que leur distance est inférieure à un seuil e. Clairement, les motifs maximaux la composant ne sont pas nécessairement de longueur l. En outre, toutes les paires alignées de motifs de longueur l ne sont pas non plus obligatoirement similaires (*cf.* Exemple 3.6). Le choix de considérer les paires maximales plutôt que les paires similaires est motivé par un gain de temps potentiel au niveau du pré-traitement des séquences. Si ce gain semble évident lorsque la maximalité préserve la similarité, il n'en est pas de même lorsque la similarité n'est pas conservée. Cette discussion est reprise ultérieurement dans le manuscrit (*cf.* Remarque 3.16 – page ci-contre).

Exemple 3.6 (paires maximales issues de candidats dans un alignement de deux séquences).
Soient les séquences $s_1 = acgtcagctgcgatcagctctatcgg$ *et* $s_2 = atcgatgcgagcagtgatgctgtctagc$ *dans l'alignement* $\delta = 4$*. En considérant comme similaires les paires de motifs alignés de longueur* $\ell = 5$

et à distance de HAMMING *au plus* 1*, l'extraction des paires maximales à partir des paires de mots similaires donne* $(agctgcgat, agcagtgat)$ *et* $(ctctatc, gtctagc)$.



Il est également possible de représenter chaque paire par l'intervalle de ces positions de début et de fin dans s_1*, les positions dans* s_2 *s'en déduisant grâce à la valeur de* δ*. Ainsi, les paires maximales dans l'alignement* $\delta = 4$ *sont représentées par* $[6; 14]$ *et* $[18; 24]$.

Si tous les couples de motifs alignés de longueur 5 *contenu dans la paire maximale représentée par l'intervalle* $[18; 24]$ *sont tous similaires, il n'en est pas de même pour tous les couples de motifs alignés de longueur* 5 *contenus dans la paire maximale représentée par l'intervalle* $[6; 14]$.

3.3 Algorithme

Dans la première partie de ce chapitre ont été définies les notions de similarité qui permettent de délimiter un espace de recherche de solutions au problème de l'extraction de motifs dans un ensemble de séquences, problème qui constitue le *leitmotiv* des travaux présentés dans ce manuscrit. Dans cette précédente partie figurent également plusieurs fonctions de score, notamment les fonctions *Block-Based*. Ces concepts et outils sont utilisés dans cette section qui présente un algorithme d'extraction présenté dans [97], dont la version implémentée est nommée STAMS$^{\textcircled{3}}$ [is a **T**ool for **A**nalysis & **R**esearch in **S**equences]. Cette implémentation utilise exclusivement la e-similarité. Afin de mesurer la similarité, sont implémentées deux fonctions basées sur la mesure de l'entropie (dont celle utilisée dans PRATT, *cf.* Section 2.5.2 – page 38), les fonctions *Block-Based* Q-indépendantes et Q-dépendantes présentées dans la table 3.1 – page 52, ainsi qu'une mesure basée sur l'usage de matrices de similarité (*cf.* Annexe C).

Note : Un exemple simple est fourni à l'annexe F afin d'illustrer le déroulement de l'algorithme.

Remarque 3.16. Contrairement à la $\langle l, e\rangle_H$-similarité ou $\langle e\rangle_\rho$-similarité, la e-similarité est préservée par l'opération d'extension. En effet, étant données deux paires de motifs $\langle l, e\rangle_H$-similaires (respectivement $\langle e\rangle_\rho$-similaires) extensibles l'une par l'autre, la notion de similarité n'est pas conservée par la paire de motifs résultante. Bien que cela ne pose pas de réels problèmes algorithmiques, du point de vue sémantique, le concept de maximalité est alors sujet à caution. C'est pourquoi ne sera traité ici que le cas de la e-similarité.

Rappelons par ailleurs que selon la définition 3.3 – page 48, deux mots de même longueur sont e-similaires si leur alignement sans trou engendre au plus e « non correspondances » consécutives. De plus, l'objectif de l'algorithme n'est pas de renvoyer toutes les solutions, mais seulement les plus pertinentes (*i.e.*, les collections ayant les meilleurs scores de similarité). Ceci afin de ne pas être submergé par trop d'informations (*cf.* Section 2.5.3 – page 42). Il est donc nécessaire d'introduire un nouveau paramètre : p, qui correspond au nombre de solutions à renvoyer au maximum. C'est un paramètre que l'utilisateur doit pouvoir fixer.

Remarque 3.17. Les notions de « correspondance » et de « non correspondance » sont intrinsèquement liées à la couverture de l'alphabet $\mathcal{C}(\Sigma)$ utilisée. Dans le cas des séquences d'ADN, l'alphabet considéré est l'alphabet dégénéré tel que défini par le standard IUPAC (*cf.* Table A.1 – page 185), sa couverture est fournie dans l'exemple 2.1 – page 24. Dans le cas des séquences d'acides aminés, l'alphabet considéré est également l'alphabet dégénéré tel que défini par le standard IUPAC (*cf.* Table A.3 – page 187).

Le problème que se propose de résoudre l'algorithme peut dès lors être reformulé ainsi :

Extraction de motifs communs à un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), une fonction de score f, une tolérance d'erreurs e et un entier p.

Problème: Parmi les collections de motifs $(m_1 \in s_1, \ldots, m_t \in s_t)$ telles que les motifs $m_1, \ldots, m_t$ soient e-similaires à un même modèle m représentatif de la collection, renvoyer les p collections de meilleur score.

Note : Les fonctions composantes de score implémentées dans le programme STAMS sont les fonctions $nulle : k \mapsto 0$, $id : k \mapsto k$, $carre : k \mapsto k^2$, $suite : k \mapsto \frac{k(k-1)}{2}$, $racine : k \mapsto \sqrt{k}$ et $log : k \mapsto \log_2 k$. Les fonctions *Block-Based* disponibles sont donc celles dont la fonction d'absence est l'une des fonctions composantes de score disponibles et la fonction de présence est choisie parmi id, $carre$, $suite$, $racine$ et log. Pour chaque combinaison, les versions Q-dépendantes et Q-indépendantes sont proposées.

Parmi les approches envisageables pour répondre au problème de l'extraction de motifs, il est possible d'en distinguer au moins deux. La première consiste à « fabriquer » un (ou plusieurs) motif(s) consensuel(s). Il est alors question de modèle dit externe, dont on cherche les occurrences dans les séquences. La seconde approche consiste à extraire un ensemble de motifs – chacun situé sur une séquence – similaires deux à deux. Ces deux approches (*cf.* Figure 2.2) présentent toutes deux des avantages et inconvénients, et l'idée de l'algorithme est de tirer profit des deux approches. ROYTBERG a présenté en 1992 une approche hybride [117] où l'une des séquence de l'ensemble est choisie comme étant la séquence de référence, supposée contenir tous les modèles à extraire dans les autres séquences. L'algorithme présenté dans cette section est basée sur le même concept.

3.3.1 Principe

Au cours du déroulement de l'algorithme, chaque séquence de l'ensemble va être considérée au moins une fois comme étant la séquence de référence. Cette séquence est dite « de référence » car elle est supposée contenir au moins un des modèles consensuels correspondant à une des solutions de l'instance traitée.

Ainsi, l'algorithme se décompose en deux processus communiquants, l'un correspond au calcul d'un ensemble de solutions (*i.e.*, un ensemble de collections de motifs alignés) pour une séquence de référence donnée, tandis que l'autre est dédié aux changements de la séquence de référence et à la mise à jour des informations acquises à chaque solution fournie par le premier processus. Le calcul d'un ensemble de solutions (premier processus) est appelé dans la suite du manuscrit un cycle. L'algorithme dans son intégralité est donc présenté comme une approche multi-cycle (*cf.* Algorithme 3.1 – page ci-contre).

```
1  Nom : STABS
2  Entrées : S = {s_1, ..., s_t} % Ensemble des séquences à traiter. %
3    Σ, C(Σ) % Alphabet et couverture de l'alphabet utilisé. %
4    p % Nombre de résultats à renvoyer. %
5    f : Σ* × Σ* → ℝ % Fonction de score. %
6  Sortie : Sol % Ensemble des solutions. %
7  Variable : res % Un ensemble de collections. %
8  Début
9    Sol ← ∅
10   Répéter
11     Choisir un ordre de traitement des séquences de S   (cf. Section 3.3.4, p. 67)
12     Choisir une séquence de référence dans S            (cf. Section 3.3.4, p. 67)
13     Construire un nouvel ensemble de solutions res      (cf. Section 3.3.2, p. 59)
14     Mettre à jour l'ensemble Sol                        (cf. Section 3.3.4, p. 67)
15   Tant Que (l'ensemble Sol n'est pas satisfaisant)      (cf. Section 3.3.4, p. 67)
16   Afficher Sol
17 Fin
```

Algorithme 3.1 – Lignes générales de l'algorithme STABS.

3.3.2 Un cycle de l'algorithme : STABS-χ

Pour une séquence de référence fixée, les motifs e-similaires à ceux présents dans cette séquence de référence sont cherchés successivement sur chacune des autres séquences de l'ensemble. Les motifs considérés dans la séquence de référence sont appelés les candidats afin de les distinguer. À chaque nouvelle séquence de l'ensemble examinée, les sous-motifs des candidats qui sont similaires à des motifs de la séquence en cours, définissent les nouveaux candidats pour la prochaine séquence à intégrer dans l'exploration (d'où la nécessité de fixer un ordre de traitement des séquences et le besoin de procéder à plusieurs cycles). À chacun des candidats est assigné un score afin d'extraire en priorité ceux ayant le meilleur score.

Les séquences sont traitées séquentiellement, c'est pourquoi l'algorithme nécessite qu'une permutation π des séquences soit donnée. La première séquence selon la permutation (notée s'_1) est choisie comme étant la séquence de référence, l'ensemble initial des candidats est l'ensemble des motifs similaires entre la séquence de référence et elle-même. Ainsi, tous les motifs de la séquence sont nécessairement candidats. La séquence est également l'unique motif maximal. Le motif candidat étant maximal, il inclut par définition de la maximalité (page 56) tous les motifs candidats. Ce motif est alors comparé à la seconde séquence, pour chaque décalage δ entre la séquence de référence et la séquence en cours de traitement (ici s'_2). Seuls les motifs e-similaires maximaux pour chacun des alignements associés au décalages δ sont conservés comme candidats pour le traitement de la séquence suivante, et ainsi de suite. En effet, les motifs candidats à l'issue de la comparaison entre la dernière séquence explorée (s'_i) et la séquence de référence pour chaque décalage δ sont des motifs de la séquence de référence. Il sont donc bien des motifs candidats présents sur s'_1 ayant une occurrence (*i.e.*, motif e-similaires) sur s'_i. De surcroît, ils sont également des sous-motifs des motifs candidats ayant au moins une occurrence sur les séquences s'_j (avec $1 \leq j < i$) traitées auparavant pour au moins un décalage entre s'_1 et s'_j, comme illustré sur la figure 3.5. Chaque séquence agit comme un « filtre » des candidats issus de la séquence de référence. Il est possible, étant données les positions de début et de fin d'un motif m dans la séquence de référence, ainsi que les décalages (*cf.* Définition 3.10 – page 54) entre la séquence de référence et les

autres séquences de déterminer les positions des occurrences de m dans les autres séquences. En effet, soit δ_i le décalage entre s'_1 et s'_i, alors si $m = s'_1[a..b]$ a une occurrence m_i sur s'_i pour le décalage δ_i, cela signifie que $m_i = [a + \delta_i..b + \delta_i]$ (*cf.* Exemple 3.6 – page 56).

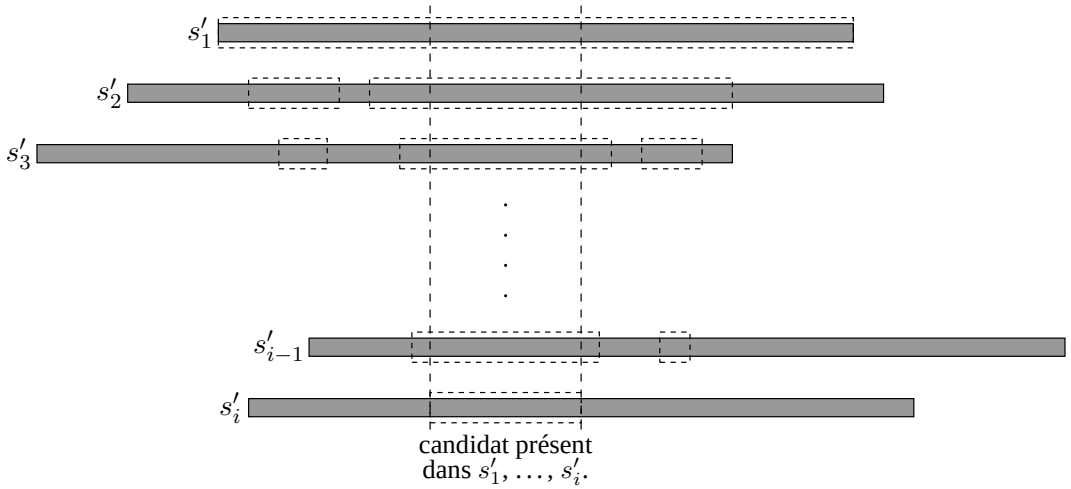


Figure 3.5 – Filtrage des candidats de s'_1 présents dans i séquences.

Les résultats du filtrage des candidats sont mémorisés à chaque instant dans un arbre enraciné, appelé arbre des solutions, tel que chaque nœud u à la profondeur $Niv(u)$ correspond à un sous-ensemble des candidats – représentés par les intervalles de leurs positions de début et fin sur s'_1 – après traitement des $Niv(u)$ premières séquences. La figure 3.6 donne une représentation de cet arbre, dont la racine λ correspond à la séquence de référence ($Niv(\lambda) = 1$). Le chemin de la racine à un nœud u est étiqueté par les décalages $\delta_2, \ldots, \delta_{Niv(u)}$ à appliquer respectivement aux séquences $s'_2, \ldots, s'_{Niv(u)}$ par rapport à la séquence de référence pour obtenir les candidats associés au nœud u. Les feuilles de l'arbre (à la profondeur t) contiennent alors l'ensemble des solutions. À chaque candidat d'un nœud u est associé un score « local » (*i.e.*, entre les séquences s'_1 et $s_{Niv(u)}$ pour l'alignement donné par l'arête menant à u) reflétant son degré de similarité avec le motif candidat (présent sur la séquence de référence), ainsi qu'un score « global » (traduisant la similarité globale de l'ensemble des occurrences du candidat sur les séquences $s'_1, \ldots, s'_{Niv(u)}$ alignées selon les décalages donnés par le chemin de λ à u). Chaque nœud u est alors étiqueté à la fois par la liste des candidats qui lui sont associés (notée $\mathcal{L}(u)$), et le meilleur score global des occurrences des candidats de $\mathcal{L}(u)$ sur les séquences $s'_1, \ldots, s'_{Niv(u)}$ (noté $score(u)$). Ces informations sont bien sûr très utiles lorsque les nœuds sont des feuilles, mais également dans le cas contraire. En effet, si u n'est pas une feuille, chaque candidat de $\mathcal{L}(u)$ est comparé, pour chaque décalage $\delta_{Niv(u)+1}$ existant entre s'_1 et $s'_{Niv(u)+1}$, à la séquence $s'_{Niv(u)+1}$. Seuls les motifs e-similaires résultant de ces comparaisons sont conservés dans l'arbre dans les nœuds issus de u.

Soient un nœud u et $[a; b] \in \mathcal{L}(u)$, un intervalle représentant le candidat $s'_1[a..b]$ au niveau $Niv(u)$. Soient $\delta_2, \ldots, \delta_{Niv(u)}$ la liste des décalages étiquetant le chemin de la racine au nœud u. Le score global associé aux occurrences de ce candidat dans les séquences $s'_1, \ldots, s'_{Niv(u)}$ est donc la similarité mesurée de la collection $(s'_1[a..b], s'_2[a + \delta_2..b + \delta_2], \ldots, s'_{Niv(u)}[a + \delta_{Niv(u)}..b + \delta_{Niv(u)}])$. Si ce score, noté ς, est le plus élevé parmi les scores globaux des collections obtenues à partir des intervalles de $\mathcal{L}(u)$ et des décalages $\delta_1, \ldots, \delta_{Niv(u)}$, alors $score(u) = \varsigma$.

Le choix retenu pour fixer le score global d'une collection de motifs e-similaires à un candidat est

de calculer le minimum des scores locaux entre le candidat et ses occurrences. Il s'agit d'une approche en étoile (*cf.* Figure 2.2). Ainsi, le calcul du score global d'un candidat représenté par $[a;b]$ s'effectue-t-il en deux temps. Tout d'abord, les scores locaux $f(s'_1[a..b], s'_i[a+\delta_i..b+\delta_i])$ $(2 \leq i \leq Niv(u))$ doivent être calculés. Ensuite, le score global est déterminé en considérant le minimum de tous les scores locaux. Plutôt que de calculer à chaque niveau l'ensemble des scores locaux $f(s'_1[a..b], s'_i[a+\delta_i..b+\delta_i])$ $(2 \leq i \leq Niv(u))$, la valeur

$$\min_{2 \leq i < Niv(u)} \left(f(s'_1[a..b], s'_i[a+\delta_i..b+\delta_i]) \right)$$

est approximée par $score(Pere(u))$. Cette approximation trouve sa justification dans les expérimentations menées. En effet, d'après les observations, pour $i \geq 4$, les listes de candidats associées à chaque nœud ne comportent généralement qu'un seul intervalle, dont les bornes varient extrêmement peu. L'intérêt de cette approximation réside non seulement dans le gain de temps réalisé lors du calcul d'une collection de motif, mais aussi dans le gain de temps réalisé lors du calcul de $score(u)$. De fait, soient $[a_1;b_1], \ldots, [a_\ell;b_\ell]$ tels que $\mathcal{L}(u) = \{[a_i;b_i] \,|\, 1 \leq i \leq \ell\}$. En notant ς_i $(1 \leq i \leq \ell)$ le score local $f(s'_1[a_1..b_1], s'_{Niv(u)}[a_i + \delta_{Niv(u)}..b_i + \delta_{Niv(u)}])$, le score de la collection représentée par l'intervalle $[a_i;b_i]$ est donc approximé par $\min(score(Pere(u)), \varsigma_i)$. Le score associé au nœud u est donc

$$\begin{aligned} score(u) &= \max_{1 \leq i \leq \ell} \left(\min\left(score(Pere(u)), \varsigma_i \right) \right) \\ &= \min\left(score(Pere(u)), \max_{1 \leq i \leq \ell} \left(\varsigma_i \right) \right). \end{aligned}$$

Cette approximation confère également à l'arbre une nouvelle propriété, qui est utilisée afin de l'élaguer plus efficacement :

Propriété 3.4. *Chaque sous-arbre de l'arbre des solutions d'un cycle de* STAMS *est tel que le score global de sa racine est supérieur ou égal au score global de ses nœuds.*

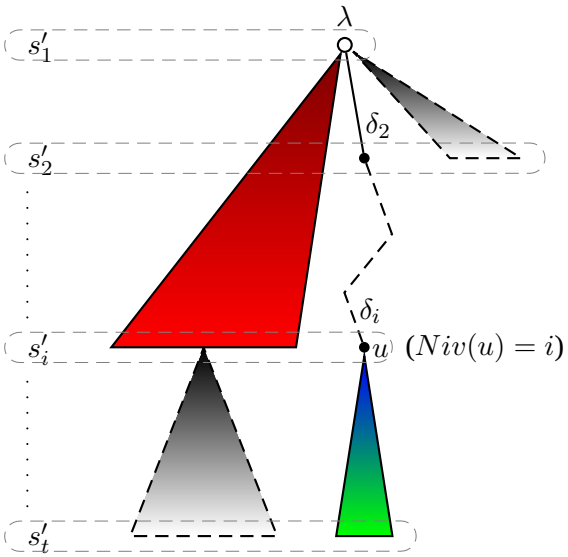


Figure 3.6 – Arbre de stockage des résultats d'un cycle de STAMS.

La création de l'arbre des solutions fournit toutes les solutions du cycle en cours d'exécution. À chaque nœud u de l'arbre, donc *a fortiori* à chaque feuille, est associé un score : $score(u)$. Il est alors possible de classer les nœuds (*i.e.*, les collections de motifs e-similaires à un candidat de $\mathcal{L}(u)$) par scores décroissants.

En outre, le nombre de candidats à l'issue de la construction demeure important. En effet, la taille de l'arbre complet est exponentielle. En effet, chaque nœud interne u possède au plus un fils par décalage entre la séquence de référence et la séquence $s'_{Niv(u)+1}$. En pratique, chaque nœud interne possède $\Theta(n)$ fils, avec n la longueur moyenne des séquences. Le nombre de nœuds dans l'arbre est donc en $\Theta(n^t)$, t étant le nombre de séquences de $\mathcal{S}$. Or parmi l'ensemble des solutions stockées dans les feuilles de l'arbre, l'objectif de l'algorithme est de ne renvoyer que les p meilleures solutions (données par leurs scores). Une procédure par séparation/évaluation (*Branch & Bound* [en anglais]) peut s'appliquer. Ainsi les deux stratégies classiques de construction de l'arbre ont été envisagées : création en profondeur d'abord (*depth-first search*) et création en largeur d'abord (*breadth-first search*). Un des objectifs du cahier des charges établi au chapitre précédent est d'obtenir une solution en temps polynomial. Aussi le paramètre p est-il utilisé dans la seconde stratégie (construction en largeur d'abord), pour une heuristique simple consistant à ne développer que les p nœuds les plus prometteurs *a priori* (ayant les meilleurs score), et cela pour chaque niveau de l'arbre. Les résultats obtenus lors des expérimentations sont sensiblement les mêmes que ceux obtenus par la construction de l'arbre sans cette heuristique (implémentation de la stratégie de création en profondeur d'abord). En contrepartie de la perte de fiabilité qu'elle engendre nécessairement, cette heuristique permet de garantir que la taille (exprimée en nombre de nœuds) de l'arbre est toujours dans $O(n\,t\,p)$, où t représente le nombre de séquences et n la longueur moyenne des séquences. C'est pourquoi cette solution heuristique a finalement été retenue (*cf.* Algorithme 3.2 – page suivante). Une fois l'arbre des solutions créé pour une permutation π donnée de $\mathcal{S}$, il suffit de parcourir les chemins menant de la racine à une feuille f pour obtenir les décalages à appliquer à s_{π_1} pour identifier les occurrences des motifs de $\mathcal{L}(f)$ sur les séquences s_{π_i}.

3.3.3 Détails sur le calcul des candidats

Il a été vu précédemment qu'un candidat est un motif de la séquence de référence identifiable par un intervalle représentant ses positions de début et de fin dans la séquence. Ainsi le motif débutant à la position a et terminant à la position b dans une séquence s, est représenté par $[a;b]$ plutôt que par la chaîne $s[a..b]$. Afin d'accélérer l'extraction des motifs similaires entre deux séquences, chaque séquence s_i subit un pré-traitement. Ainsi, pour chaque séquence s_i est créé un tableau $Stock_{s_i}$ de taille $|\Sigma|$, tel que $Stock_{s_i}[\alpha]$ est la liste ordonnée des positions des symboles $\alpha \in \Sigma$ dans la séquence s_i (*cf.* Algorithme 3.3 – page 64). De sorte que la liste des « correspondances » entre deux séquences s_i et s_j pour chaque alignement δ s'obtient en parcourant linéairement s_i ; en effet, pour chaque position k de s_i et pour chaque symbole $\alpha \in \Sigma$ tel que $s_i[k] =_{\mathcal{C}(\Sigma)} \alpha$, la liste mémorisée dans $Stock_{s_j}[\alpha]$ est la liste des positions l de s_j en « correspondance » avec $s_i[k]$ dans les différents alignements $\delta = l - k$ (*cf.* Algorithme 3.4 – page 65). Il est alors possible de construire une table $Cand_{s_i,s_j}$ de cardinalité $|s_i| + |s_j| - 1$ contenant tous les motifs e-similaires pour chaque alignement δ (compris entre $1 - |s_i|$ et $|s_j| - 1$) en parcourant pour chaque décalage la liste des positions en « correspondance » entre les deux séquences (*cf.* Algorithme 3.5 – page 66). Un pré-traitement similaire est utilisé dans le programme FASTA [91] (*cf.* Figure 3.7).

Les éléments des tableaux $Stock_{s_i}$, $Shared_{s_i,s_j}$ et $Cand_{s_i,s_j}$ sont des listes. Ils peuvent être assimi-

Nom : STABS-χ
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  π % *Permutation sur* $\mathcal{S}$ ($s'_i = s_{\pi_i}$). %
  p % *Nombre de résultats à renvoyer.* %
  $f : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score.* %
Sortie : *res* % *Ensemble de solutions pour ce cycle.* %
Variables : λ, u, v, w % *Nœuds.* %
  $\mathcal{A}$. % *Arbre des solutions.* %
  δ % *décalage entre deux séquences.* %
  Feuilles % *Liste ordonnée (par scores* $\searrow$*) des* p *meilleurs nœuds.* %
  Liste % *Liste temporaire de nœuds.* %
  Niv % *Profondeur de l'arbre en construction.* %
  chemin % *pile.* %
Début
  % *Création de l'arbre des solutions.* %
  Créer la racine λ de l'arbre $\mathcal{A}$.
  $\mathcal{L}(\lambda) \leftarrow \{[1..|s'_1|]\}$, $score(\lambda) \leftarrow +\infty$, $Liste \leftarrow \{\lambda\}$.
  Pour *Niv* de 2 à t **Faire**
    $Feuilles \leftarrow \emptyset$.
    Pour Chaque nœud $u \in Liste$ **Faire**
      Pour Chaque décalage δ entre s'_1 et s'_{Niv} **Faire**
        Créer un nouveau nœud v, ainsi qu'un arc de u vers v étiqueté par δ.
        Calculer $\mathcal{L}(v)$ à partir de $\mathcal{L}(u)$ pour l'alignement donné par δ.
        $score(v) \leftarrow \min(score(u), \max(\{f(s'_1[a..b], s'_{Niv}[a+\delta..b+\delta]) \mid [a..b] \in \mathcal{L}(v)\}))$.
        Ajouter v dans *Feuilles*.
      Fin Pour
    Fin Pour
    Trier *Feuilles* par scores $\searrow$ et ne conserver que les p premiers éléments.
    $Liste \leftarrow Feuilles$.
  Fin Pour

  % *Parcours de l'arbre des solutions.* %
  $res \leftarrow \emptyset$.
  Pour Chaque feuille v de l'arbre $\mathcal{A}$ **Faire**
    $chemin \leftarrow \$$. % *Pile vide.* %
    $u \leftarrow v$.
    Tant Que $u \neq \lambda$ **Faire**
      empiler l'étiquette de l'arc $(Pere(u), u)$ sur *chemin*.
      $u \leftarrow Pere(u)$.
    Fin Tant Que
    $res \leftarrow res \cup (\mathcal{L}(v), score(v), chemin, \pi)$.
  Fin Pour Chaque
  Retourner *res*
Fin

Algorithme 3.2 – Construction de l'arbre des solutions.

lés – dans ce contexte – à des ensembles[2] afin d'en donner une formalisation plus claire[3] :

$$\forall\, s_i, s_j \in \mathcal{S}, s_i \neq s_j, \quad \forall\, \delta, 1 - |s_i| \leq \delta \leq |s_j| - 1, \quad \forall\, \alpha \in \Sigma :$$

$$Stock_{s_i}[\alpha] = \left\{ k \;\middle|\; s_i[k] =_{\mathcal{C}(\Sigma)} \alpha, 1 \leq k \leq |s_i| \right\};$$

$$Shared_{s_i,s_j}[\delta] = \left\{ k \;\middle|\; \begin{array}{l} s_i[k] =_{\mathcal{C}(\Sigma)} s_j[k+\delta], \\ 1 \leq k \leq |s_i|, \\ 1 \leq k + \delta \leq |s_i| \end{array} \right\};$$

$$Cand_{s_i,s_j}[\delta] = \left\{ [a;b] \;\middle|\; \begin{array}{l} e\text{-similaire}(s_i[a..b], s_j[a+\delta..b+\delta]), \\ 1 \leq a < b \leq |s_i|, \\ 1 \leq a + \delta < b + \delta \leq |s_j| \end{array} \right\}.$$

```
1  Nom : Créer_Tableau_Stockage
2  Entrée : s % Séquence à traiter. %
3  Sortie : Stock_s % Tableau de Stockage de taille |Σ|. %
4  Variable : k % Position courante dans la séquence. %
5  Début
6    Pour Chaque α ∈ Σ Faire
7      Stock_s[α] ← ∅
8    Fin Pour
9    Pour k de 1 à |s| Faire
10     Ajouter k à la fin de la liste Stock_s[s[k]].
11     % k est nécessairement le plus grand élément de Stock_s[s[k]]. %
12   Fin Pour
13   Retourner Stock_s
14 Fin
```

Algorithme 3.3 – Pré-traitement des séquences.

Exemple 3.7 (Création d'une table de stockage des positions pour une séquence).
Soit l'alphabet $\Sigma = \{a, b, c, d\}$*. En considérant maintenant les séquences* $s_1 = ababc$ *et* $s_2 = badac$*, les tables de stockages obtenues sont alors :*

	Σ			
	a	b	c	d
$Stock_{s_1}$	1, 3	2, 4	5	
$Stock_{s_2}$	2, 4	1	5	3

En pratique, la notion de couverture de l'alphabet introduite au chapitre précédent (*cf.* Définition 2.6 – page 24) est utilisée pour permettre l'emploi d'un alphabet dégénéré (*cf.* Section 1.3.1 – page 18) dans les séquences. Aussi, il est nécessaire de partitionner l'alphabet (dit étendu) en deux sous-ensembles, l'alphabet initial et la dégénérescence de l'alphabet. L'alphabet étendu de l'ADN, par exemple, se compose d'un alphabet initial de quatre symboles $\{A, C, G, T\}$ et d'un alphabet dégénéré comprenant toutes

[2] Les éventuels doublons ne présentant pas d'intérêt, ils sont supprimés à la construction.
[3] La notion d'ordre n'apparaît pas dans cette formalisation.

les combinaisons (d'au moins deux symboles) de l'alphabet initial (*i.e.*,, 11 nouveaux symboles). Entre autres, le symbole R correspond à A ou G, le symbole V correspond à $A, C, G, \ldots$ Le test d'appartenance à chaque classe peut être facilement géré en utilisant un vecteur de bits de taille $|\Sigma|$ pour représenter l'alphabet initial, tel que chaque symbole soit associé à un bit unique ($A = 0001$, $C = 0010$, $G = 0100$, $T = 1000$). Ainsi un symbole dégénéré se construit en utilisant la disjonction bit à bit sur les vecteurs associés aux symboles de l'alphabet initial concernés ($R = 0101$, $V = 0111$, ...). Ainsi l'appartenance à une classe correspond à une conjonction bit à bit. Dans le cadre de l'algorithme 3.3 – page précédente, ce qui nous intéresse est de stocker non pas tous les symboles de l'alphabet étendu, mais seulement les symboles de l'alphabet initial. Ainsi, si le symbole à la position i de la séquence s est R, il faut ajouter i aux listes associées aux symboles A et G. C'est la raison pour laquelle il est nécessaire d'effectuer le test de la ligne 19 dans l'algorithme 3.4. Ces détails d'implémentation seront repris lors de l'étude de la complexité des algorithmes (*cf.* Section 3.3.5 – page 68).

```
1  Nom : Créer_Tableau_Correspondances
2  Entrées : s % séquence de référence. %
3    Stock_s' % Tableau de stockage associé à une séquence s'. %
4  Sortie : Shared_s,s' % Tableau des positions de s en correspondances dans s' %
5                       % pour chaque alignement donné par un décalage δ. %
6  Variables : δ % Décalages entre s et s'. %
7    k,l % indices de positions des séquences s et s'. %
8  Début
9    Pour δ de 1 − |s_i| à |s_j| − 1 Faire
10     Shared_s,s'[δ] ← ∅
11   Fin Pour
12   Pour k de 1 à |s| Faire
13     Pour Chaque α ∈ Σ Faire
14       Si s[k] =_C(Σ) α Alors
15         Pour Chaque l dans Stock_s'[α] Faire
16           % Les listes Stock_s'[α] sont ordonnées par construction. %
17           δ ← l − k
18           % En raison de la dégénérescence de l'alphabet, %
19           Si le dernier élément de Shared_s,s'[δ] n'est pas k Alors
20             % il faut prévenir l'ajout potentiel de doublons. %
21             Ajouter k à la fin de la liste Shared_s,s'[δ]
22           Fin Si
23           % Les listes Shared_s,s'[δ] sont donc également ordonnées. %
24         Fin Pour
25       Fin Si
26     Fin Pour
27   Fin Pour
28   Retourner Shared_s,s'
29 Fin
```

Algorithme 3.4 – Extraction des positions en « correspondances » entre deux séquences.

Exemple 3.8 (Création d'une table de positions en « correspondances » entre deux séquences).

En reprenant l'exemple de la page ci-contre, et en considérant que l'alphabet est muni de la couverture $\mathcal{C}(\Sigma) = \big\{\{a\}, \{b\}, \{c, d\}\big\}$*, les tables des positions partagées entre* $s_1 = ababc$ *et* $s_2 = badac$ *pour*

chaque décalages entre ces deux séquences sont alors :

	δ								
	-4	-3	-2	-1	0	1	2	3	4
$Shared_{s_1,s_2}$		*4*	*5*	*2, 3*	*5*	*1, 3*		*1*	
$Shared_{s_2,s_1}$		*4*		*2, 4*	*5*	*1, 2*	*3*	*1*	

```
Nom : Créer_Tableau_Candidats
Entrée : Shared_{s,s'} % Tableau des positions de s en correspondances dans s' %
                       % pour chaque alignement donné par le décalage δ. %
Sortie : Cand_{s,s'} % Tableau des candidats de s ayant une occurrence dans s' %
                     % pour chaque alignement donné par le décalage δ. %
Variables : δ % Décalages entre s et s'. %
  k,l % indices de positions des séquences s et s'. %
Début
  Pour δ de 1−|s| à |s'|−1 Faire
    deb ← 0 % Pour chaque nouveau décalage, il n'y a pas encore de motif. %
    Pour Chaque k de la liste Shared_{s,s'}[δ] Faire
      % La liste est ordonnée par construction. %
      Si deb = 0 Alors % Pas encore de motif en cours d'extraction. %
        deb ← k % Un nouveau motif débute à la position courante dans s, %
        fin ← deb % et se termine au plus tôt à cette même position. %
      Sinon % Motif en cours d'extraction. %
        Si (k − fin − 1 ≤ e) Alors % Non correspondances consécutives. %
          fin ← k % Le motif s'achève au plus tôt à la position courante. %
        Sinon % Le motif s[deb..fin] est un nouveau candidat. %
          Ajouter [deb; fin] à la fin de la liste Cand_{s,s'}[δ].
          deb ← k % Un nouveau motif débute à la position courante dans s, %
          fin ← deb % et se termine au plus tôt à cette même position . %
        Fin Si
      Fin Si
    Fin Pour
    Si deb ≠ 0 Alors % Il reste un motif en construction à ajouter. %
      Ajouter [deb; fin] à la fin de la liste Cand_{s,s'}[δ].
    Fin Si
    % La liste Cand_{s,s'}[δ] est également ordonnée. %
  Fin Pour
  Retourner Cand_{s,s'}.
Fin
```

Algorithme 3.5 – Extraction des motifs e-similaires entre deux séquences.

Afin d'optimiser le traitement des candidats, il est possible de modifier légèrement l'algorithme précédent afin de ne conserver que les candidats de longueur au moins 2 (*i.e.*, en ajoutant les candidats uniquement si $fin > deb$). Cette restriction permet de ne pas encombrer les tables $Cand$ avec des mots d'un seule lettre, qui ne recèlent aucun intérêt biologique dans le cadre de cette étude.

Exemple 3.9 (Création d'une table de motifs e-similaires entre deux séquences).
En reprenant le dernier exemple (3.8 – page précédente), l'extraction des motifs 1*-similaires présents entre* $s_1 = ababc$ *et* $s_2 = badac$ *donne les tableaux* $Cand_{s_1,s_2}$ *et* $Cand_{s_2,s_1}$*. Les motifs unitaires sont*

représentés en gris afin de faire ressortir les motifs qui sont conservés avec la version modifiée de ce dernier algorithme.

	δ								
	-4	-3	-2	-1	0	1	2	3	4
$Cand_{s_1,s_2}$		[4 ; 4]	[5 ; 5]	[2 ; 3]	[5 ; 5]	[1 ; 3]		[1 ; 1]	
$Cand_{s_2,s_1}$		[4 ; 4]		[2 ; 4]	[5 ; 5]	[1 ; 2]	[3 ; 3]	[1 ; 1]	

Comme l'illustre la figure 3.7, les tables $Shared$ et $Cand$ sont en étroite relation avec la méthode dite « des diagonales » décrite par WILBUR & LIPMAN [137], et utilisée entre autres dans l'algorithme FASTA [91, 106]. Les points correspondent au « correspondances » entre les deux séquences $s_1 = ababc$ et $s_2 = badac$ des exemples précédents. Le point gris n'apparaîtrait pas dans la méthode des diagonales, car il s'agit d'une « correspondance » entre deux symboles différents. Néanmoins, la méthode décrite en 1983 s'applique pleinement dans notre contexte. Les zones mises en relief sur la figure sont la représentation sur la matrice des deux motifs 1-similaires donnés par la table $Cand_{s_1,s_2}$.

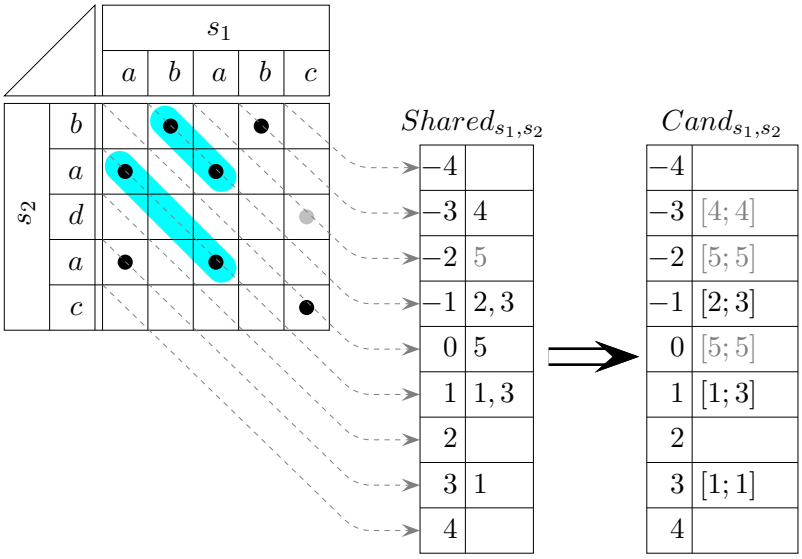


Figure 3.7 – De la méthode des diagonales aux tables $Shared$ et $Cand$.

Dans l'algorithme de construction de l'arbre des solutions (*cf.* Algorithme 3.2 – page 63), la ligne 23 mentionne le calcul de l'ensemble $\mathcal{L}(v)$ à partir de l'ensemble $\mathcal{L}(u)$, où u et v sont deux nœuds de l'arbre avec $u = Pere(v)$, et l'arête (u, v) étant étiquetée par le décalage δ. Ce calcul est effectué en comparant les listes $\mathcal{L}(u)$ (liste des motifs de s'_1 présents sur les séquences s'_i, $1 \leq i \leq Niv(u)$) et $Cand_{s'_1,s'_{Niv(v)}}[\delta]$ (la liste des candidats de s'_1 présents dans la séquence $s'_{Niv(v)}$ en cours de traitement pour un décalage δ entre ces deux séquences). Les listes étant ordonnées par construction, il s'agit de les parcourir linéairement en comparant les bornes des intervalles en cours dans les deux listes afin d'extraire les candidats présents dans les séquences s'_i, $0 \leq i \leq Niv(v)$.

3.3.4 Mise à jour des solutions et arrêt de l'algorithme

L'algorithme STAMS-χ construit, étant donnés un ensemble de séquence $\mathcal{S}$, une permutation π, un entier p et une fonction de score f, un ensemble d'au plus p solutions (si on ne considère qu'une solution

– la meilleure – par feuille construite dans l'arbre) au problème de l'extraction de motifs communs à l'ensemble de séquences $\mathcal{S}$. Elles sont valuées selon la fonction f. Toutefois, afin d'améliorer la pertinence des résultats, il est nécessaire d'effectuer plusieurs cycles (*i.e.*, il est nécessaire de relancer l'algorithme STAMS-χ plusieurs fois). En effet, la qualité des résultats dépend principalement du choix de la permutation. Aussi est-il nécessaire de construire un arbre de solutions pour plusieurs permutations, et d'en extraire à chaque fois les p meilleures solutions, lesquelles sont à comparer (selon leur score) avec les solutions déjà calculées. Les informations conservées pour chacune d'entre elles sont la permutation utilisée, la liste des décalages permettant de procéder à l'alignement des séquences selon la permutation, la liste des motifs de la séquence de référence correspondant à cette solution (s'il y a plusieurs motifs dans la liste, celui engendrant le meilleur score est considéré comme le motif consensuel de la solution, les autres sont justes conservés à titre de complément d'information), ainsi que le score global associé à cette liste (*i.e.*, la mesure de la similarité de la collection de motifs alignés avec le modèle consensuel). La mise à jour des solutions consiste donc à ordonner cycle après cycle la liste des solutions Sol et à ne conserver que les p meilleures. Chaque quadruplet renvoyé par STAMS-χ contient les informations suffisantes pour reconstruire une solution au problème EM. Il est par conséquent possible d'assimiler chacun d'eux à une solution.

S'il n'est pas envisageable de tester toutes les permutations, il n'est pas non plus raisonnable de choisir les permutations totalement au hasard. Le choix de la séquence de référence est évidemment crucial, et toutes les séquences doivent au moins une fois être considérées comme telles. Les tests préliminaires à l'élaboration de l'algorithme ont mis en lumière que la contribution (nouveauté des motifs extraits, améliorations du score) apportée par l'ordre de traitement des séquences (hormis le choix de la séquence de référence) décroît très rapidement. L'heuristique retenue finalement est donc de faire varier de manière cyclique la séquence de référence, et de traiter les autres séquences dans un ordre aléatoire. Lorsque l'ensemble des résultats n'est plus modifié après plusieurs permutations, l'algorithme s'arrête. Ce nombre de permutations a été fixé expérimentalement à t^2, où t représente le nombre de séquences.

3.3.5 Complexité en moyenne

L'algorithme 3.1 – page 59 – permet d'avoir une vue d'ensemble des étapes du programme. Cependant, avant d'entamer l'étude de sa complexité, il est préférable de le détailler, en y intégrant les différents traitements détaillés ci-avant (*cf.* Algorithme 3.6 – page ci-contre).

Afin de simplifier les calculs, nous supposerons que toutes les séquences sont de même longueur n, et que la fonction de score f appliquée à deux motifs de même longueur k requiert un temps $O(k)$ (cette considération est valable pour toutes les fonctions présentées dans ce chapitre). Nous dénotons par t le nombre de séquences traitées, et par p le nombre de maximum de résultats à renvoyer. Enfin, pour tout $\alpha \in \Sigma$ (avec Σ l'alphabet initial, *i.e.*, l'ensemble des symboles présents dans un singleton de la couverture de l'alphabet), nous noterons p_α la probabilité d'avoir le symbole α (ou un symbole d'une même classe que α selon la couverture) dans les séquences de l'ensemble. Ainsi, le nombre moyen d'occurrences des symboles de l'alphabet dans une séquence est $p_\alpha\, n$.

Pré-traitement des séquences (ligne 17 à 28)

Pour chaque séquence s_ℓ de l'ensemble $\mathcal{S}$, la construction du tableau $Stock_{s_\ell}$ (*cf.* Algorithme 3.3 – page 64) requiert un temps et un espace en $O(|\Sigma|\, n)$. En effet, la séquence est parcourue une fois, et chacun des symboles de la séquence est comparé à l'ensemble des symboles de l'alphabet initial

Nom : STAMS
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  $\Sigma, \mathcal{C}(\Sigma)$ % *Alphabet et couverture de l'alphabet utilisé.* %
  p % *Nombre de résultats à renvoyer.* %
  $f : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score.* %
Sortie : *Sol* % *Ensemble des résultats.* %
Variables : *OldSol* % *Ensemble des résultats avant le dernier cycle.* %
  ℓ % *Numéro de la séquence de référence* %
  k % *indice de la séquence en cours de traitement* %
  π % *Permutation sur* $\mathcal{S}$ ($s'_i = s_{\pi_i}$). %
  nb_cycles % *Compteur de cycles.* %
  TStock % *Vecteur de taille* t *pour le pré-traitement des séquences.* %
  TShared % *Vecteur de taille* $2n-1$ *pour les décalages entre 2 séquences.* %
  TCand % *Matrice Carrée de taille* t *pour le traitement de TShared.* %
Début
  % *Pré-traitement des séquences.* %
  Pour ℓ de 1 à t **Faire**
    $TStock[\ell] \leftarrow$ Créer_Tableau_Stockage(s_ℓ). % *Algorithme* 3.3, *p.*64 %
  Fin Pour

  Pour ℓ de 1 à t **Faire**
    Pour k de 1 à t **Faire**
      Si $\ell \neq k$ **Alors**
        $TShared \leftarrow$ Créer_Tableau_Correspondances($s_\ell, TStock[k]$). % *Algorithme* 3.4, *p.*65 %
        $TCand[\ell, k] \leftarrow$ Créer_Tableau_Candidats($TShared$). % *Algorithme* 3.5, *p.*66 %
      Fin Si
    Fin Pour
  Fin Pour

  % *Processus d'extraction.* %
  $Sol \leftarrow \emptyset$, $OldSol \leftarrow \emptyset$.
  $\ell \leftarrow 0$, *nb_cycles* $\leftarrow 0$.
  Répéter
    % *Choix de la séquence de référence, puis de la permutation.* %
    $\ell \leftarrow (\ell \mod t) + 1$.
    Choisir une permutation π de $\mathcal{S}$ telle que $s'_1 = s_\ell$.
    % *Calcul de la solution pour cette permutation.* %
    $Sol \leftarrow Sol \cup$ STAMS-$\chi(\mathcal{S}, \pi, p, f)$. % *Algorithme* 3.2, *p.*63 %
    Trier *Sol* par scores $\searrow$ et ne conserver que les p meilleurs résultats.
    % *Mise à jour du nombre de cycles exempts de modification.* %
    Si $Sol \neq OldSol$ **Alors**
      nb_cycles $\leftarrow 0$.
    Sinon
      nb_cycles $\leftarrow$ *nb_cycles* $+ 1$.
    Fin Si
  Tant Que (*nb_cycles* $< t^2$)
  Afficher et **Retourner** *Sol*.
Fin

Algorithme 3.6 – Algorithme d'extraction de motifs STAMS.

(*cf.* discussion de la section 3.3.3 – page 62). Ainsi, l'initialisation de la variable $TStock$ (lignes 17 à 19) requiert un temps et un espace en

$$O(|\Sigma|\, t\, n). \tag{3.2}$$

Étant données deux séquences s_ℓ et s_k, la création du tableau $Shared_{s_\ell,s_k}$ (*cf.* Algorithme 3.4 – page 65) requiert un temps et un espace en $O\left(n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right)$. Outre l'initialisation des listes vides en $\Theta(n)$, pour chaque symbole (alphabet étendu) de s_ℓ et pour chaque symbole α de l'alphabet initial dans une même classe, un calcul en temps constant avec la chaque élément de la liste $Stock_{s_k}$ est nécessaire. En moyenne sur l'ensemble des séquences, les listes $Stock_{s_k}[\alpha]$ sont de longueur $O(p_\alpha\, n)$. Donc pour un symbole $s_\ell[i] =_{\mathcal{C}(\Sigma)} \alpha$, il faut un temps et un espace en $O\left(\sum_{\alpha=_{\mathcal{C}(\Sigma)} s_\ell[i]} p_\alpha\, n\right)$, soit pour la séquence entière $O\left(\sum_{i=1}^{n}\sum_{\alpha=_{\mathcal{C}(\Sigma)} s_\ell[i]} p_\alpha\, n\right) = O\left(\sum_{\beta\in\Sigma} p_\beta\, n \sum_{\alpha=_{\mathcal{C}(\Sigma)}\beta} p_\alpha\, n\right) = O\left(n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right)$. La construction de l'ensemble de la table $TShared$ se fait donc en $O\left(t\,(t-1)\, n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right)$. La complexité temporelle et spatiale de la création de la table $TShared$ est donc en

$$O\left(t^2\, n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right). \tag{3.3}$$

La construction de la liste de candidats entre deux séquences s_ℓ et s_k pour un décalage δ donné (*i.e.*, $Cand_{s_\ell,s_k}[\delta]$, *cf.* Algorithme 3.5 – page 66) à partir de la liste des positions en « correspondances » entre ces deux séquences pour ce décalage (*i.e.*, $Shared_{s_\ell,s_k}[\delta]$) se fait en temps linéaire sur la longueur de la liste des positions (*i.e.*, $|Shared_{s_\ell,s_k}[\delta]|$, *cf.* Équation 3.3). Concernant la complexité spatiale, comme la longueur de la liste $Shared_{s_\ell,s_k}[\delta]$ est au pire en $O(n)$, la longueur de la liste $Cand_{s\ell,s_k}[\delta]$ est en $O(n/e)$. De surcroît, la longueur de la liste $Cand_{s_\ell,s_k}[\delta]$ est également en $O\left(|Shared_{s_\ell,s_k}[\delta]|\right)$. Comme $|Shared_{s_\ell,s_k}|$ est en $O(n^2 \sum p_\alpha^2)$ et qu'en moyenne $|Shared_{s_\ell,s_k}[\delta]|$ est en $O(n \sum p_\alpha^2)$, il résulte que $|Cand_{s_\ell,s_k}[\delta]| = O\left(\min(n/e, |Shared_{s_\ell,s_k}[\delta]|)\right)$. Donc pour tous les δ, la construction de $Cand_{s_\ell,s_k}$ requiert un espace en $O\left(\min(n^2/e, n^2 \sum_{\alpha\in\Sigma} p_\alpha^2)\right)$. Par conséquent, la création de la matrice $TCand$ (lignes 21 à 28) requiert un temps en

$$O\left(t^2\, n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right), \tag{3.4}$$

et un espace en

$$O\left(t^2\, n^2\, \min\left(1/e, \sum_{\alpha\in\Sigma} p_\alpha^2\right)\right). \tag{3.5}$$

Initialisation des Variables (lignes 31 à 32)

Ces quatre opérations se font en temps et en espace constant $\Theta(1)$.

Processus d'extraction (lignes 33 à 46)

L'initialisation de la permutation se base sur l'algorithme de permutation fourni par la *Standard Template Library (*STL*)*, extrait de [77]. Cet algorithme est linéaire en temps sur les données à permuter, et constant en espace. Le positionnement de la séquence de référence est une simple permutation entre s_{π_1} et s_{π_ℓ}. Ainsi la construction de π (et de π^{-1} qui permet notamment de retrouver la séquence de référence en temps constant) se fait en $O(t)$.

La construction de l'arbre des solutions (*cf.* Algorithme 3.2 – page 63) requiert un temps en

$$O(p\,t\,n^2) \tag{3.6}$$

et un espace en

$$O(p\,t\,n^2/e). \tag{3.7}$$

En effet, la création d'un nœud v issu d'un nœud u dans l'arbre nécessite d'effectuer un parcours linéaire des listes $\mathcal{L}(u)$ et $Cand_{s'_1,s'_{Niv(v)}}[\delta]$, où δ est l'étiquette de l'arête reliant u à v. La liste résultante de ce parcours à une longueur en $O(n/e)$. Plus généralement, pour tout nœud de l'arbre, la liste de candidats qui lui sont associés a une longueur en $O(n/e)$, ce qui est également le cas de $Cand_{s'_1,s'_{Niv(v)}}[\delta]$. Il en résulte donc que la création de $\mathcal{L}(v)$ nécessite un temps et un espace en $O(n/e)$. La création du nœud v demande également que soit calculé un score local pour chaque candidat de $\mathcal{L}(v)$ et que le score maximum soit comparé à $score(u)$ (*cf.* Propriété 3.4 – page 61). Il a été posé comme pré-requis au début de cette sous-section que le calcul du score se faisait au pire en $O(k)$ pour des paires de motifs de longueur k. Comme la liste $\mathcal{L}(v)$ représente nécessairement des motifs disjoints sur la séquence de référence, la somme des temps requis par le calcul des scores des couples formés par les occurrences de chaque candidat est en $O(n)$. Ainsi le calcul du score associé au nœud v se fait en $O(n)$. Il en résulte que la création du nœud v s'effectue en $O(n)$ et que cela nécessite un espace en $O(n/e)$. Ainsi la création des fils d'un nœud dans l'arbre (lignes 21 à 26 dans l'algorithme 3.2 – page 63) requiert un temps en $O(n^2)$ et un espace en $O(n^2/e)$. Comme le nombre de nœuds traités à chaque niveau de l'arbre est au plus égal à p, le traitement de tous les nœuds d'un niveau requiert un temps en $O(p\,n^2)$ et un espace en $O(p\,n^2/e)$. L'élagage éventuel de l'arbre (ligne 28) est effectué de deux façons selon la longueur de la liste $Feuilles$, notée ζ ci-après. En effet, si ζ est supérieur à 2^p, l'algorithme utilisé est un tri par extraction, qui s'arrête lorsque les p premiers éléments sont traités, les autres sont ensuite supprimés. Ainsi le tri s'opère en $O(p\,\zeta)$, et la suppression en $O(\zeta - p)$. Dans le cas contraire, l'algorithme utilisé est un tri par tas, qui s'effectue donc en $O(\zeta\,\log\zeta)$, et les $\zeta - p$ derniers éléments sont supprimés (en $O(\zeta - p)$). La copie de la liste $Feuilles$ dans la liste $Liste$ (ligne 29) s'effectue en $O(p)$. Ainsi, la complexité temporelle de l'élagage est $O(\zeta\,\min(\log\zeta, p))$. Or un nœud engendre au plus $2\,n - 1$ nœuds, donc le nombre ζ de nœuds développés à un niveau est en $O(p\,n)$. Un tour de boucle (lignes 18 à 30) s'effectue donc en $O(p\,t\,n^2 + p\,n\,\min(\log(n\,p), p))$, et requiert un espace en $O(p\,t\,n^2/e + p\,n\,\min(\log(n\,p), p))$.

La mise à jour des résultats (lignes 34 à 42) nécessite le parcours du chemin de la racine à chaque feuille (lignes 37 à 40), de complexité spatiale et temporelle $O(p\,t)$, la copie (ligne 41) de la permutation en $O(t)$, la copie des listes de candidats en $O(p\,n/e)$ et la sauvegarde du score en $O(1)$. Cette étape a donc une complexité

$$O(p\,(t + n/e)). \tag{3.8}$$

Le tri et la sélection de l'ensemble Sol (ligne 39 de l'algorithme 3.6 – page 69) s'effectuent en $O(p\log p)$, et la liste Sol occupe un espace en $O(p\,(t + n/e))$.

Ainsi, un tour de boucle s'effectue en

$$O\left(\underbrace{t}_{l.\ 36} + \underbrace{p\,t\,n^2}_{\substack{l.\ 38,\\ (3.6)\,et\,(3.8)}} + \underbrace{p\,\log p}_{l.\ 39}\right), \tag{3.9}$$

pour un espace en

$$O\left(\underbrace{t}_{l.\ 36} + \underbrace{p\,t\,n^2/e}_{l.\ 38,(3.7)} + \underbrace{p\,(t+n/e)}_{l.\ 39}\right). \tag{3.10}$$

Soit $\mathcal{N}$ le nombre d'itérations de l'algorithme, alors la complexité temporelle de l'exécution de l'algorithme entier est

$$O\left(\underbrace{|\Sigma|\,t\,n}_{(3.2)} + \underbrace{t^2\,n^2\sum_{\alpha\in\Sigma} p_\alpha^2}_{(3.4)} + \underbrace{\mathcal{N}\,p\,t\,n^2}_{(3.9)}\right), \tag{3.11}$$

tandis que la complexité spatiale est

$$O\left(\underbrace{|\Sigma|\,t\,n}_{(3.2)} + \underbrace{t^2\,n^2\,\min\left(1/e, \sum_{\alpha\in\Sigma} p_\alpha^2\right)}_{(3.5)} + \underbrace{p\,t\,n^2/e}_{(3.10)}\right). \tag{3.12}$$

Selon les résultats expérimentaux réalisés sur un grand nombre d'exemples, la valeur $\mathcal{N}$ est de l'ordre de $N \times O(t)$ où N représente le nombre de cycles entraînant une mise à jour des résultats. En pratique, N est dans $O(p\,t)$; de plus, il est tout à fait raisonnable de considérer que $p = O(t)$. Il en résulte que la complexité temporelle observée de STABS est $O(p^2\,t^3\,n^2)$, la complexité spatiale étant $O(t^2\,n^2/e)$.

3.3.6 Améliorations

À partir des tests préliminaires, afin d'améliorer la qualité des résultats, il a été envisagé de relancer l'algorithme principal (*i.e.,* STABS) en choisissant comme séquence de référence la dernière séquence ayant permis une mise à jour des résultats, et de faire varier cycliquement la seconde séquence, en considérant comme période d'arrêt $(t-1)$ cycles sans mise à jour. Cela a abouti à alourdir la complexité pour une très maigre amélioration des résultats.

Un inconvénient de l'algorithme 3.2 – page 63, provient du fait que lorsqu'une feuille contient une liste de motifs résultats $\mathcal{L}$, elle identifie une seule occurrence de chaque motif de $\mathcal{L}$ sur chacune des séquences. Or si au moins une des séquences, traitée au niveau i dans l'arbre, contient ne serait-ce que deux occurrences de chaque motif pour deux décalages donnés δ_i' et δ_i'' par rapport à la séquence de référence, alors la liste $\mathcal{L}$ est à l'origine du développement de deux branches différentes (*cf.* Figure 3.8) dans l'arbre. Ainsi pour les deux décalages δ_i' et δ_i'', la même liste est représentée, et génère le même sous-arbre ϑ.

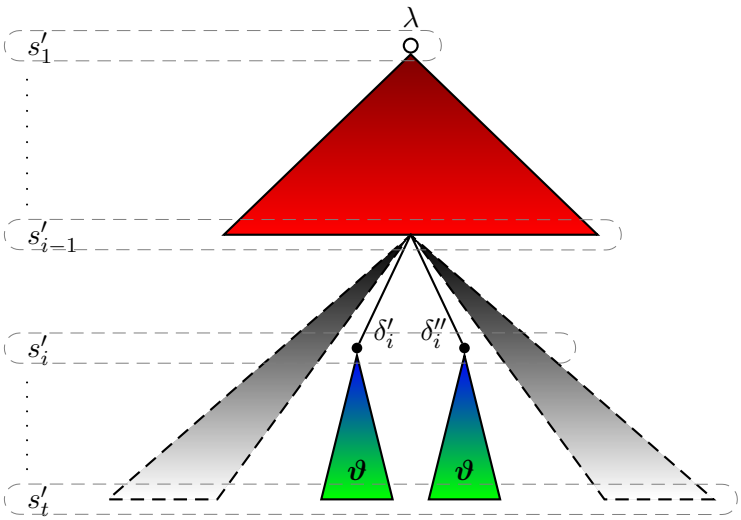


Figure 3.8 – Sous-arbres identiques (ϑ) dans l'arbre des solutions construit par STAMS-χ.

Plus généralement, si la séquence traitée au niveau i contient x occurrences du meilleur motif m à renvoyer, alors x branches seront engendrées au niveau $i+1$. Si plusieurs séquences sont dans ce cas, le nombre de feuilles dans l'arbre non tronqué correspondant au motif m devient vite exponentiel. Or l'objectif de l'algorithme est de renvoyer p motifs différents. C'est pourquoi, un traitement supplémentaire est nécessaire afin d'éviter ce phénomène. Ce traitement – linéaire en temps – consiste à diminuer le score des nœuds étiquetant des motifs déjà représentés dans une autre liste d'un nœud au même niveau dans l'arbre et de score plus important.

3.3.7 Développement

Cet algorithme a été implémenté en `c++`, en respectant la norme ANSI. Il est distribué sous licence GPL (Licence Publique Générale) et accessible sous formes d'archives (`.deb`, `.rpm` et `.tar.gz`) à l'adresse suivantes : `http://www.sciences.univ-nantes.fr/lina/bioserv/stars/`. Son développement est accompagné d'une documentation complète sous différents formats (`man`, `info`, `ps`, `pdf` et `html`).

Une interface *Web* a également été mise en place permettant son utilisation distante à l'adresse `http://www.sciences.univ-nantes.fr/lina/bioserv/WebStars/`. Cet outil est également distribué sous licence GPL et disponible sous formes d'archives (`.deb`, `.rpm` et `.tar.gz`). Cette interface propose – en plus de la sortie standard du programme en la ligne de commande – un affichage des « *Sequence Logo* » de chaque motif extrait.

3.4 Tests & Résultats

Les tests présentés ci-après ont été réalisés sur un serveur de calcul quadri-processeur *Pentium*® III, cadencés chacun à $700MHz$, et disposant de $4Go$ de mémoire vive.

L'algorithme a été testé sur des séquences générées aléatoirement dans lesquelles des motifs ont été insérés, puis sur des séquences biologiques, contenant des motifs validés expérimentalement.

3.4.1 Séquences générées aléatoirement

Les tests sur les séquences aléatoires ont été menés afin d'estimer la performance de l'algorithme, et par conséquent, pour valider le choix des heuristiques utilisées (construction et troncature des arbres et critère d'arrêt de l'algorithme). Pour cela, l'algorithme est considéré comme un test au sens statistique du terme (*cf.* Table 3.3) ; en effet, STAMS va scinder l'ensemble des mots de Σ^* en deux parties, l'une contenant (selon l'algorithme) les motifs significatifs présents dans toutes les séquences, l'autre partie contenant tous les autres motifs. Des séquences sont générées aléatoirement, ainsi que plusieurs motifs. Les motifs sont altérés (*i.e.*, des substitutions sont aléatoirement introduites) puis sont insérés au hasard dans les séquences. Un motif est vu comme une suite de positions, et chaque position renvoyée par STAMS est comparée aux positions des motifs insérés dans les séquences. Lorsqu'une position d'un motif renvoyé par l'algorithme correspond à une position d'un motif inséré, cette position est dite « vraie positive » (VP) ; si elle ne correspond pas à une position d'un motif inséré, elle est alors dite « fausse positive » (FP). Au contraire, une position d'un motif inséré qui n'est pas renvoyée par STAMS est dite « fausse négative » (FN), et enfin, une position qui n'est pas renvoyée par l'algorithme et qui n'appartient à aucun motif est dite « vraie négative » (VN).

	Critère validé	Critère non validé
Test positif	Vrais Positifs (VP)	Faux Positifs (FP)
Test négatif	Faux Négatifs (FN)	Vrais Négatifs (VN)

Table 3.3 – Classification des résultats dans les tests statistiques.

À partir de cette classification des résultats, il est possible de calculer plusieurs indicateurs classiques (sensibilité, spécificité, valeurs prédictives positives et négatives) permettant d'évaluer la qualité d'un test statistique [128] (*cf.* Table 3.4).

$$\text{Sensibilité} = \frac{\text{VP}}{\text{VP} + \text{FN}} \qquad \text{VPP} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

$$\text{Spécificité} = \frac{\text{VN}}{\text{VN} + \text{FP}} \qquad \text{VPN} = \frac{\text{VN}}{\text{VN} + \text{FN}}$$

Table 3.4 – Indicateurs usuels de la performance des tests statistiques.

Le nombre de vrais négatifs étant quasi exclusivement dépendant de la longueur des séquences, les indicateurs retenus sont donc la valeur prédictive positive et la sensibilité. Pour chaque exécution effectuée, ces deux indicateurs ont donc été calculés, ainsi que la moyenne sur l'ensemble des exécutions.

Les séquences ont été générées par une source sans mémoire, sur un alphabet à 4 lettres équiprobables. Chaque ensemble contient au moins 10 séquences de longueur au moins 100. Pour chaque ensemble, trois ensembles de motifs 5-similaires de tailles minimales fixées (25, 15 et 10) et donnant chacun un score différent fixé ont été insérés dans les séquences à des positions choisies aléatoirement. Au total, 1 000 ensembles de séquences ont été engendrés.

L'algorithme a été exécuté sur chaque ensemble avec en paramètres $p = 1$ (recherche du meilleur motif), $Simil = 5$-similarité, $f^{=}(k) = 2\,k$ et $f^{\neq}(k) = -k$ (fonction *Block-Based* Q-indépendante).

Pour chaque jeu de test, la VPP et la sensibilité ont été calculés. Les résultats obtenus sont illustrés par la figure 3.9. L'axe des abscisses correspond aux valeurs possibles de la valeur de la VPP (respectivement de la sensibilité) par tranches de 5% et l'axe des ordonnées au nombre de jeux de tests ayant engendré une VPP (respectivement une sensibilité) comprise dans un des intervalles en abscisse. Par exemple, 203 des 1 000 jeux d'essais ont engendrés une sensibilité comprise entre 95% inclus et 100% exclus. Il est possible de noter sur la figure que la quasi totalité des exécutions a abouti à une VPP supérieure à 90%, et que pour une très grande majorité, la sensibilité est également supérieure à 90%.

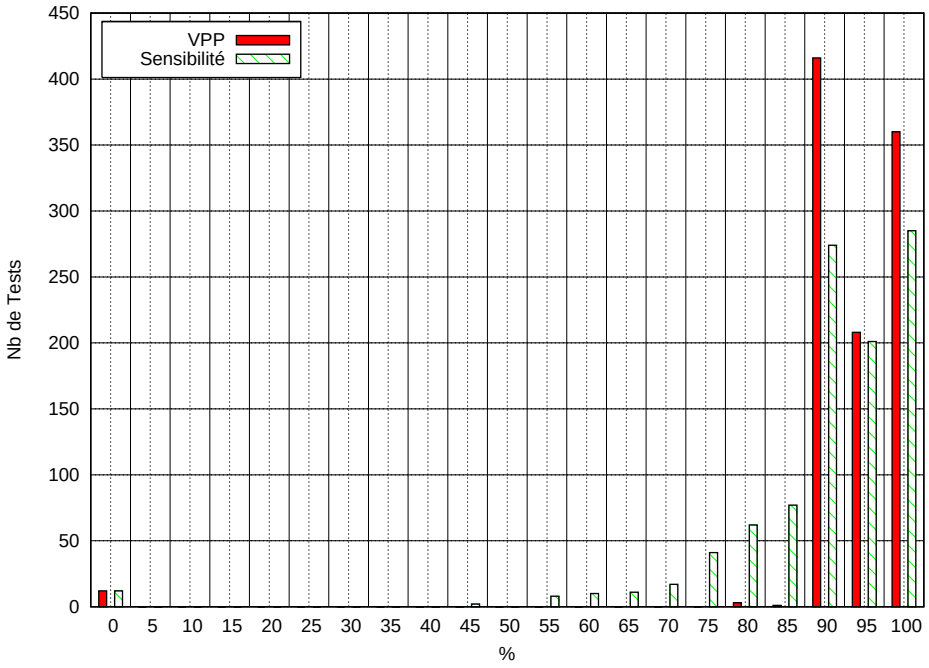


Figure 3.9 – Sensibilité et VPP de STAMS mesurées sur 1 000 jeux de séquences générés aléatoirement.

La valeur prédictive positive (VPP) moyenne sur l'ensemble des exécutions est de 0,95, tandis que la sensibilité moyenne obtenue est de 0,91. Ceci signifie qu'en moyenne, 95% des positions des motifs extraits par STAMS correspondent aux positions des motifs insérés avec le meilleur score et que 91% des positions des motifs insérés ont été découverts par l'algorithme. Une autre vision de ces résultats est que les motifs extraits sont très majoritairement ceux qui ont été insérés avec le meilleur score (ils recouvrent en moyenne 91% des positions des motifs insérés sur chaque séquence), mais ils sont légèrement décalés à gauche ou à droite. Un biais possible dans la procédure d'évaluation est que les motifs insérés dans les séquences peuvent engendrer des motifs 5-similaires plus grands (avec un score plus élevé) dûs à leurs contextes droits et gauches, notamment avec un alphabet à quatre symboles. Si les calculs de la VPP et de la sensibilité sont effectués en comparant les motifs extraits à l'un des trois motifs insérés, alors la VPP moyenne est de 0,96, quant à et la sensibilité moyenne, elle passe à 0,92. Soit une très faible augmentation. Ceci se traduit par le fait que dans quelques cas, l'algorithme a détecté un motif qui n'a pas été inséré par le générateur de jeux d'essais (les trois jeux d'essais qui ont une VPP et une sensibilité à 0). Il n'en résulte pas moins que ces motifs existent ; ce phénomène est principalement dû à la petitesse de la taille de l'alphabet.

3.4.2 Séquences biologiques

Les tests sur des séquences biologiques ont été réalisés sur 6 ensembles de séquences d'ADN, dont 5 ont été construits[4] à partir des sites de liaison ADN-protéines présents dans le génome d'*Escherichia coli*, le dernier contenant des sites promoteurs dans les génomes de plantes à deux cotylédons (plantes *Dicot*). Ces ensembles sont présentés dans la table 3.5. Les sites de liaisons des 5 premiers ensembles sont disponibles à l'adresse `http://arep.med.harvard.edu/ecoli_matrices/`. Parmi les 55 ensembles proposés (et validés) dans [114], ceux dont le nombre et la longueur des séquences étaient les plus importants ont été sélectionnés. Ces deux critères de choix ont été retenus afin de délimiter un espace de recherche conséquent. En outre, les « *Sequence Logo* » des sites de liaison sont fournis (*cf.* Figures 3.10 à 3.14). Le dernier ensemble contient 131 séquences et est disponible à l'adresse `http://www.softberry.com/berry.phtml`. Chaque séquence contient une occurrence d'un promoteur, dont le « *Sequence Logo* » (*cf.* Figure 3.15) a été généré afin de disposer également de cet outil de comparaison.

Ensemble	Nb de séquences	ADN de	Longueur $[\min, \max]$	types de sites	Protéine régulatrice
1	9	*E. coli*	$[53, 806]$	liaison	ArgR
2	8	*E. coli*	$[53, 806]$	liaison, en tandems	ArgR
3	16	*E. coli*	$[100, 3\,841]$	liaison	LexA
4	18	*E. coli*	$[299, 5\,795]$	liaison	PurR
5	8	*E. coli*	$[251, 2\,594]$	liaison	TyrR
6	131	Plantes *Dicot*	$[251, 251]$	promoteurs	

Table 3.5 – Synthèse de 6 ensembles de séquences biologiques.

STAMS a été testé en premier lieu sur les 5 ensembles de sites de liaisons du génome d'*E. coli* en faisant varier la valeur de e, et avec plusieurs fonctions de score (incluant les foncions *Block-Based* Q-indépendantes, Q-dépendantes – pour différentes valeurs de Q, ainsi que la mesure de l'entropie utilisée dans PRATT). À partir de ces tests, le paramétrage consensuel a été fixé à $e = 5$, et comme fonction de score, la fonction *Block-Based* Q-dépendante (*cf.* Section 3.2.2 – page 49) ayant comme composantes $f^{=}(k) = 2\,k$ et $f^{\neq}(k) = -k$, avec $Q = 70\%$. Pour chaque ensemble, le programme a été exécuté 20 fois afin de vérifier que les résultats soient les mêmes, indépendamment des permutations engendrées, avec $p = 1$. Pour chaque jeu de séquences issues d'*E. coli*, les motifs consensuels (*cf.* Table 3.6) ont été établis[5] et comparés aux « *Sequence Logo* » correspondants.

Ensemble	Protéine régulatrice	Motif consensuel extrait	« *Sequence Logo* » associé
1	ArgR	`TGAATxAxxATxCAxxTxxxxTGxxT`	Figure 3.10
2	ArgR en tandem	`TGAATxAxxATxCAxxTAxxxTGxxTxxxAATxCA`	Figure 3.11
3	LexA	`ACTGTATATxxAxxCAG`	Figure 3.12
4	PurR	`GCxAxCGTTTTC`	Figure 3.13
5	TyrR	`TGTAAAxxAAxxTxTAC`	Figure 3.14

Table 3.6 – Motifs extraits par STAMS pour les jeux de séquences d'*E. coli*.

Pour le premier ensemble, les positions des motifs dans les séquences sont correctes. La longueur

[4] Les séquences ont été récupérées sur la banque de données GENBANK [13] en utilisant BLASTn pour chacun des sites de liaison de chaque ensemble. Cela explique également pourquoi le choix a été restreint à seulement 5 ensembles parmi les 55 disponibles.

[5] Les motifs ont été construits à partir de l'alignement des consensus proposés par STAMS pour chacune des 20 exécutions réalisées.

importante du motif extrait comparé au « *Sequence Logo* » est due au fait que, parmi les 9 séquences, 8 contiennent le motif en tandem, et que la $9^{\text{ème}}$ séquence contient uniquement le début de la seconde occurrence.

Pour les jeux d'essais 2, 3 et 5, la longueur des motifs consensuels est correcte, ainsi que leurs positions dans les séquences.

Concernant le $4^{ème}$ ensemble, les positions dans les séquences sont correctes et les caractères conservés sont très bien retranscrits. Néanmoins, le site de liaison n'est que partiellement détecté. Il manque en effet le début et la fin du motif. Ceci est dû à la faible conservation des premières bases dans les occurrences, comme en témoigne par ailleurs le « *Sequence Logo* » qui lui est associé.

Bien que ces résultats soient très satisfaisants, ils ont été comparés aux résultats obtenus par d'autres algorithmes. Il semble évident qu'il n'est pas possible de tester tous les algorithmes existants, aussi le choix a été fait d'utiliser des méthodes [re]connues, et utilisant des approches soit combinatoires (PRATT, TEIRESIAS [112], MEME [10] et SMILE [98]), soit probabilistes (échantillonnage de GIBBS [81] et CONSENSUS [67]). Pour chaque algorithme, le paramétrage (*e.g.* la longueur des motifs, des trous, ...) a été fixé en fonction des motifs à extraire. Les résultats ont ensuite été analysés au cas par cas.

Dans cette étude comparative, seule la qualité des résultats a été comparée (*cf.* Figure 3.16). En effet, les algorithmes ayant été testés *via* des interfaces *Web*, le temps de réponse (parfois plusieurs heures) n'a pas été pris en compte. Les critères retenus pour le caractère qualitatif sont bien évidemment subjectifs. Il s'agit notamment de la pertinence du motif (est-ce le bon ? Est-il tronqué ? Trop grand ? ...), sa bonne localisation sur les séquences, le modèle consensuel renvoyé, et le cas échéant la cohérence de ce modèle (est-il représentatif ? Lisible ?).

Les méthodes probabilistes GIBBS et CONSENSUS sont parvenues à extraire les sites de liaison avec à peu près la même précision que STAMPS. Concernant les méthodes combinatoires, l'algorithme MEME (ainsi que la combinaison de ses résultats avec MAST [11]) n'a donné le résultat correct que pour le troisième ensemble de séquences. L'algorithme PRATT n'a réussi à identifier que des sous-motifs avant la phase de raffinement, tandis que les motifs renvoyés après raffinement sont souvent trop longs et inexploitables. En effet, les motifs renvoyés contiennent beaucoup de caractères dégénérés (*cf.* Annexe A), y compris pour les positions significatives. TEIRESIAS a mis en évidence plus de 200 résultats pour chaque jeu d'essai. Il est impossible de distinguer les motifs à extraire dans les ensembles de motifs renvoyés. SMILE a ce même inconvénient. C'est pourquoi ces deux derniers algorithmes ne sont pas présents sur le diagramme.

Concernant le dernier ensemble de séquences (DICOT sur la figure), seul STAMPS a permis de mettre en évidence le motif recherché (TATAAAT) avec la bonne longueur et les bases significatives bien mises en évidence. En revanche, ce motif a été trouvé à beaucoup d'autres endroits dans les séquences. Les autres algorithmes ont parfois mis en évidence des motifs plus grands. Seuls MEME et MAST ont renvoyé un motif plus grand englobant ce promoteur.

3.4.3 Conclusion

Au vu de ces expérimentations, il apparaît clairement que STAMPS donne des résultats comparables avec les plus performants des algorithmes existants. Les bases significatives ont été clairement mises en évidence, encourageant ainsi l'utilisation de scores Q-dépendants. En outre, l'identification des motifs biologiques par STAMPS avec le paramètre $p = 1$ conforte l'idée que la définition formelle de la similarité

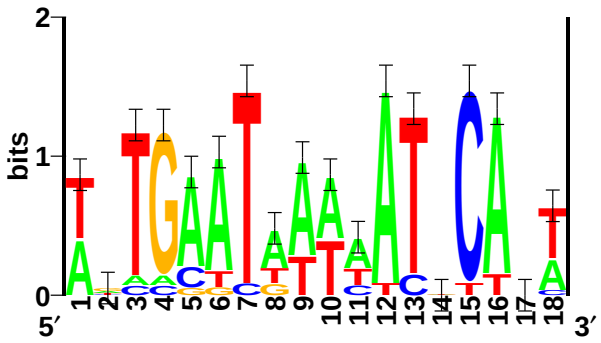


Figure 3.10 – « *Sequence Logo* » du site de liaison à ArgR.

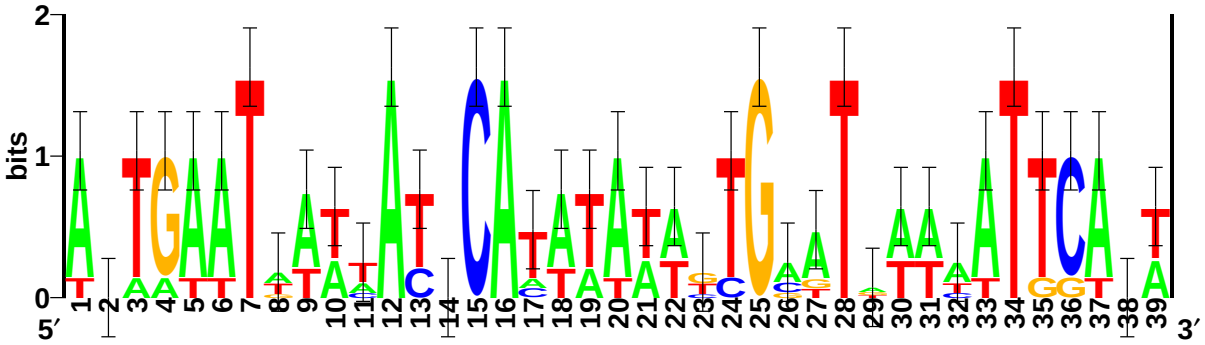


Figure 3.11 – « *Sequence Logo* » du site de liaison (en tandem) à ArgR.

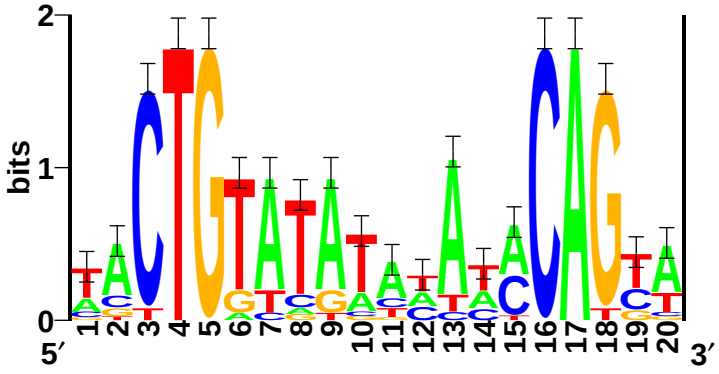


Figure 3.12 – « *Sequence Logo* » du site de liaison à LexA.

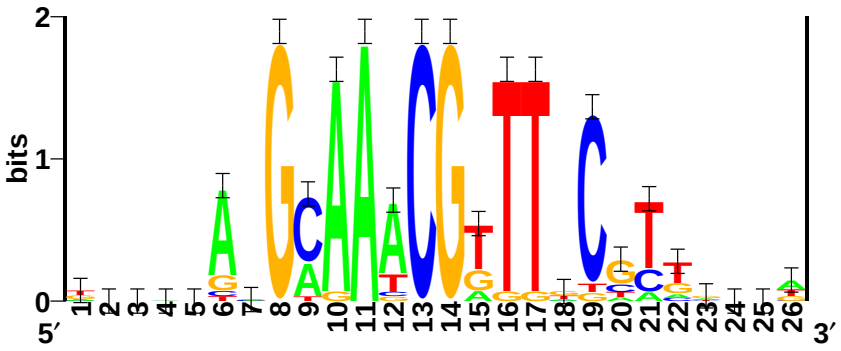


Figure 3.13 – « *Sequence Logo* » du site de liaison à PurR.

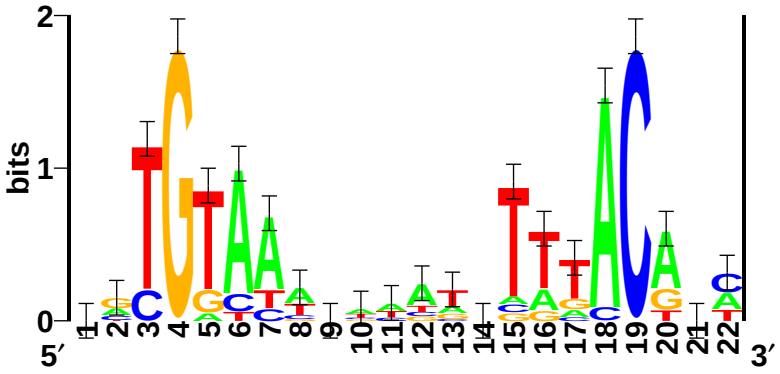


Figure 3.14 – « *Sequence Logo* » du site de liaison à TyrR.

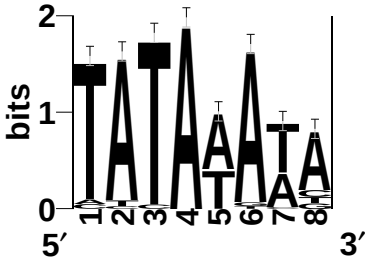


Figure 3.15 – « *Sequence Logo* » de la séquence promotrice des plantes *Dicot.*

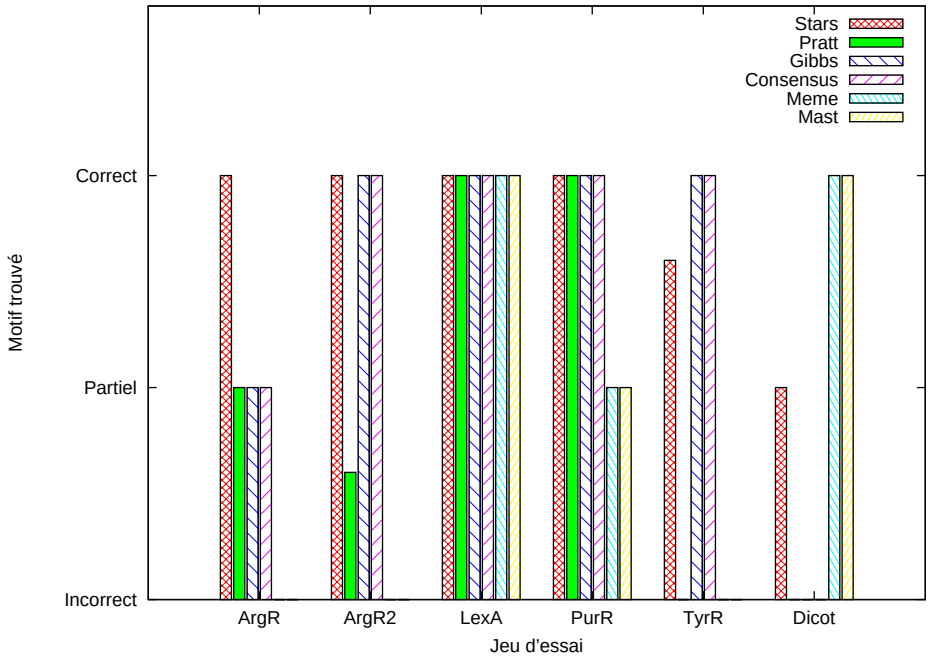


Figure 3.16 – Comparatif de plusieurs algorithmes sur des jeux de séquences biologiques.

(en particulier la e-similarité) est correcte du point de vue biologique. Enfin, les mêmes tests avec $p > 1$ ont renvoyé à chaque fois les motifs à extraire, les autres motifs renvoyés étant de scores inférieurs (souvent des sous-motifs des sites de liaisons/promoteurs).

Enfin, outre le fait que STAMS apporte une réponse très satisfaisante au problème de l'extraction de motifs, il offre une paramétrisation simplifiée et intuitive, le rendant beaucoup plus accessible aux utilisateurs finaux que la majorité des méthodes existantes. De plus, sa permissivité quant au choix des fonctions de score lui offre une grande modularité quant à son champ d'application. Bien que cet outil ait été développé dans le contexte très particulier de l'extraction de sites de liaisons ADN-protéines, il n'est pas saugrenu d'envisager son utilisation dans d'autres domaines que la bioinformatique.

CHAPITRE 4

Distribution limite des fonctions *Block-Based* et StatiSTARS

Toute connaissance dégénère en probabilité.

— David HUME [1711–1776], Traité de la nature humaine.

Ce chapitre est consacré d'une part à l'étude statistique des fonctions *Block-Based* introduite dans le précédent chapitre, et d'autre part à la présentation d'un nouvel algorithme polynomial, basé sur les propriétés statistiques de ces fonctions, répondant au problème de l'extraction de motifs sous contrainte de quorum commun dans un ensemble de séquences. Ces deux contributions sont respectivement présentés dans [15] et dans [94].

L'utilisation de fonctions de score dans les algorithmes d'extraction de motifs est souvent sujette à caution. Leur rôle ne peut souvent pas dépasser le stade de la valuation de la similarité entre motifs. Afin d'accréditer l'usage d'une fonction de score, plusieurs méthodes proposent de calculer ou bien d'estimer la moyenne et la variance de ladite fonction. Cela permet alors de calculer un nouveau score, appelé communément Z-valeur, Z-score ou également « score standardisé ». Cette information traduit l'écart entre le score et la moyenne. Celui-ci est mesuré en nombre d'écarts types, et permet de quantifier le caractère exceptionnel d'un score et trouve notamment comme applications, soit de comparer des mesures dont on ne connaît pas les distributions (éventuellement différentes), soit de fournir une indication statistique lorsque la distribution du score est connue. Dans ce dernier cas, la Z-valeur permet alors de calculer la probabilité d'obtenir au moins un tel score en fonction de sa distribution. Cette probabilité est également connue sous le nom de P-valeur ou encore de P-score. Il est évident qu'une Z-valeur est plus intéressante qu'un score dont on ignore tout de son comportement. Mais la connaissance de la P-valeur accrédite singulièrement la pertinence des résultats fournis par un algorithme (*cf.* [38] pour une discussion à ce propos). En pratique, peu de méthodes utilisent le concept de Z-score, la raison principale étant que le comportement des fonctions n'est pas toujours connu. Certains algorithmes approchent cette valeur en estimant la moyenne et la variance empiriquement ; cela se fait alors souvent au détriment du temps de calcul.

JONASSEN *& al.* [72] recommandent que la mesure utilisée respecte quelque propriétés. Entre autres, les auteurs suggèrent que la fonction de score soit représentative dans le même temps de la « puissance » du motif, et de la diversité des séquences. Bien que ces deux concepts semblent naturels de prime abord, ils sont relativement difficiles à mettre en œuvre, excepté en ayant une connaissance préalable de la distribution de la fonction de score utilisée. Considérons pour l'exemple le logiciel d'alignement de séquences BLAST [7]. Il s'agit de l'une des applications les plus utilisées en bioinformatique. Il a été démontré que le score utilisé par l'algorithme suit une *loi des valeurs extrêmes* (ou loi de GUMBEL) [7, 75]. Cette connaissance permet d'expliquer en grande partie le succès de BLAST. Un autre algorithme connu est FASTA [137, 91, 106]. Celui-ci est basé sur la distance de NEEDLEMAN-WUNSCH (*cf.* Définition 2.20 – page 27), qui suit également une loi de GUMBEL. D'autres méthodes, basées sur la mesure de l'entropie de l'information [121, 119] (*cf.* Section 2.2.1 – page 25), ont également des résultats probants, comme PRATT [40, 71] par exemple. Cette dernière mesure suit une loi de distribution *Gamma* [67] (*cf.* Annexe E.2), dont le comportement est régi par deux paramètres n et λ, dépendants notamment des fréquences des symboles de l'alphabet dans les séquences analysées.

La première partie de ce chapitre est consacrée à l'étude des fonctions *Block-Based* (basées sur le découpage en blocs consécutifs de « correspondances » et de « non correspondances »), présentées à la section 3.2.2 – page 49. Après quelques rappels de statistiques descriptives, cette famille de fonctions sera associée à une série génératrice, laquelle permettra de calculer la moyenne et la variance de chaque fonction composant cette famille. Le caractère gaussien de la distribution limite de ces fonctions est ensuite établi. La seconde partie de ce chapitre est alors dédiée à l'étude de StatiSTARS, un nouvel algorithme utilisant ces fonctions, et surtout leurs propriétés statistiques. Après avoir présenté l'algorithme, les résultats obtenus seront comparés à ceux de STARS.

4.1 Rappels de statistiques

Avant de décrire le contexte de l'étude, rappelons quelques notions de statistiques descriptives. Loin d'être exhaustive, cette première section a pour objectif de rappeler les concepts principaux utilisés dans les sections suivantes. La section 4.1.1 introduit la notion de variable aléatoire, et la section 4.1.2 – page ci-contre – les séries génératrices (introduisant notamment un lemme d'extraction de coefficients utilisé dans la suite). Pour de plus amples rappels, il est possible de se référer à [36, 54, 135].

4.1.1 Variables aléatoires réelles & moments

Définition 4.1. Étant donné un ensemble fondamental Ω correspondant à une expérience, les éléments de Ω sont les résultats possibles de l'expérience ou évènements. Une application $X : \Omega \to \mathbb{R}$ est une variable aléatoire. Cette variable aléatoire est dite discrète si $X(\Omega)$ est fini ou infini et dénombrable.

Notation 4.2. Par abus de notation, la valeur $X(\xi)$ (ξ étant un évènement de Ω) est notée X. Étant donné $x \in \mathbb{R}$, la probabilité que le résultat de l'expérience Ω par l'application X soit égal à x est notée $\text{Prob}[X = x]$.

Remarque 4.3. Étant donnée une variable aléatoire discrète X, par définition $\sum_{x \in X(\Omega)} \text{Prob}[X = x] = 1$. L'application $x \mapsto \text{Prob}[X = x]$ définit la loi (ou distribution) de probabilité de la variable aléatoire X.

Définition 4.4. Étant donnée une variable aléatoire discrète X, le premier moment $\mathbf{E}[X]$ de la variable aléatoire X est le nombre réel (s'il existe, *i.e.*, si la somme est absolument convergente) défini par

$$\mathbf{E}[X] = \sum_{x \in X(\Omega)} x \,\text{Prob}[X = x].$$

Ce moment définit la moyenne de la variable aléatoire X. Le second moment centré $\mathbf{Var}[X]$, de la variable aléatoire X est le nombre réel positif (s'il existe) défini par

$$\mathbf{Var}[X] = \sum_{x \in X(\Omega)} (x - \mathbf{E}[X])^2 \,\text{Prob}[X = x].$$

Ce moment est également appelé la variance de la variable aléatoire X.

Les deux propriétés suivantes sont fréquemment utilisées pour faciliter les calculs de la moyenne et de la variance d'une variable aléatoire discrète. La première traduit le changement d'échelle et la translation d'une variable aléatoire discrète, tandis que la seconde permet d'établir la relation existante entre moyenne et variance. Pour de plus amples détails sur les relations entre moyenne et variance (ainsi que pour les preuves de ces propriétés), il est possible de se référer à [54, pages 409–413].

Propriété 4.1. *Étant donnés une variable aléatoire discrète X qui admet un moment $\boldsymbol{E}[X]$, et un réel $k \in \mathbb{R}$, les propriétés suivantes sont vérifiées :*

$$\boldsymbol{E}[k \times X] = k \times \boldsymbol{E}[X] \qquad\qquad \boldsymbol{E}[X + k] = \boldsymbol{E}[X] + k$$

Propriété 4.2. *Étant donnée une variable aléatoire discrète X qui admet un second moment centré $\boldsymbol{Var}[X]$, alors l'égalité suivante est avérée :*

$$\boldsymbol{Var}[X] = \boldsymbol{E}\left[X^2\right] - \boldsymbol{E}^2\left[X\right].$$

4.1.2 Séries génératrices

Une série génératrice est un outil mathématique permettant de représenter formellement une suite infinie par une série entière. Cet outil peut être adapté à l'étude de variables aléatoires. La série est alors une série génératrice probabilisée.

Définition 4.5. Une source aléatoire de mots est une suite $(X_i)_{i\in\mathbb{N}}$ de variables aléatoires discrètes à valeur dans Σ. Un mot $w = \alpha_0 \cdots \alpha_n$ (avec $\forall 0 \leq i \leq n, \alpha_i \in \Sigma$) est émis par la source avec une probabilité $p_w = \text{Prob}[X_0 = \alpha_0 \wedge \cdots \wedge X_n = \alpha_n]$.

Conformément à la définition 4.3 – page précédente, pour tout $n \geq 0$, $\sum_{w\in\Sigma^n} p_w = 1$. En outre, p_w représente également la probabilité qu'un mot émis par la source ait w pour préfixe.

Définition 4.6. Une source aléatoire $(X_i)_{i\in\mathbb{N}}$ telle que les variables X_i soient indépendantes et de même loi est une source sans mémoire.

Les sources aléatoires sans mémoire offrent une représentation généralement éloignée de la réalité, toutefois, elles ont des propriétés particulières – page courante – les rendant beaucoup plus faciles à manipuler [14].

Propriété 4.3. *Étant donnée une source aléatoire sans mémoire $(X_i)_{i\in\mathbb{N}}$, alors pour tout $\alpha \in \Sigma$ et pour tout $i \geq 0$, Prob$[X_i = \alpha] = p_\alpha$ avec $0 < p_\alpha < 1$ et $\sum_{\alpha\in\Sigma} p_\alpha = 1$. De surcroît, pour tout n, Prob$[X_0 = \alpha_0 \wedge \cdots \wedge X_n = \alpha_n] = p_{\alpha_0} \times \cdots \times p_{\alpha_n}$.*

Remarque 4.7. Soit $w \in \Sigma^*$ un mot produit par une source sans mémoire de probabilités $\{p_\alpha \mid \alpha \in \Sigma\}$. Étant donnés deux mots u et v tels que $w = u\,v$, la probabilité d'émission du mot w est telle que $p_w = p_u\,p_v$. Cette remarque est reprise plus loin dans ce chapitre (*cf.* Propriété 4.4 – page 87).

Il est possible de définir une série entière associée à une caractéristique d'une source aléatoire de mots. Ces séries sont alors des séries dites génératrices. L'objectif de cette association est de formaliser les relations existantes entre les informations numériques (probabilités d'émissions, longueurs, scores, ...) relatives aux mots émis par la source.

Définition 4.8. Étant donné un ensemble de mots $\mathcal{L}$ produits par une source aléatoire, la série génératrice (probabilisée) simple associée à l'ensemble $\mathcal{L}$ est la série (entière) formelle $L(z)$ telle que :

$$L(z) := \sum_{w\in\mathcal{L}} p_w z^{|w|} = \sum_{n\geq 0} z^n \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w,$$

où p_w est la probabilité que le mot w soit engendré par la source.

Le rôle de la variable z est d'établir le lien entre la longueur des mots (l'exposant n) et leurs probabilités d'apparition (le coefficient p_w). L'étude d'une source aléatoire revient donc à étudier les caractéristiques de ces séries, en étudiant les coefficients des puissances de z. Ceux-ci sont obtenus en utilisant les théorèmes classiques d'analyse complexe [43], en considérant le domaine de convergence des séries.

Notation 4.9. Le coefficient de z^n dans la série formelle $L(z)$ est dénoté par $[z^n]L(z)$.

Le coefficient de z^n représente une information qui est sémantiquement reliée à la longueur des mots n. S'il est besoin de relier entre elles, non plus deux, mais trois informations, il est possible d'ajouter une autre variable portant en exposant la nouvelle information. Il est alors question de série génératrice double.

Définition 4.10. Étant donnée une fonction de coût $S(w)$ associée à un mot w, il est possible de définir la série génératrice (probabilisée) double du coût associée à l'ensemble $\mathcal{L}$ par la série formelle $L(z,u)$ suivante :

$$L(z,u) := \sum_{w\in\mathcal{L}} p_w u^{S(w)} z^{|w|} = \sum_{n\geq 0} z^n \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w u^{S(w)}. \tag{4.1}$$

Cette série, plus particulièrement son coefficient de z^n, est utilisée ci-après pour déterminer la moyenne et la variance de la fonction de coût sur tous les mots de même longueur n produits par la source aléatoire. Il est d'ores et déjà possible d'établir la remarque suivante :

Remarque 4.11. La dérivée partielle du coefficient de z^n par rapport à u dans la série formelle $L(z,u)$ est équivalente au coefficient de z^n dans la dérivée partielle par rapport à u de la série formelle $L(z,u)$.

$$\frac{\partial}{\partial u}\Big([z^n]L(z,u)\Big) \equiv [z^n]\left(\frac{\partial}{\partial u}L(z,u)\right).$$

Cette remarque peut être généralisée :

Remarque 4.12. La $k^{ème}$ dérivée partielle du coefficient de z^n par rapport à u dans la série formelle $L(z,u)$ est équivalente au coefficient de z^n dans la $k^{ème}$ dérivée partielle par rapport à u de la série formelle $L(z,u)$.

$$\frac{\partial^k}{\partial u^k}\Big([z^n]L(z,u)\Big) \equiv [z^n]\left(\frac{\partial^k}{\partial u^k}L(z,u)\right).$$

Principe du calcul de la moyenne et de la variance

Le calcul de la moyenne et de la variance d'une fonction de score associée à des mots de longueur fixée n générés par une source aléatoire se résume alors à calculer la moyenne et la variance du coût $S(w)$ dans la série $L(z,u)$ pour les mots w de longueur n. L'objectif poursuivi dans cette section est de déterminer la moyenne et la variance des fonctions de score *Block-Based* (*cf.* Section 3.2.2 – page 49), et cela en fonction de la probabilité d'apparition des symboles de l'alphabet dans les séquences, et de la longueur des mots alignés. Il est donc nécessaire de définir une source aléatoire de mots de longueur n correspondant aux mots alignements (*cf.* Définition 3.5 – page 50) utilisés dans le calcul par les fonctions de score de cette famille. Cette source définie, il « suffit » d'associer la série $L(z,u)$ à la variable aléatoire, notée S_n, représentant le score $S(w)$ des mots w de longueur n fixée, puis de calculer la moyenne et la variance de S_n. La source de mots est présenté à la section 4.2 – page 89, mais il est nécessaire auparavant de présenter quelques remarques et propriétés afin de procéder au calcul de la moyenne et de la variance.

Étant donnée la variable aléatoire S_n associée au score $S(w)$ des mots w de longueur n de l'ensemble $\mathcal{L}$ où chaque mot w a une probabilité p_w d'être émis par la source, la moyenne et la variance de la variable aléatoire S_n sont définies (Définition 4.4 – page 84) par

$$\mathbf{E}[S_n] := \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w S(w) \qquad\qquad \mathbf{Var}[S_n] := \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w (S(w) - \mathbf{E}[S_n])^2.$$

L'intérêt de la série génératrice $L(z,u)$ dans cette étude est précisément de déterminer la moyenne et de la variance de S_n sans pour autant calculer explicitement ces deux précédentes sommes [54, pages 419–420]. Les remarques 4.13 – page ci-contre – et 4.14 – page suivante – illustrent ce propos.

Remarque 4.13. La dérivée partielle du coefficient de z^n par rapport à u dans la série formelle $L(z,u)$ quand $u=1$ donne le premier moment de la variable aléatoire S_n.

$$\mathbf{E}[S_n] := \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w S(w) = [z^n]\frac{\partial}{\partial u}L(z,u)\bigg|_{u=1} \tag{4.2}$$

De façon similaire, la seconde dérivée partielle du coefficient de z^n par rapport à u dans la série formelle $L(z,u)$ quand $u=1$ permet de calculer le second moment ($\mathbf{E}\left[S_n^2\right]$) de la variable aléatoire S_n.

Remarque 4.14. La moyenne de S_n^2 est égale à la somme des deux premières dérivées partielles du coefficient de z^n par rapport à u dans la série formelle $L(z,u)$ quand $u=1$:

$$\mathbf{E}\left[S_n^2\right] := \sum_{\substack{w\in\mathcal{L},\\ |w|=n}} p_w (S(w))^2 = [z^n]\left(\frac{\partial^2}{\partial u^2}L(z,u)\bigg|_{u=1} + \frac{\partial}{\partial u}L(z,u)\bigg|_{u=1}\right). \tag{4.3}$$

Ainsi, le calcul de la variance de S_n se résume à calculer les dérivées partielles première (*cf.* Équations 4.2) et seconde (*cf.* Équations 4.3) de $L(z,u)$ (*cf.* Propriété 4.2 – page 84). La difficulté réside alors dans l'écriture de $L(z,u)$. En effet, il est nécessaire de trouver une expression de cette série qui se dérive « facilement », et dont il est possible d'extraire les coefficients.

Propriétés algébriques des séries génératrices

Il est possible de reprendre et d'étoffer la remarque 4.7 – page 85 – avec la propriété ci-dessous.

Propriété 4.4. *Soient un langage $\mathcal{L}$ associé à une source sans mémoire, une série formelle probabilisée associée à ce langage, et a, b, et w trois mot de $\mathcal{L}$ tels que $w = a\,b$. L'égalité $p_w = p_a\,p_b$ est vérifiée et, si le coût $S(w)$ est additif (i.e., $S(w) = S(a) + S(b)$), alors $p_w\,u^{S(w)}\,z^{|w|} = (p_a\,u^{S(a)}\,z^{|a|})\,(p_b\,u^{S(b)}\,z^{|b|})$.*

Démonstration. La probabilité d'émission p_w d'un mot w par la source est égale à la probabilité d'émission p_a du mot a multipliée par la probabilité d'émission du mot b sachant que le mot a a été émis ($p_{b|a}$). Or la source étant sans mémoire, la probabilité d'émission du mot b sachant que a a été émis est égale à la probabilité d'émission p_b du mot b. Ainsi, lorsque le coût est additif, $p_w\,u^{S(w)}\,z^{|w|} = (p_a\,p_b)\,u^{S(a)+S(b)}\,z^{|a|+|b|} = (p_a\,u^{S(a)}\,z^{|a|})\,(p_b\,u^{S(b)}\,z^{|b|})$. □

À partir de cette propriété, il est possible de définir un « dictionnaire » permettant de traduire des relations algébriques portant sur des ensembles vers des relations sur les séries génératrices [43, page 26]. Ainsi, lorsque la source est sans mémoire et que la fonction de coût est additive, il est possible de se référer à la table 4.1 – page suivante, où les séries $A(z,u)$, $B(z,u)$ et $C(z,u)$ sont les séries génératrices doubles associées aux ensembles de mots respectifs $\mathcal{A}$, $\mathcal{B}$ et $\mathcal{C}$.

Le lemme suivant permet d'obtenir le coefficient de z^n de certaines séries génératrices. Il est une conséquence du théorème basique de transfert présenté dans [42] et présenté dans une version simplifiée dans [14].

Lemme 4.5 (Extraction de coefficient)**.** *Soit une série entière $Q(z)$ qui s'écrit sous la forme :*

$$Q(z) = \left(\frac{1}{1-z}\right)^{b+1} z^m\,P(z),$$

Ensemble	Série Génératrice Associée
$\mathcal{A} = \Sigma$	$A(z,u) = z\left(\sum_{\alpha\in\Sigma} p_\alpha\, u^{S(\alpha)}\right)$
$\mathcal{A} = \mathcal{B}\cup\mathcal{C}$	$A(z,u) = B(z,u) + C(z,u)$
$\mathcal{A} = \mathcal{B}\times\mathcal{C}$	$A(z,u) = B(z,u)\times C(z,u)$
$\mathcal{A} = \mathcal{B}^*$	$A(z,u) = \dfrac{1}{1-B(z,u)}$

Table 4.1 – Équivalences entre relations ensemblistes et relations sur les séries génératrices.

où $P(z)$ est une fonction analytique dans un disque qui inclut $z=1$. Pour tout $n \geq m$, le coefficient de z^n dans $Q(z)$ est égal à :

$$[z^n]Q(z) = \sum_{r=0}^{b} (-1)^r \binom{n-m+b-r}{b-r} \frac{P^{(r)}(1)}{r!} + o(1).$$

Démonstration. Comme $P(z)$ est une fonction analytique au voisinage de $z=1$, elle se développe en série entière autour de 1 et

$$\begin{aligned} P(z) &= \sum_{r\geq 0} (z-1)^r \frac{P^{(r)}(1)}{r!} \\ &= \left(\sum_{r=0}^{b} (-1)^r \left(\frac{1}{1-z}\right)^{-r} \frac{P^{(r)}(1)}{r!}\right) + o\left((z-1)^b\right). \end{aligned}$$

La série $Q(z)$ est donc égale à

$$Q(z) = z^m \left(\left(\sum_{r=0}^{b} (-1)^r \left(\frac{1}{1-z}\right)^{b-r+1} \frac{P^{(r)}(1)}{r!}\right) + o(1)\right).$$

En remarquant [54, page 356] que $\left(\frac{1}{1-z}\right)^{c+1} = \sum_{n\geq 0} \binom{c+n}{n} z^n$, la série peut s'écrire

$$\begin{aligned} Q(z) &= z^m \left(\left(\sum_{r=0}^{b} (-1)^r \frac{P^{(r)}(1)}{r!} \sum_{n\geq 0} \binom{n+b-r}{n} z^n\right) + o(1)\right) \\ &= z^m \left(\left(\sum_{n\geq 0}\sum_{r=0}^{b} (-1)^r \binom{n+b-r}{b-r} \frac{P^{(r)}(1)}{r!} z^n\right) + o(1)\right). \end{aligned}$$

Par conséquent, pour $n \geq m$

$$\begin{aligned}
[z^n]Q(z) &= [z^n]\, z^m \left(\left(\sum_{n\geq 0} \sum_{r=0}^{b} (-1)^r \binom{n+b-r}{b-r} \frac{P^{(r)}(1)}{r!}\, z^n \right) + o(1) \right) \\
&= [z^{n-m}] \left(\left(\sum_{n\geq 0} \sum_{r=0}^{b} (-1)^r \binom{n+b-r}{b-r} \frac{P^{(r)}(1)}{r!}\, z^n \right) + o(1) \right) \\
&= \sum_{r=0}^{b} (-1)^r \binom{n-m+b-r}{b-r} \frac{P^{(r)}(1)}{r!} + o(1).
\end{aligned}$$

□

Remarque 4.15. Si $P(z)$ ne s'annule ni en 0, ni en 1 alors la somme correspond à la forme canonique du coefficient de z^n dans $Q(z)$. D'autre part, en ne considérant que les $b_0 < b$ premiers termes de la somme donnant le coefficient de z^n dans la série $Q(z)$, le terme d'erreur n'est plus $o(1)$, mais $o(n^{b-b_0})$.

4.2 Étude statistique des fonctions de scores *Block-Based*

Les propriétés, remarques et formules énoncées ci-avant sont ici appliquées à l'étude des fonctions *Block-Based* présentées au chapitre précédent (*cf.* Section 3.2.2 – page 49) et rappelées dans cette section. Comme mentionné à la section précédente, il est nécessaire de définir la source aléatoire de mots utilisée. Ceci fait, le cas simple des fonctions linéaires est analysé. Cette première étude se conclut par la preuve du caractère gaussien de la distribution des fonctions de score linéaires. Le cas plus général est ensuite abordé, et une expression de la moyenne et de la variance sont alors proposées. Une méthode permettant de calculer des formes closes de ces expressions est décrite, avec une application à quelques une des fonctions utilisées dans STAMS. Enfin, une expérimentation sur des sources sans mémoires, ainsi que sur des séquences biologiques est proposée, afin d'illustrer le caractère gaussien de la distribution de ces fonctions.

Les fonctions *Block-Based* sont basées sur le découpage en blocs consécutifs de « correspondances » et de « non correspondances » des symboles dans un alignement sans trous de deux mots de même longueur. Les fonctions de score appliquées à un alignement sans trou de deux mots de longueur n sont définies sur la notion de mot alignement (*cf.* Définition 3.5 – page 50), qui est un mot de longueur n de $\{0,1\}^*$. L'étude des propriétés statistiques des fonctions *Block-Based* nécessite donc de définir une source aléatoire de mots alignements.

Rappelons auparavant qu'un mot alignement w, défini sur $\{0,1\}^*$, peut s'écrire

$$w = 0^{k_0}1^{\ell_1}0^{k_1}\cdots 1^{\ell_t}0^{k_t}1^{\ell_{t+1}}, \tag{4.4}$$

avec $t \geq 0$, $\forall\, i \in \{1,\ldots,t\}, k_i, \ell_i > 0$ et $k_0, \ell_{t+1} \geq 0$.

Rappelons également qu'étant donnés un mot alignement w, et deux fonctions composantes de score $f^{=}$ (fonction de présence) et $f^{\neq}$ (fonction d'absence) définies de $\mathbb{N}$ dans $\mathbb{R}$, la fonction de score $S : \{0,1\}^* \to \mathbb{R}$ est définie par

$$S(w) = \sum_{i=1}^{t+1} f^{=}(\ell_i) + \sum_{i=0}^{t} f^{\neq}(k_i), \tag{4.5}$$

où $(\ell_1, \ldots, \ell_{t+1}) = l\_matches(w)$ et $(k_0, \ldots, k_t) = l\_mismatches(w)$.

Remarque 4.16. Les fonctions *Block-Based* ne sont pas additives, en effet, $S(1000101) \neq S(100) + S(0101)$. Toutefois, elles sont additives « par blocs » (par définition). De fait, $S(u\,v) = S(u) + S(v)$ si la dernière lettre de u est différente de la première lettre de v. Cette notion faible d'additivité sera suffisante pour l'étude, et le dictionnaire fourni par la table 4.1 – page 88 – reste valable à condition que la décomposition respecte le découpage par blocs.

Lorsque le mot w est généré aléatoirement, le score $S(w)$ lui-même est une variable aléatoire à valeur dans $\mathbb{R}$. Ainsi, l'étude de la moyenne, de la variance et de la distribution limite de cette variable aléatoire permet de déterminer le score moyen, sa variance et sa distribution lorsque le mot w est produit par une source sans mémoire de probabilités $\{p_0, p_1\}$ (p_0 et p_1 étant les probabilités d'émissions respectives des symboles 0 et 1, avec $p_0 + p_1 = 1$). Les probabilités p_1 et p_0 correspondent du point de vue biologique aux probabilités respectives de « correspondance » et de « non correspondance » entre deux symboles de deux séquences. Ainsi, si $p_\alpha^{(1)}$ (respectivement $p_\alpha^{(2)}$) désigne la probabilité d'apparition du symbole α dans la séquence s_1 (respectivement s_2) pour tout $\alpha \in \Sigma$, la probabilité de « correspondance » entre s_1 et s_2 pour un alignement donné est définie par

$$p_1 := \sum_{\alpha \in \Sigma} \sum_{\substack{\beta \in \Sigma \\ \beta =_{\mathcal{C}(\Sigma)} \alpha}} p_\alpha^{(1)}\ p_\beta^{(2)},$$

et la probabilité de « non correspondance » est définie par

$$p_0 := \sum_{\alpha \in \Sigma} \sum_{\substack{\beta \in \Sigma \\ \beta \neq_{\mathcal{C}(\Sigma)} \alpha}} p_\alpha^{(1)}\ p_\beta^{(2)}.$$

Il est facile de vérifier que la somme $p_0 + p_1$ vaut bien 1.

Pour la suite, il est nécessaire de poser quelques restrictions sur les fonctions composantes du score.

Restrictions sur les fonctions composantes de score. Afin de s'assurer que les premiers et seconds moments de la variable aléatoire S_n existent, il est nécessaire d'imposer que les fonctions composantes soient définies sur $[0, +\infty[$. Il est également nécessaire que ces fonctions soient d'ordre polynomial afin d'assurer la convergence de certaines sommes (cette restriction sera développée ultérieurement). Enfin, posons (afin de simplifier les calculs) que $f^{=}(0) = f^{\neq}(0) = 0$.

4.2.1 Cas des fonctions linéaires

L'étude de ces fonctions ne nécessite par d'avoir recours aux séries génératrices. En effet, il est possible de calculer la moyenne et la variance de la variable aléatoire S_n par un calcul combinatoire. De surcroît, le théorème utilisé pour démontrer que la distribution limite des fonctions linéaires est gaussienne permet de calculer directement leur moyenne et leur variance. Néanmoins, l'objectif de cette section est également d'illustrer sur un exemple simple l'emploi des séries génératrices, qui sont utilisées dans le cas général.

Les fonctions linéaires sont les fonctions dont les composantes du score sont toutes deux linéaires :

$$f^{=}(k) = \alpha\, k \qquad \text{et} \qquad f^{\neq}(k) = \beta\, k \qquad \text{avec } \alpha \text{ et } \beta \text{ deux constantes, } \alpha \neq \beta.$$

La fonction de score $S(w)$ correspondante est donc additive[1] (chaque symbole à une contribution fixée au score égale à α si c'est un 1 et β sinon). Dans le cas où $\alpha = \beta$, le score d'un mot w est alors $S(w) = \alpha\,|w|$, et le score moyen est $\alpha\,|w|$ et la variance du score est 0. Dans le cas où $\alpha \neq \beta$, la série génératrice associée au langage $\Sigma^* = (\{0\} \cup \{1\})^*$ s'écrit donc (*cf.* dictionnaire [Table 4.1 – page 88]) :

$$L(z,u) = \frac{1}{1 - z(p_0\,u^{f^{\neq}(1)} + p_1\,u^{f^{=}(1)})}.$$

Le calcul des première et seconde dérivées partielles par rapport à u de cette série génératrice, quand $u = 1$, permet de calculer les premier et second moments de la variable aléatoire S_n (*cf.* Remarques 4.13 – page 87 – et 4.14 – page 87).

$$\frac{\partial}{\partial u} L(z,u)\Big|_{u=1} = \frac{z\,c_1}{(1-z)^2} \tag{4.6}$$

$$\left(\frac{\partial^2}{\partial u^2} + \frac{\partial}{\partial u}\right) L(z,u)\Big|_{u=1} = \frac{z\,c_2}{(1-z)^2} + \frac{2\,z^2\,c_1^2}{(1-z)^3}, \tag{4.7}$$

où $c_1 = \alpha\,p_1 + \beta\,p_0$ et $c_2 = \alpha^2\,p_1 + \beta^2\,p_0$.

Lemme 4.6. *La moyenne de la variable aléatoire S_n associée aux scores obtenus par les fonctions Block-Based linéaires ($f^{=}(k) = \alpha\,k$ et $f^{\neq}(k) = \beta\,k$) sur des mots alignements générés par une source sans mémoire de probabilités $\{p_0, p_1\}$ est*

$$\boldsymbol{E}[S_n] = n\,(\alpha\,p_1 + \beta\,p_0). \tag{4.8}$$

Démonstration. En notant [54, page 356] que $[z^n](1-z)^{-c} = \binom{c+n-1}{n}$, il se déduit des équations 4.2 – page 87 – et 4.6 que la moyenne est :

$$\begin{aligned}
\mathbf{E}[S_n] &= [z^n](z\,c_1 \sum_{n\geq 0}(n+1)\,z^n) \\
&= [z^n](c_1 \sum_{n\geq 0} n\,z^n) \\
&= n\,(\alpha\,p_1 + \beta\,p_0).
\end{aligned}$$

□

Lemme 4.7. *La variance de la variable aléatoire S_n associée aux scores obtenus par les fonctions Block-Based linéaires ($f^{=}(k) = \alpha\,k$ et $f^{\neq}(k) = \beta\,k$) sur des mots alignements générés par une source sans mémoire de probabilités $\{p_0, p_1\}$ est*

$$\boldsymbol{Var}[S_n] = n\,p_0\,p_1\,(\alpha - \beta)^2. \tag{4.9}$$

[1] Le cas des composantes de score affi nes est traité dans le cas général, la fonction de score $S(w)$ n'étant alors plus additive.

Démonstration. Cette démonstration est similaire à la précédente. En effet, la variance se calcule de la même manière (*cf.* Remarque 4.2 – page 84) à partir des équations 4.3, 4.6 et 4.7 – pages 87 à 91 :

$$\begin{aligned}\mathbf{Var}[S_n] &= [z^n]\left(c_2\sum_{n\geq 0} n\,z^n + c_1^2\sum_{n\geq 0} n\,(n-1)\,z^n\right) - n^2\,c_1^2\\ &= c_2\,n + c_1^2\,n\,(n-1) - n^2\,c_1^2\\ &= n\,(c_2 - c_1^2)\\ &= n\,p_0\,p_1\,(\alpha-\beta)^2.\end{aligned}$$

□

La démonstration que la distribution de S_n est asymptotiquement gaussienne est basée sur le théorème de DE MOIVRE-LAPLACE, qui est un cas particulier du théorème de la limite centrale (*cf.* Nomenclature & Biographies).

Théorème 4.8 (Approximation de DE MOIVRE-LAPLACE)**.** *Étant donnée une variable aléatoire X suivant une loi binomiale de paramètres p et n (notée $\mathcal{B}(n,p)$), la loi de X tend vers une loi normale de paramètres $n\,p$ et $n\,p\,q$ (avec $q = 1-p$), notée $\mathcal{N}(n\,p, n\,p\,q)$.*

Théorème 4.9. *La variable aléatoire S_n associée aux scores obtenus par les fonctions Block-Based linéaires ($f^{=}(k) = \alpha\,k$ et $f^{\neq}(k) = \beta\,k$) sur des mots alignements générés par une source sans mémoire de probabilités $\{p_0, p_1\}$ converge vers une loi normale.*

Démonstration. Soit X_n une variable aléatoire associée au nombre de 1 dans un mot de $\{0,1\}^n$ émis par une source sans mémoire avec les probabilités $\{p_1, p_0\}$. La variable X_n suit une loi binomiale $\mathcal{B}(n, p_1)$, et selon le théorème d'approximation de DE MOIVRE-LAPLACE ci-dessus, la variable X_n suit asymptotiquement une loi normale $\mathcal{N}(n\,p_1, n\,p_1\,p_0)$.

Il est possible d'écrire S_n en fonction de X_n. En effet, un mot de longueur n, composé de x symboles 1 (et donc $n-x$ symboles 0) a un score $x\,\alpha + (n-x)\,\beta$. La variable S_n est donc telle que

$$\begin{aligned}S_n &= X_n\,\alpha + (n - X_n)\,\beta\\ &= X_n\,(\alpha-\beta) + n\,\beta.\end{aligned}$$

Il s'en suit donc que S_n tend également vers une loi normale. □

4.2.2 Cas général

Dans ce cas, la fonction de score n'est pas nécessairement additive. Il n'est donc pas possible d'utiliser la décomposition du langage précédente (*i.e.*, $(\{0\}\cup\{1\})^*$) pour trouver une formulation de la série génératrice $L(z,u)$. Il est donc nécessaire de trouver une autre décomposition du langage engendré par la source aléatoire. Les fonctions de score étant additives « par bloc », (*i.e.*, $S(u\,v) = S(u) + S(v)$ si le dernier symbole composant le mot u est différent du premier symbole composant le mot v), toute décomposition du langage qui respecte cette notion de « blocs » pourra être utilisée, et permettra de surcroît l'utilisation du dictionnaire afin de simplifier les calculs. En effet, l'usage du dictionnaire nécessite que la fonction de coût soit additive (*cf.* Remarque 4.16 – page 90). Or, dans le cas présent, elle n'est additive que par blocs. Les sous-ensembles utilisés par le dictionnaire devant valider le critère d'additivité, les

plus petits sous-ensembles utilisés doivent donc correspondre chacun à un bloc. Il faut également s'assurer que la décomposition du langage choisie est déterministe (*i.e.*, qu'il n'y ait pas deux manières de générer un mot du langage).

Il apparaît évident que tout mot de $\{0,1\}^*$ se décompose en suites consécutives de 0 et de 1 (*cf.* Équation 4.4 – page 89). Ainsi, l'ensemble de mots peut s'écrire :

$$\{0,1\}^* = \{0\}^* \times (\{1\}^+ \times \{0\}^+)^* \times \{1\}^*,$$

ou plus simplement, selon la syntaxe des expressions régulières :

$$(0|1)^* = 0^* \, (1^+ \, 0^+)^* \, 1^*. \tag{4.10}$$

Soient $S_0(z,u)$ et $S_1(z,)$ les séries génératrices doubles probabilisées respectivement associées aux ensembles 0^+ et 1^+ (et donc aux composantes de score respectives $f^{\neq}$ et $f^{=}$). Elles sont définies par :

$$S_0(z,u) := \sum_{k>0} p_0^k \, u^{f^{\neq}(k)} \, z^k \qquad \text{et} \qquad S_1(z,u) := \sum_{k>0} p_1^k \, u^{f^{\neq}(k)} \, z^k. \tag{4.11}$$

La série génératrice double probabilisée associée à l'ensemble 0^* (respectivement 1^*) s'obtient à partir du dictionnaire (*cf.* Table 4.1 – page 88), en notant que $0^* = \varepsilon|0^+$ (respectivement $1^* = \varepsilon|1^+$). Elle est donc égale à la somme de la série génératrice associée au mot vide $\sum_{n\in\mathbb{N}} [\![n=0]\!] \, z^n$ et de la série génératrice associée au mot 0^+ (respectivement 1^+), à savoir $1 + S_0(z,u)$ (respectivement $1 + S_1(z,u)$).

Le découpage par bloc étant respecté, les fonctions sont bien additives par bloc, et la propriété 4.4 – page 87 – reste valable. La série génératrice $L(z,u)$ de l'ensemble $(0|1)^*$ s'écrit donc (à l'aide du dictionnaire – page 88) à partir des équations 4.10 et 4.11 sous la forme :

$$L(z,u) = (1 + S_0(z,u)) \, \frac{1}{1 - S_1(z,u) \, S_0(z,u)} \, (1 + S_1(z,u)). \tag{4.12}$$

Avant de pouvoir calculer la moyenne et la variance de la variable S_n dans le cas général, il est nécessaire de formuler une dernière propriété.

Propriété 4.10. *Étant données deux fonctions composantes de score $f^{=}$ et $f^{\neq}$ d'ordre polynomial, les fonctions*

$$\begin{array}{lll} S_0(z,1), & \dfrac{\partial}{\partial u} S_0(z,u)\Big|_{u=1}, & \left(\dfrac{\partial^2}{\partial u^2} + \dfrac{\partial}{\partial u}\right) S_0(z,u)\Big|_{u=1}, \\ S_1(z,1), & \dfrac{\partial}{\partial u} S_1(z,u)\Big|_{u=1}, & \left(\dfrac{\partial^2}{\partial u^2} + \dfrac{\partial}{\partial u}\right) S_1(z,u)\Big|_{u=1}, \end{array}$$

sont analytiques dans un disque qui inclut $z = 1$.

Démonstration. La fonction $S_0(z,1)$ est égale à $\displaystyle\sum_{k>0} p_0^k \, z^k = \sum_{k\geq 0} p_0^k \, z^k - 1$. Or la série $\displaystyle\sum_{n\geq 0} x^n \, z^n = \frac{1}{1 - x\,z}$ (*cf.* [54, page 356]) est une série analytique définie sur un disque de centre 0 et de rayon $\frac{1}{x}$ (qui inclut donc 1 lorsque $x < 1$) [43, page 226]. C'est par conséquent également le cas de $S_0(z,1) = \dfrac{p_0 \, z}{1 - p_0 \, z}$.

La série $\dfrac{\partial}{\partial u} S_0(z,u)\Big|_{u=1}$ est, quant à elle, égale à $\sum_{k>0} p_0^k\, f^{\neq}(k)\, z^k$. Dans le cas où $z=1$, cette série est donc égale à $\sum_{k>0} p_0^k\, f^{\neq}(k)$. Le critère de CAUCHY permet d'établir la convergence d'une série $\sum_{n\geq 0} c_n$ s'il existe $N>0$ et une constante $0<\rho<1$ tels que $\forall n>N,\ \sqrt[n]{|c_n|}<\rho$. Ainsi pour que la série converge, il est nécessaire qu'il existe $N>0$ et $0<\rho<1$ tels que $\forall n>N,\ \sqrt[n]{|p_0^n\, f^{\neq}(n)|}<\rho$, ce qui est clairement vérifié ici, puisque $f^{\neq}(n)$ est polynomiale.

Concernant la série $\left(\dfrac{\partial^2}{\partial u^2}+\dfrac{\partial}{\partial u}\right) S_0(z,u)\Big|_{u=1}$, le raisonnement est le même. Cette série est égale à $\sum_{k>0} p_0^k \left(f^{\neq}(k)\right)^2 z^k$, ce qui équivaut à $\sum_{k>0} p_0^k \left(f^{\neq}(k)\right)^2$ lorsque $z=1$. La fonction $\left(f^{\neq}(k)\right)^2$ est également polynomiale, et le résultat précédent reste donc valable.

Les démonstrations pour les séries $S_1(z,1)=\dfrac{p_1\, z}{1-p_1\, z}$, $\dfrac{\partial}{\partial u} S_1(z,u)\Big|_{u=1} = \sum_{k>0} p_1^k\, f^{=}(k)\, z^k$ et $\left(\dfrac{\partial^2}{\partial u^2}+\dfrac{\partial}{\partial u}\right) S_1(z,u)\Big|_{u=1} = \sum_{k>0} p_1^k \left(f^{=}(k)\right)^2 z^k$ sont identiques. □

Étude du score moyen

Le calcul de la moyenne revient à calculer la dérivée première de $L(z,u)$ quand $u=1$ (*cf.* Remarque 4.13 – page 87).

Afin d'alléger l'écriture des expressions, les raccourcis de notation suivants sont adoptés :

$$S_0 := S_0(z,u), \qquad S_1 := S_1(z,u), \qquad S_0' := \frac{\partial}{\partial u} S_0(z,u), \qquad S_1' := \frac{\partial}{\partial u} S_1(z,u).$$

Les quantités $S_0\big|_{u=1}$ et $S_1\big|_{u=1}$, ainsi que quelques quantités les faisant intervenir sont utilisées dans la suite. Elles ont une expression particulièrement simple (*cf.* Propriété 4.10 – page précédente – et le début de sa preuve).

$$\begin{aligned} S_0\big|_{u=1} &= \frac{p_0}{1-p_0\, z}, & (1+S_0)\big|_{u=1} &= \frac{1}{1-zp_0}, & S_1\big|_{u=1} &= \frac{p_1}{1-p_1\, z}, \\ (1+S_1)\big|_{u=1} &= \frac{1}{1-zp_1}, & \frac{1}{1-(S_1S_0)\big|_{u=1}} &= \frac{(1-zp_0)(1-zp_1)}{1-z}. \end{aligned} \tag{4.13}$$

Théorème 4.11. *La moyenne de la variable aléatoire S_n associée aux scores obtenus par les fonctions Block-Based dont les composantes sont d'ordre polynomial sur des mots alignements générés par une source sans mémoire de probabilités $\{p_0,p_1\}$ est linéaire par rapport à n. Elle est telle que*

$$\boldsymbol{E}[S_n] = n\, k_1 + 2\, k_2 + k_1 - \overline{k_1} + o(1), \tag{4.14}$$

où $\qquad k_1 := \kappa_0'\, p_1^2 + \kappa_1'\, p_0^2, \qquad k_2 := p_0\, p_1\, (\kappa_0' + \kappa_1'), \qquad \overline{k_1} := \overline{\kappa_0'}\, p_1^2 + \overline{\kappa_1'}\, p_0^2,$

avec les constantes $\kappa'_0 = \sum_{k>0} f^{\neq}(k)\, p_0^k$, $\kappa'_1 = \sum_{k>0} f^{=}(k)\, p_1^k$,

et $\overline{\kappa'_0} = \sum_{k>0} k\, f^{\neq}(k)\, p_0^k$, $\overline{\kappa'_1} = \sum_{k>0} k\, f^{=}(k)\, p_1^k$.

Démonstration. La dérivée partielle de $L(z,u)$ par rapport à u s'écrit

$$\frac{\partial}{\partial u}L(z,u) = \frac{(S'_0\,(1+S_1) + S'_1\,(1+S_0))\;(1-S_0\,S_1) + (1+S_0)\,(1+S_1)\,(S'_1\,S_0 + S'_0\,S_1)}{(1-S_0\,S_1)^2}.$$

Après développement, puis simplification, cette expression devient :

$$\frac{\partial}{\partial u}L(z,u) = \frac{S'_0\,(1+S_1)^2 + S'_1\,(1+S_0)^2}{(1-S_0\,S_1)^2}. \tag{4.15}$$

En injectant les égalités de l'équation 4.13 – page précédente – dans l'équation 4.15, il se déduit que la dérivée en $u=1$ s'écrit alors :

$$\begin{aligned}\frac{\partial}{\partial u}L(z,u)\bigg|_{u=1} &= \frac{S'_0|_{u=1}\,(1-z\,p_0)^2 + S'_1|_{u=1}\,(1-z\,p_1)^2}{(1-z)^2}\\ &= \left(\frac{1}{1-z}\right)^2 z^0\;\left(S'_0\,(1-z\,p_0)^2 + S'_1\,(1-z\,p_1)^2\right)\bigg|_{u=1}.\end{aligned}$$

Il est donc possible d'appliquer le lemme d'extraction de coefficient (*cf.* Lemme 4.5 – page 87), avec les valeurs $b=1$, $m=0$ et $P(z) = \left(S'_0\,(1-z\,p_0)^2 + S'_1\,(1-z\,p_1)^2\right)\Big|_{u=1}$. Il est toutefois nécessaire, avant de procéder au calcul, de s'assurer que les dérivées partielles par rapport à z des séries S'_0 et S'_1 sont analytiques sur un disque qui inclut 1. Cependant, les séries entières possèdent la propriété d'invariance du rayon de convergence par dérivation par rapport à z du terme général [135]. Cela permet d'affirmer que si une série $\sum_{n\geq 0} c_n z^n$ converge absolument dans un disque ouvert de centre 0 et de rayon $R>0$, alors la série $\frac{\partial}{\partial z}\sum_{n\geq 0} c_n z^n = \sum_{n\geq 0} n\,c_n z^{n-1}$ converge absolument dans le disque ouvert de centre 0 et de rayon R. Les séries $S'_0|_{u=1}$ et $S'_1|_{u=1}$ sont analytiques sur un disque de centre 0 et de rayon $R>1$ (*cf.* Propriété 4.10 – page 93). Par conséquent, les séries $\frac{\partial}{\partial z}S'_0|_{u=1}$ et $\frac{\partial}{\partial z}S'_1|_{u=1}$, le sont également sur un disque qui inclut 1. Le lemme peut donc être appliqué.

Afin d'alléger l'écriture des calculs ultérieurs, posons

$$\kappa'_0 := \frac{\partial}{\partial u}S_0\bigg|_{\substack{u=1\\z=1}} = \sum_{k>0} p_0^k\, f^{\neq}(k), \qquad \kappa'_1 := \frac{\partial}{\partial z}S_1\bigg|_{\substack{u=1\\z=1}} = \sum_{k>0} p_1^k\, f^{=}(k)$$

$$\overline{\kappa'_0} := \frac{\partial^2}{\partial z\,\partial u}S_0\bigg|_{\substack{u=1\\z=1}} = \sum_{k>0} k\,p_0^k\, f^{\neq}(k), \qquad \overline{\kappa'_1} := \frac{\partial^2}{\partial z\,\partial u}S_1\bigg|_{\substack{u=1\\z=1}} = \sum_{k>0} k\,p_1^k\, f^{=}(k).$$

Le coefficient de z^n dans la dérivée de $L(z,u)$ par rapport à u quand $u=1$ est alors :

$$\begin{aligned}
[z^n]\frac{\partial}{\partial u}L(z,u)\Big|_{u=1} &= \sum_{r=0}^{1}\frac{(-1)^r}{r!}\binom{n+1-r}{1-r}\frac{\partial^r}{\partial z^r}\left(S'_0\,(1-z\,p_0)^2+S'_1\,(1-z\,p_1)^2\right)\Big|_{\substack{u=1\\z=1}}+o(1)\\
&= \binom{n+1}{1}\left(S'_0\,(1-z\,p_0)^2+S'_1\,(1-z\,p_1)^2\right)\Big|_{\substack{u=1\\z=1}}\\
&\quad-\binom{n}{0}\left(\frac{\partial}{\partial z}S'_0\,(1-z\,p_0)^2+\frac{\partial}{\partial z}S'_1\,(1-z\,p_1)^2\right.\\
&\qquad\left.-2\left(S'_0\,p_0\,(1-z\,p_0)+S'_1\,p_1\,(1-z\,p_1)\right)\right)\Big|_{\substack{u=1\\z=1}}+o(1)\\
&= \Big(n+1\Big)\Big(\kappa'_0\,(1-p_0)^2+\kappa'_1\,(1-p_1)^2\Big)\\
&\quad-\Big(\overline{\kappa'_0}\,(1-p_0)^2+\overline{\kappa'_1}\,(1-p_1)^2-2\left(\kappa'_0\,p_0\,(1-p_0)+\kappa'_1\,p_1\,(1-p_1)\right)\Big)+o(1)\\
&= \Big(n+1\Big)\Big(\kappa'_0\,p_1^2+\kappa'_1\,p_0^2\Big)-\Big(\overline{\kappa'_0}\,p_1^2+\overline{\kappa'_1}\,p_0^2\Big)+2\Big(p_0\,p_1\,(\kappa'_0+\kappa'_1)\Big)+o(1)
\end{aligned}$$

En substituant respectivement dans cette dernière formule les valeurs de $\kappa'_0\,p_1^2+\kappa'_1\,p_0^2$, $\overline{\kappa'_0}\,p_1^2+\overline{\kappa'_1}\,p_0^2$ et $p_0\,p_1\,(\kappa'_0+\kappa'_1)$ par les constantes k_1, $\overline{k_1}$ et k_2, la moyenne (*cf.* Remarque 4.13 – page 87) peut alors s'exprimer $\mathbf{E}[S_n]=n\,k_1+2\,k_2+k_1-\overline{k_1}+o(1)$. □

Application à quelques fonctions classiques.

Afin d'illustrer le théorème 4.11 – page 94, il est possible de confronter son application dans le cas des fonctions linéaires au résultat énoncé par le lemme 4.6 – page 91 (*i.e.*, pour $f^{\neq}(k)=\beta\,k$ et $f^{=}(k)=\alpha\,k, \alpha\neq\beta$).

Afin de calculer la moyenne de la variable S_n, il est nécessaire de calculer les constantes κ'_0, κ'_1, $\overline{\kappa'_0}$ et $\overline{\kappa'_1}$. Elles sont toutes calculables au moyen de la fonction d'ordre supérieur définie telle que

$$\mathcal{K}(k\mapsto f(k),p):=\sum_{k>0}f(k)\,p^k. \tag{4.16}$$

Ainsi, les constantes κ'_0, κ'_1, $\overline{\kappa'_0}$ et $\overline{\kappa'_1}$ sont respectivement égales dans le cas des fonctions linéaires à $\mathcal{K}(k\mapsto\beta\,k,p_0)$, $\mathcal{K}(k\mapsto\alpha\,k,p_1)$, $\mathcal{K}(k\mapsto\beta\,k^2,p_0)$ et $\mathcal{K}(k\mapsto\alpha\,k^2,p_1)$.

Avant de débuter les calculs, il est souhaitable d'établir la propriété suivante :

Propriété 4.12. *Étant donné un entier positif ℓ et un complexe z du disque ouvert de centre 0 et de rayon 1, l'égalité suivante est vérifiée :*

$$A(\ell,z):=\sum_{n\geq0}n^\ell\,z^n=\sum_{n=0}^{\ell}\left(\sum_{k=0}^{n}\binom{n}{k}k^\ell\,(-1)^{n-k}\right)\frac{z^n}{(1-z)^{n+1}}. \tag{4.17}$$

Démonstration. La série $A(\ell, z)$ peut s'écrire [54, page 373]

$$A(\ell, z) = \sum_{n=0}^{\ell} \left\{ \ell \atop n \right\} \frac{n!\, z^n}{(1-z)^{n+1}},$$

où $\left\{ \ell \atop n \right\}$ désigne le « nombre de STIRLING de deuxième espèce » (*cf.* Nomenclature & Biographies). Or toujours d'après [54, page 281],

$$\left\{ \ell \atop n \right\} n! = \sum_{k=0}^{n} \binom{n}{k} k^{\ell} \, (-1)^{n-k}.$$

Par conséquent,

$$A(\ell, z) = \sum_{n=0}^{\ell} \left(\sum_{k=0}^{n} \binom{n}{k} k^{\ell} \, (-1)^{n-k} \right) \frac{z^n}{(1-z)^{n+1}}.$$

□

Considérons à présent le cas de κ'_0 et de $\overline{\kappa'_0}$ (les calculs de κ'_1 et $\overline{\kappa'_1}$ étant identiques, aux constantes près, à ceux de κ'_0 et de $\overline{\kappa'_0}$) :

$$\kappa'_0 = \sum_{k>0} \beta\, k\, p_0^k = \beta\, A(1,z)\big|_{z=p_0} \qquad \overline{\kappa'_0} = \sum_{k>0} \beta\, k^2\, p_0^k = \beta\, A(2,z)\big|_{z=p_0} \tag{4.18}$$

$$= \beta \frac{p_0}{(1-p_0)^2} = \frac{\beta\, p_0}{p_1^2}, \qquad = \beta \frac{2\,p_0^2 + p_0\,(1-p_0)}{(1-p_0)^3} = \frac{\beta\, p_0\,(p_0+1)}{p_1^3}. \tag{4.19}$$

À partir de ces constantes, il est possible de calculer les valeurs de k_1, k_2 et $\overline{k_1}$:

$$k_1 := \beta\, p_0 + \alpha\, p_1, \qquad k_2 := p_0\, p_1\, (\frac{\beta\, p_0}{p_1^2} + \frac{\alpha\, p_1}{p_0^2}), \qquad \overline{k_1} := \frac{\beta\, p_0\,(p_0+1)}{p_1} + \frac{\alpha\, p_1\,(p_1+1)}{p_0},$$

afin de calculer la moyenne de S_n :

$$\begin{aligned}
\mathbf{E}[S_n] &= n\, k_1 + 2\, k_2 + k_1 - \overline{k_1} + o(1) \\
&= (n+1)\,(\alpha\, p_1 + \beta\, p_0) + \frac{2\,\beta\, p_0^3 + 2\,\alpha\, p_1^3 - \alpha\, p_1^2\,(p_1+1) - \beta\, p_0^2\,(p_0+1)}{p_0\, p_1} + o(1) \\
&= (n+1)\,(\alpha\, p_1 + \beta\, p_0) + \frac{\beta\, p_0^2\, \overbrace{(p_0-1)}^{-p_1} + \alpha\, p_1^2\, \overbrace{(p_1-1)}^{-p_0}}{p_0\, p_1} + o(1) \\
&= n\,(\alpha\, p_1 + \beta\, p_0) + o(1),
\end{aligned}$$

qui est bien – au terme en $o(1)$ près – identique à celle énoncée à l'équation 4.8 – page 91.

Le calcul des constantes κ'_0, κ'_1, $\overline{\kappa'_0}$ et $\overline{\kappa'_1}$ pour des fonctions données $f^{\neq}$ et $f^{=}$, est réalisable à l'aide de la fonction $\mathcal{K}$. En réalisant un « catalogue » des formules closes (lorsqu'elles existent) ou des mécanismes d'approximations (dans le cas contraire) pour le calcul de ces constantes pour une fonction composante de score donnée, il est possible de déterminer en temps fini la moyenne de la variable aléatoire S_n (*cf.* Théorème 4.11 – page 94).

La propriété 4.12 – page 96 – permet de déterminer des formules closes pour toutes les fonctions $k \mapsto \gamma\, k^\ell$, avec $\ell \geq 0$. La table 4.2 récapitule les valeurs de $\mathcal{K}$ pour ces fonctions, étant donnée une probabilité p pour les premières valeurs de ℓ. Le cas $\ell = 0$ est le cas particulier des fonctions constantes. Parmi celles-ci, les deux fonctions constantes $k \mapsto 0$ et $k \mapsto 1$ sont très intéressantes puisque la première permet de ne pas prendre en compte les « correspondances » (ou les « non correspondances ») dans le calcul du score tandis que la deuxième permet de compter le nombre de suites de « correspondances » (ou de « non correspondances »).

ℓ	0	1	2	3
$\mathcal{K}(k \mapsto \gamma\, k^\ell, p)$	$\dfrac{\gamma}{1-p}$	$\dfrac{\gamma\, p}{(1-p)^2}$	$\dfrac{\gamma\, p\,(p+1)}{(1-p)^3}$	$\dfrac{\gamma\, p\,(p^2+4\,p+1)}{(1-p)^4}$
	$\dfrac{\gamma\, p\,(p+1)\,(p^2+10\,p+1)}{(1-p)^5}$		$\dfrac{\gamma\, p\,(p^4+26\,p^3+66\,p^2+26\,p+1)}{(1-p)^6}$	
ℓ	4		5	

Table 4.2 – Constantes intervenant dans le calcul de $\mathbf{E}[S_n]$ selon les fonctions composantes de score.

Remarque 4.17. Il y a une propriété (évidente) de linéarité pour les fonctions composantes de score dans le sens suivant :

$$\mathcal{K}(k \mapsto (\alpha\, f + \beta\, g)(k), p) = \alpha\, \mathcal{K}(k \mapsto f(k), p) + \beta\, \mathcal{K}(k \mapsto b(k), p).$$

Ainsi, il est possible de déterminer les constantes associées aux fonctions composantes de score en décomposant ces dernières en fonctions élémentaires.

Si le calcul des constantes pour les polynômes se fait aisément à partir de formules closes (après une éventuelle décomposition en fonctions élémentaires), pour d'autres types de fonctions (comme $k \mapsto \sqrt{k}$ ou $k \mapsto \ln(1+k)$), il est difficile d'obtenir une telle formule. Cependant, les fonctions considérées étant d'ordre polynomial, le terme général $f(k)\, p^k$ de la série (*cf.* Équation 4.16 – page 96) tend très rapidement vers 0 et il suffit de calculer très peu de termes pour obtenir une bonne précision dans le calcul. Le terme d'erreur peut être majoré en notant par exemple que pour tout $k \geq 1$, $\ln(1+k) < k$ et $\sqrt{k} < k$. Ainsi l'imprécision ε (*i.e.*, le reste de la série) après approximation par la somme des n premiers termes pour une probabilité p fixée est majorée par

$$\begin{aligned}
\varepsilon_{p,n} = \sum_{k>n}^{\infty} k\, p^k &= \sum_{k>0}^{\infty} k\, p^k - \sum_{k>0}^{n} k\, p^k \\
&= \frac{p}{(1-p)^2} - p \sum_{k>0}^{n} k\, p^{k-1} \\
&= \frac{p}{(1-p)^2} - p\, \frac{\partial}{\partial p} \sum_{k\geq 0}^{n} p^k \\
&= \frac{p}{(1-p)^2} - p\, \frac{\partial}{\partial p}\, \frac{1-p^{n+1}}{1-p} \\
&= \frac{p - n\, p^{n+2} + n\, p^{n+1} + p^{n+1} - p}{(1-p)^2} \\
&= \frac{p^{n+1}\,(n - n\, p + 1)}{(1-p)^2}.
\end{aligned} \tag{4.20}$$

Cette majoration, permet également de corroborer le fait que la vitesse de convergence de ce type de séries est au moins géométrique.

Étude de la variance du score

Dans cette partie, aucun outil mathématique supplémentaire n'est nécessaire. Malgré une phase de calcul plus approfondie que dans le cas de la moyenne, le principe de l'étude de la variance est similaire à ce qui précède.

Le calcul de la variance nécessite de connaître les dérivées partielles premières et secondes par rapport à u de la série $L(z,u)$ quand $u = 1$ (*cf.* Propriété 4.2 – page 84 – et Remarque 4.14 – page 87). La dérivée première a déjà été calculée à la section précédente (*cf.* Équation 4.15 – page 95). La dérivée seconde s'obtient en dérivant l'expression de la dérivée première. Comme dans la partie précédente, les quantités S_0, S_1, S_0', et S_1' seront utilisées pour désigner les fonctions à deux variables $S_0(z,u)$ ou $S_1(z,u)$ et leurs dérivées premières par rapport à u. De la même manière, les quantités S_0'' et S_1'' désigneront leurs dérivées secondes par rapport à u :

$$S_0'' := \frac{\partial^2}{\partial u^2} S_0(z,u), \qquad\qquad S_1'' := \frac{\partial^2}{\partial u^2} S_1(z,u).$$

Théorème 4.13. *La variance de la variable aléatoire S_n associée aux scores obtenus par les fonctions Block-Based dont les composantes sont d'ordre polynomial sur des mots alignements générés par une source sans mémoire de probabilités $\{p_0, p_1\}$ est linéaire par rapport à n. Elle est telle que*

$$\textbf{\textit{Var}}[S_n] = n\,\left(k_1^{(2)} + k_1 + 2\,k_3 - k_1^2 - 2\,k_1\,\overline{k_1} + 2\,k_1\,k_2\right) + o(n). \tag{4.21}$$

où $\quad k_1 := \kappa_0'\,p_1^2 + \kappa_1'\,p_0^2, \qquad \overline{k_1} := \overline{\kappa_0'}\,p_1^2 + \overline{\kappa_1'}\,p_0^2, \qquad k_1^{(2)} := \kappa_0''\,p_1^2 + \kappa_1''\,p_0^2,$
$k_2 := p_0\,p_1\,(\kappa_0' + \kappa_1'), \qquad k_3 := p_0\,p_1\,\kappa_0'\,\kappa_1',$

avec les constantes $\kappa_0' = \sum_{k>0} f^{\neq}(k)\,p_0^k, \quad \overline{\kappa_0'} = \sum_{k>0} k\,f^{\neq}(k)\,p_0^k, \quad \kappa_0'' := \sum_{k>0} p_0^k\,f^{\neq}(k)\,(f^{\neq}(k)-1),$

et $\kappa_1' = \sum_{k>0} f^{=}(k)\,p_1^k, \quad \overline{\kappa_1'} = \sum_{k>0} k\,f^{=}(k)\,p_1^k \quad \kappa_1'' := \sum_{k>0} p_1^k\,f^{=}(k)\,(f^{=}(k)-1).$

Démonstration. La variance peut s'obtenir à partir de $\mathbf{E}\left[S_n^2\right]$ et de $\mathbf{E}[S_n]$ (*cf.* Propriété 4.2 – page 84). La valeur de $\mathbf{E}[S_n]$ est déjà connue (*cf.* Théorème 4.11 – page 94). Ainsi, il reste à calculer la valeur de $\mathbf{E}\left[S_n^2\right]$. D'après la formule de l'équation 4.3 – page 87,

$$\mathbf{E}\left[S_n^2\right] = [z^n]\left(\frac{\partial^2}{\partial u^2} + \frac{\partial}{\partial u}\right) L(z,u)\Big|_{u=1}.$$

À partir de l'expression de $\dfrac{\partial}{\partial u} L(z,u)$ (*cf.* Équation 4.15 – page 95), en dérivant une nouvelle fois

par rapport à u, et en ajoutant la dérivée première, nous obtenons après simplifications

$$\left(\frac{\partial^2}{\partial u^2}+\frac{\partial}{\partial u}\right) L(z,u) = \frac{1}{(1-S_0\,S_1)^2}\Big((S_0''+S_0')\,(1+S_1)^2+(S_1''+S_1')\,(1+S_0)^2 + 2\,S_0'\,S_1'\,(1+S_0+1+S_1)\Big) + \frac{2}{(1-S_0\,S_1)^3}\left(\left(S_0'\,(S_1+1)^2+S_1'\,(S_0+1)^2\right)\,(S_0'\,S_1+S_1'\,S_0)\right).$$

Toutes les quantités sont analytiques quand $u=1$ sur un disque qui inclut $z=1$ pour les fonctions étudiées (*cf.* Propriété 4.10 – page 93). Ainsi, pour obtenir $\mathbf{E}\left[S_n^2\right]$, il suffit d'extraire le coefficient de z^n dans cette série quand $u=1$. Toutefois, il est possible de fractionner l'extraction du coefficient en plusieurs étapes afin de simplifier les calculs. Ainsi, la démarche adoptée dans la suite de cette preuve consiste à extraire les termes du coefficient de z^n par degrés décroissants, et cela, pour chacune des deux composantes de la série.

Étude de la première composante :

$$S_A := \frac{2}{(1-S_0\,S_1)^3}\left(\left(S_0'\,(S_1+1)^2+S_1'\,(S_0+1)^2\right)\,(S_0'\,S_1+S_1'\,S_0)\right).$$

Cette composante fait intervenir (comme lors de l'étude de la moyenne) les valeurs $1+S_0$, $1+S_1$ et $S_0\,S_1$ qui ont une expression facile à manipuler quand $u=1$ (*cf.* Équation 4.13 – page 94). Ainsi,

$$S_A|_{u=1} = \frac{2\,(1-p_0\,z)^3\,(1-p_1\,z)^3}{(1-z)^3}\left(\frac{S_0'|_{u=1}}{(1-p_1\,z)^2}+\frac{S_1'|_{u=1}}{(1-p_0\,z)^2}\right)\left(\frac{S_0'|_{u=1}\,p_1\,z}{(1-p_1\,z)}+\frac{S_1'|_{u=1}\,p_0\,z}{(1-p_0\,z)}\right)$$
$$S_A|_{u=1} = \frac{2\,z}{(1-z)^3}\left(S_1'\,p_0\,(1-p_1\,z)+S_0'\,p_1\,(1-p_0\,z)\right)\left(S_1'\,(1-p_1\,z)^2+S_0'\,(1-p_0\,z)^2\right)\Big|_{u=1}.$$

Le lemme 4.5 – page 87 – s'applique de nouveau avec $b=2$, $m=1$ et

$$P(z) = 2\left(S_1'\,p_0\,(1-p_1\,z)+S_0'\,p_1\,(1-p_0\,z)\right)\left(S_1'\,(1-p_1\,z)^2+S_0'\,(1-p_1\,z)^2\right)\Big|_{u=1}.$$

Le coefficient de z^n dans $S_A|_{u=1}$ est donc égal à

$$[z^n]S_A|_{u=1} = \underbrace{\binom{n+1}{2}P(1)}_{S_A^{[0]}} - \underbrace{\binom{n}{1}\frac{\partial}{\partial z}P(z)|_{z=1}}_{S_A^{[1]}} + \underbrace{\binom{n-1}{0}\frac{\partial^2}{\partial z^2}\frac{P(z)|_{z=1}}{2}}_{S_A^{[2]}} + o(1).$$

Il reste alors à calculer les trois quantités $S_A^{[0]}$, $S_A^{[1]}$ et $S_A^{[2]}$. Comme précédemment, les quantités

$$\kappa_0' := \frac{\partial}{\partial u}S_0\Big|_{\substack{u=1\\z=1}} = S_0'|_{\substack{u=1\\z=1}} \qquad \kappa_1' := \frac{\partial}{\partial z}S_1\Big|_{\substack{u=1\\z=1}} = S_1'|_{\substack{u=1\\z=1}},$$
$$\overline{\kappa_0'} := \frac{\partial^2}{\partial u\,\partial z}S_0\Big|_{\substack{u=1\\z=1}}, \qquad \overline{\kappa_1'} := \frac{\partial^2}{\partial u\,\partial z}S_1\Big|_{\substack{u=1\\z=1}},$$

sont utilisées afin d'alléger l'écriture, les fonctions $S'_0|_{u=1}$ et $S'_1|_{u=1}$ étant analytiques sur un disque qui inclut $z=1$ (*cf.* Propriété 4.10 – page 93).

$$\begin{aligned}
S_A^{[0]} &= n\,(n+1)\left(\kappa'_1\,p_0\,(1-p_1)+\kappa'_0\,p_1\,(1-p_0)\right)\left(\kappa'_1\,(1-p_1)^2+\kappa'_0\,(1-p_1)^2\right)\\
&= n\,(n+1)\left(\kappa'_1\,p_0^2+\kappa'_0\,p_1^2\right)^2;\\
S_A^{[1]} &= 2\,n\left(\kappa'_0\,p_1\,(1-p_0)+\kappa'_1\,p_0\,(1-p_1)\right)\left(\overline{\kappa'_0}\,(1-p_0)^2+\overline{\kappa'_1}\,(1-p_1)^2\right.\\
&\qquad\qquad\left.-2\left(\kappa'_0\,p_0\,(1-p_0)+\kappa'_1\,p_1\,(1-p_1)\right)\right)\\
&\quad+\left(\overline{\kappa'_0}\,p_1\,(1-p_0)+\overline{\kappa'_1}\,p_0\,(1-p_1)-p_0\,p_1\,(\kappa'_0+\kappa'_1)\right)\left(\kappa'_0\,(1-p_0)^2+\kappa'_1\,(1-p_1)^2\right)\\
&= 2\,n\left(\kappa'_0\,p_1^2+\kappa'_1\,p_0^2\right)\left(2\left(\overline{\kappa'_0}\,p_1^2+\overline{\kappa'_1}\,p_0^2\right)-3\left(p_0\,p_1\,(\kappa'_0+\kappa'_1)\right)\right);\\
S_A^{[3]} &= O(1).
\end{aligned}$$

Ainsi en substituant respectivement les quantités $\kappa'_0\,p_1^2+\kappa'_1\,p_0^2$, $\overline{\kappa'_0}\,p_1^2+\overline{\kappa'_1}\,p_0^2$ et $p_0\,p_1\,(\kappa'_0+\kappa'_1)$ par k_1, $\overline{k_1}$ et k_2, le coefficient de z^n dans S_A quand $u=1$ s'écrit

$$\begin{aligned}
S_A|_{u=1} &= n\,(n+1)\,k_1^2-2\,n\,k_1\,(2\,\overline{k_1}-3\,k_2)+O(1)\\
&= n^2\,k_1^2+n\,(k_1^2-4\,k_1\,\overline{k_1}+6\,k_1\,k_2)+O(1).
\end{aligned}\tag{4.22}$$

Étude de la seconde composante :

$$S_B := \frac{1}{(1-S_0\,S_1)^2}\left((S''_0+S'_0)\,(1+S_1)^2+(S''_1+S'_1)\,(1+S_0)^2+2\,S'_0\,S'_1\,(1+S_0+1+S_1)\right).$$

Cette seconde composante fait également intervenir les quantités $1+S_0$, $1+S_1$ et $\dfrac{1}{1-S_0\,S_1}$. Lorsque $u=1$, cette composante s'écrit

$$\begin{aligned}
S_B|_{u=1} &= \frac{(1-p_0\,z)^2\,(1-p_1\,z)^2}{(1-z)^2}\left(\frac{(S'_0+S''_0)}{(1-p_1\,z)^2}+\frac{(S'_1+S''_1)}{(1-p_0\,z)^2}\right.\\
&\qquad\qquad\left.+2\,S'_0\,S'_1\left(\frac{2-z}{(1-p_1\,z)\,(1-p_0\,z)}\right)\right)\Bigg|_{u=1}\\
&= \left(\frac{1}{(1-z)}\right)^2 z^0\left((S'_0+S''_0)\,(1-p_0\,z)^2+(S'_1+S''_1)\,(1-p_1\,z)^2\right.\\
&\qquad\qquad\left.+2\,S'_0\,S'_1\,(2-z)\,(1-p_1\,z)\,(1-p_0\,z)\right)\Bigg|_{u=1}.
\end{aligned}$$

Le lemme d'extraction de coefficient (Lemme 4.5 – page 87) s'applique cette fois ci encore avec $b=1$, $m=0$ et

$$\begin{aligned}
P(z) = \big((S'_0+S''_0)\,(1-p_0\,z)^2+(S'_1+S''_1)\,(1-p_1\,z)^2\\
+2\,(S'_0\,S'_1)\,(2-z)\,(1-p_1\,z)\,(1-p_0\,z)\big)\big|_{u=1}.
\end{aligned}$$

En vue de simplifier l'écriture, deux nouvelles notations κ_0'' et κ_1'' sont utilisées

$$\kappa_0' := \frac{\partial}{\partial u} S_0 \Big|_{\substack{u=1\\z=1}}, \quad \overline{\kappa_0'} := \frac{\partial^2}{\partial u\,\partial z} S_0 \Big|_{\substack{u=1\\z=1}}, \quad \kappa_0'' := \frac{\partial^2}{\partial u^2} S_0 \Big|_{\substack{u=1\\z=1}} = \sum_{k>0} p_0^k\, f^{\neq}(k)\,(f^{\neq}(k)-1),$$
$$\kappa_1' := \frac{\partial}{\partial z} S_1 \Big|_{\substack{u=1\\z=1}}, \quad \overline{\kappa_1'} := \frac{\partial^2}{\partial u\,\partial z} S_1 \Big|_{\substack{u=1\\z=1}}, \quad \kappa_1'' := \frac{\partial^2}{\partial u^2} S_1 \Big|_{\substack{u=1\\z=1}} = \sum_{k>0} p_1^k\, f^{=}(k)\,(f^{=}(k)-1).$$

Ainsi, le coefficient de z^n dans $S_B|_{u=1}$ est égal à

$$\begin{aligned}
[z^n]S_B|_{u=1} &= \binom{n+1}{1} P(1) + o(n) \qquad \text{(\textit{cf.} Remarque 4.15 – page 89)}\\
&= n\left((\kappa_0' + \kappa_0'')\,(1-p_0)^2 + (\kappa_1' + \kappa_1'')\,(1-p_1)^2 + 2\,\kappa_0'\,\kappa_1'\,(1-p_1)\,(1-p_0)\right) + o(n)\\
&= n\left(\kappa_0'\,p_1^2 + \kappa_0''\,p_1^2 + \kappa_1'\,p_0^2 + \kappa_1''\,p_0^2 + 2\,p_0\,p_1\,\kappa_0'\,\kappa_1'\right) + o(n).
\end{aligned}$$

En substituant alors respectivement les quantités $\kappa_0'\,p_1^2 + \kappa_1'\,p_0^2$, $\kappa_0''\,p_1^2 + \kappa_1''\,p_0^2$ et $\kappa_0'\,\kappa_1'\,p_0\,p_1$ par k_1, $k_1^{(2)}$ et k_3, le coefficient de z^n dans S_B quand $u=1$ s'écrit

$$[z^n]S_B|_{u=1} = n\,(k_1 + k_1^{(2)} + 2\,k_3) + o(n). \tag{4.23}$$

Ainsi, d'après les équations 4.22 – page précédente – et 4.23 :

$$\begin{aligned}
\mathbf{E}\left[S_n^2\right] &= [z^n](S_A + S_B)|_{u=1}\\
&= [z^n]S_A|_{u=1} + [z^n]S_B|_{u=1}\\
&= n^2\,k_1^2 + n\,(k_1^2 - 4\,k_1\,\overline{k_1} + 6\,k_1\,k_2) + O(1) + n\,(k_1 + k_1^{(2)} + 2\,k_3) + o(n)\\
&= n^2\,k_1^2 + n\,(k_1^2 - 4\,k_1\,\overline{k_1} + 6\,k_1\,k_2 + k_1 + k_1^{(2)} + 2\,k_3) + o(n).
\end{aligned} \tag{4.24}$$

Calcul de la variance

À partir de la propriété 4.2 – page 84, et des équations 4.14 – page 94 – et 4.24, il est possible de déduire la variance de S_n :

$$\begin{aligned}
\mathbf{Var}[S_n] &= \mathbf{E}\left[S_n^2\right] - \mathbf{E}^2\left[S_n\right]\\
&= [z^n]\left(\frac{\partial^2}{\partial u^2} + \frac{\partial}{\partial u}\right) L(z,u) - \left(n\,k_1 + 2\,k_2 + k_1 - \overline{k_1} + o(1)\right)^2,\\
&= n^2\,k_1^2 + n\,(k_1^2 - 4\,k_1\,\overline{k_1} + 6\,k_1\,k_2 + k_1 + k_1^{(2)} + 2\,k_3) + o(n)\\
&\quad - n^2\,k_1^2 - n\,(4\,k_1\,k_2 + 2\,k_1^2 - 2\,k_1\,\overline{k_1}) - o(1),\\
&= n\,(k_1^{(2)} + k_1 + 2\,k_3 - k_1^2 - 2\,k_1\,\overline{k_1} + 2\,k_1\,k_2) + o(n).
\end{aligned}$$

La variance de S_n est par conséquent une fonction linéaire par rapport à n. □

Application à quelques fonctions classiques.

Comme pour le calcul de la moyenne, il est possible de s'assurer que dans le cas des fonctions linéaires, la variance obtenue par le biais du théorème 4.13 – page 99 – est la même (au terme d'erreur $o(n)$ près) que celle donnée par l'équation 4.9 – page 91.

Pour cela, il faut calculer les valeurs de κ'_0, κ'_1, $\overline{\kappa'_0}$, $\overline{\kappa'_1}$, κ''_0 et κ''_1 intervenant dans le calcul des quantités k_1, k_2, $\overline{k_1}$, k_3 et $k_1^{(2)}$. Les quatre premières valeurs à calculer étant les mêmes que celles utilisées dans le calcul de la moyenne (*cf.* Équation 4.18 – page 97), il est possible d'utiliser la fonction $\mathcal{K}(f(k),p)$ (*cf.* Équation 4.16 – page 96). Pour le calcul des deux dernières valeurs (à savoir κ''_0 et κ''_1). Définissons la fonction d'ordre supérieur

$$\mathscr{K}(k\mapsto f(k),p) := \sum_{k>0} f(k)\,(f(k)-1)\,p^k. \tag{4.25}$$

Les valeurs κ''_0 et κ''_1 sont par conséquent égales respectivement à $\mathscr{K}(f^{\neq},p_0)$ et $\mathscr{K}(f^{=},p_1)$.

Remarque 4.18. Sur la base de la remarque 4.17 – page 98, la fonction $\mathscr{K}$ s'écrit également

$$\begin{aligned}\mathscr{K}(k\mapsto f(k),p) &= \mathcal{K}(k\mapsto f(k)\,(f(k)-1),p)\\ &= \mathcal{K}(k\mapsto (f(k))^2,p) - \mathcal{K}(k\mapsto f(k),p).\end{aligned}$$

De cette dernière remarque, il se déduit que $\kappa''_0 = \mathcal{K}((f^{\neq})^2,p_0)-\kappa'_0$ et que $\kappa''_1 = \mathcal{K}((f^{=})^2,p_1)-\kappa'_1$. Dans le cas des fonctions linéaires, la valeur $\mathcal{K}((f^{\neq})^2,p_0)$ (respectivement $\mathcal{K}((f^{=})^2,p_1)$) est égale à $\mathcal{K}(k\mapsto\beta^2k^2,p_0)$ soit $\beta^2\,\mathcal{K}(k\mapsto k^2,p_0)$ (respectivement $\alpha^2\,\mathcal{K}(k\mapsto k^2,p_1)$). En utilisant de nouveau la fonction $A(\ell,z)$ (*cf.* Propriété 4.12 – page 96), les valeurs de $\mathcal{K}((f^{\neq})^2,p_0)$ et de $\mathcal{K}((f^{=})^2,p_1)$ sont donc respectivement égales à $\beta^2\,A(2,z)|_{z=p_0}$ et $\alpha^2\,A(2,z)|_{z=p_1}$, soit $\dfrac{\beta^2\,p_0\,(p_0+1)}{p_1^3}$ et $\dfrac{\alpha^2\,p_1\,(p_1+1)}{p_0^3}$ (*cf.* Table 4.2 – page 98). Par conséquent, les quantités k_1, k_2, $\overline{k_1}$, k_3 et $k_1^{(2)}$ sont égales à

$$k_1 = \beta\,p_0+\alpha\,p_1,\qquad k_2 = \frac{\beta\,p_0^2}{p_1}+\frac{\alpha\,p_1^2}{p_0},\qquad k_3=\alpha\,\beta,$$
$$\overline{k_1} = \frac{\beta\,p_0\,(p_0+1)}{p_1}+\frac{\alpha\,p_1\,(p_1+1)}{p_0},\qquad k_1^{(2)} = \frac{\beta^2\,p_0\,(p_0+1)}{p_1}+\frac{\alpha^2\,p_1\,(p_1+1)}{p_0}-(\beta\,p_0+\alpha\,p_1).$$

La variance de S_n est donc égale (selon le théorème 4.13 – page 99) à

$$\begin{aligned}\mathbf{Var}[S_n] &= n\,\left(k_1^{(2)}+k_1+2\,k_3-k_1^2-2\,k_1\,\overline{k_1}+2\,k_1\,k_2\right)+o(n)\\ &= n\,\left(\frac{\beta^2\,p_0\,(p_0+1)}{p_1}+\frac{\alpha^2\,p_1\,(p_1+1)}{p_0}+2\,\alpha\,\beta-(\beta\,p_0+\alpha\,p_1)^2\right.\\ &\qquad\left.-2\,(\beta\,p_0+\alpha\,p_1)\,\left(\frac{\beta\,p_0}{p_1}+\frac{\alpha\,p_1}{p_0}\right)\right)+o(n)\\ &= n\,\left(\frac{\beta^2\,p_0\,(1-p_0)}{p_1}+\frac{\alpha^2\,p_1\,(1-p_1)}{p_0}+2\,\alpha\,\beta\,(1-(p_1+p_0))-(\beta\,p_0+\alpha\,p_1)^2\right)+o(n)\\ &= n\,\left(\beta^2\,p_0\,(1-p_0)+\alpha^2\,p_1\,(1-p_1)-2\,\alpha\,\beta\,p_0\,p_1\right)+o(n)\\ &= n\,p_0\,p_1\,(\alpha-\beta)^2+o(n).\end{aligned}$$

Ce résultat est bien conforme (au terme en $o(n)$ près) au résultat énoncé par le lemme 4.7 – page 91.

Comme lors de l'étude de la moyenne de S_n, le calcule de la variance de la variable S_n nécessite de calculer les constantes κ'_0, κ'_1, $\overline{\kappa'_0}$ et $\overline{\kappa'_1}$ pour les fonctions $f^{\neq}$ et $f^{=}$. Il est également nécessaire de calculer les deux constantes nouvellement introduites κ''_0 et κ''_1. Pour les quatre premières constantes,

des outils de calcul de formules closes (*cf.* Équation 4.17 – page 96) ou approchées (*cf.* Équation 4.20 – page 98) ont déjà été proposés, permettant ainsi de calculer ces quatre constantes en un temps fini.

En s'appuyant sur la remarque 4.18 – page précédente, il demeure possible de déterminer les formules closes pour le calcul des constantes κ_0'' et κ_1'' dans le cas des fonctions composantes de score de la forme $f(k) = \gamma\, k^\ell$. De même qu'un catalogue a été proposé pour la fonction $\mathcal{K}$, pour quelques valeurs de ℓ (*cf.* Table 4.2 – page 98), un catalogue similaire est proposé pour les formules closes obtenues par la fonction $\mathscr{K}$ (*cf.* Table 4.3).

ℓ	0	1
$\mathscr{K}(k \mapsto \gamma\, k^\ell, p)$	$\dfrac{\gamma\,(\gamma-1)}{1-p}$	$\dfrac{\gamma\, p\,((\gamma+1)\, p+\gamma-1)}{(1-p)^3}$
	$\dfrac{\gamma\, p\,(p+1)\,((\gamma-1)\, p^2+(10\,\gamma+2)\, p+\gamma-1)}{(1-p)^5}$	
ℓ	2	

Table 4.3 – Constantes intervenant dans le calcul de $\mathbf{Var}[S_n]$ selon les fonctions composantes de score.

Remarque 4.19. Bien que la fonction $\mathscr{K}$ ne soit pas une application linéaire (contrairement à $\mathcal{K}$), il est tout de même possible de la décomposer (*cf.* Remarque 4.18 – page précédente) comme suit :

$$\begin{aligned}
\mathscr{K}(k \mapsto (\alpha\, f+\beta\, g)(k), p) &= \mathcal{K}(k \mapsto (\alpha\, f(k)+\beta\, g(k))^2, p) - \mathcal{K}(k \mapsto \alpha\, f(k)+\beta\, g(k), p),\\
&= \mathcal{K}(k \mapsto (\alpha\, f(k))^2, p) - \mathcal{K}(k \mapsto \alpha\, f(k), p)\\
&\quad + \mathcal{K}(k \mapsto (\beta\, g(k))^2, p) - \mathcal{K}(k \mapsto \beta\, g(k), p)\\
&\quad + 2\,\mathcal{K}(k \mapsto \alpha\,\beta\, f(k)\, g(k), p),\\
&= \mathscr{K}(k \mapsto \alpha\, f(k), p) + \mathscr{K}(k \mapsto \beta\, g(k), p) + 2\,\mathcal{K}(k \mapsto (\alpha\, f \times \beta\, g)(k).
\end{aligned}$$

Selon cette dernière remarque, il demeure donc possible de déterminer la variance des fonctions composantes de score en les décomposant en fonctions élémentaires.

Pour les fonctions comme $k \mapsto \sqrt{k}$ ou $k \mapsto \ln(1+k)$, il est au moins aussi difficile d'obtenir une formule close qu'avec la fonction $\mathcal{K}$. Cependant, en appliquant le même principe que pour $\mathcal{K}$, *i.e.*, en calculant les premiers termes de la somme, la précision obtenue est très rapidement suffisante. Toujours en remarquant que pour tout $k \geq 1, k \geq \sqrt{k} > \ln(1+k)$, il est possible de majorer l'imprécision après calcul des n premiers termes de $\mathscr{K}$. En effet,

$$\begin{aligned}
\mathscr{K}(k \mapsto \sqrt{k}, p) &= \mathcal{K}(k \mapsto k, p) + \mathcal{K}(k \mapsto \sqrt{k}, p) && \text{(\textit{cf.} Remarque 4.18 – page précédente)}\\
&\leq \frac{p}{1-p} + \sum_{k>0}^{n} p^k\, \sqrt{k} + \varepsilon_{p,n} && \text{(\textit{cf.} Équation 4.20 – page 98).}
\end{aligned}$$

De la même manière,

$$\begin{aligned}
\mathscr{K}(k \mapsto \ln(1+k), p) &= \mathcal{K}(k \mapsto (\ln(1+k))^2, p) + \mathcal{K}(k \mapsto \ln 1+k, p)\\
&\leq \sum_{k>0}^{n} p^k\, \ln(1+k) + \varepsilon_{p,n} + \sum_{k>0}^{n} p^k\, \ln(1+k) + \varepsilon_{p,n}\\
&\leq 2\left(\sum_{k>0}^{n} p^k\, \ln(1+k) + \varepsilon_{p,n}\right).
\end{aligned}$$

La vitesse de convergence de ce type de séries est donc (ici aussi) au moins géométrique.

4.2.3 Étude de la distribution

Dans un premier temps, plusieurs tests ont été réalisés, tant sur des séquences générées par des sources sans mémoires que sur des séquences d'ADN alignées sans trous. L'objectif étant d'avoir une idée de la distribution des fonctions de score. De ces expérimentations, l'hypothèse que toutes les fonctions de score *Block-Based* étudiées dans ce chapitre suivent une loi normale est devenue une conviction. Toutefois, si dans le cas des fonctions linéaires, le théorème d'approximation de DE MOIVRE-LAPLACE permet de le démontrer, il n'en est rien dans le cas plus général. En effet, la démonstration dans le cas des fonctions linéaires utilise la propriété d'additivité des fonctions. Or dans le contexte général, cette propriété n'est plus vérifiée. Le résultat a été prouvé ultérieurement à nos expérimentations dans [15] en utilisant des approximations de la loi et le théorème de la limite centrale. L'étude de la distribution dans un contexte plus général est évoquée en conclusion de cette section.

Protocole d'expérimentation.

Toutes les fonctions composantes de score qui ont été testées ont donné lieu à la même interprétation. Aussi, seul un exemple est commenté dans ce manuscrit. Les résultats présentés ci-après ont été obtenus en utilisant les fonctions composantes de score suivantes :

$$f^{\neq}(k) = -k, \qquad \text{et} \qquad f^{=}(k) = k^2,$$

le score global d'un mot $w = 0^{k_0}1^{\ell_1}0^{k_1}\cdots 1^{\ell_t}0^{k_t}1^{\ell_{t+1}}$, étant donné par (*cf.* Équation 4.5 – page 89)

$$\begin{aligned} S(w) &= \sum_{i=1}^{t+1} f^{=}(\ell_i) + \sum_{i=0}^{t} f^{\neq}(k_i) \\ &= \sum_{i=1}^{t+1} \ell_i^2 - \sum_{i=0}^{t} k_i. \end{aligned}$$

Pour chacune des deux fonctions composantes de score, il est possible de calculer une formule close pour les quantités κ_0', κ_1', $\overline{\kappa_0'}$, $\overline{\kappa_1'}$, κ_0'' et κ_1'' (*cf.* Section 4.2.2 – page 92). Par conséquent, il est possible de calculer une expression de la moyenne (*cf.* Théorème 4.11 – page 94) ainsi que de la variance (*cf.* Théorème 4.13 – page 99).

En effet, en reprenant les résultats énoncés dans les tables 4.2 – page 98 – et 4.3 – page ci-contre,

$$\begin{cases} \kappa_0' & = \mathcal{K}(k \mapsto f^{\neq}(k), p_0) & = \mathcal{K}(k \mapsto -k, p_0) & = -\dfrac{p_0}{p_1^2}, \\ \overline{\kappa_0'} & = \mathcal{K}(k \mapsto k\, f^{\neq}(k), p_0) & = \mathcal{K}(k \mapsto -k^2, p_0) & = -\dfrac{p_0\,(p_0+1)}{p_1^3}, \\ \kappa_0'' & = \mathscr{K}(k \mapsto f^{\neq}(k), p_0) & = \mathscr{K}(k \mapsto -k, p_0) & = \dfrac{2\,p_0}{p_1^3}, \\ \kappa_1' & = \mathcal{K}(k \mapsto f^{=}(k), p_1) & = \mathcal{K}(k \mapsto k^2, p_1) & = \dfrac{p_1\,(p_1+1)}{p_0^3}, \\ \overline{\kappa_1'} & = \mathcal{K}(k \mapsto k\, f^{=}(k), p_1) & = \mathcal{K}(k \mapsto k^3, p_1) & = \dfrac{p_1\,(p_1^2+4\,p_1+1)}{p_0^4}, \\ \kappa_1'' & = \mathscr{K}(k \mapsto f^{=}(k), p_1) & = \mathscr{K}(k \mapsto k^2, p_1) & = \dfrac{12\,p_1^2\,(p_1+1)}{p_0^5}. \end{cases}$$

Ainsi, les constantes intervenant dans l'expression de la moyenne et de la variance se calculent facilement[2],

$$\begin{cases} k_1 & = \kappa'_0\,p_1^2 + \kappa'_1\,p_0^2 & = \dfrac{2\,p_1 - p_0}{p_0}, \\ \overline{k_1} & = \overline{\kappa'_0}\,p_1^2 + \overline{\kappa'_1}\,p_0^2 & = \dfrac{6\,p_1^3 + p_0^2\,p_1 - 2\,p_0^3}{p_0^2\,p_1}, \\ k_1^{(2)} & = \kappa''_0\,p_1^2 + \kappa''_1\,p_0^2 & = \dfrac{12\,p_1^4 + 12\,p_1^3 + 2\,p_0^4}{p_0^3\,p_1}, \\ k_2 & = p_0\,p_1\,(\kappa'_0 + \kappa'_1) & = \dfrac{p_1^4 - p_0^4 + p_1^3}{p_0^2\,p_1}, \\ k_3 & = p_0\,p_1\,\kappa'_0\,\kappa'_1 & = -\dfrac{p_1 + 1}{p_0}. \end{cases}$$

Et par conséquent, en reprenant les formules des équations 4.14 – page 94 – et 4.21 – page 99,

$$\begin{cases} \mathbf{E}[S_n] & = n\,k_1 + 2\,k_2 + k_1 - \overline{k_1} + o(1) \\ & = \dfrac{n\,p_0\,(2 - 3\,p_0) - 2\,(p_0^2 - 2\,p_0 + 1)}{p_0^2} + o(1) \\ & = \dfrac{n\,p_0\,(2\,p_1 - p_0) - 2\,p_1^2}{p_0^2} + o(1), \\ \mathbf{Var}[S_n] & = n\,\left(k_1^{(2)} + k_1 + 2\,k_3 - k_1^2 - 2\,k_1\,\overline{k_1} + 2\,k_1\,k_2\right) + o(n) \\ & = \dfrac{4\,n\,(2 - p_0)\,(1 - p_0)}{p_0^3} + o(n) \\ & = \dfrac{4\,n\,p_1\,(p_1 + 1)}{p_0^3} + o(n). \end{cases}$$

Source sans mémoire

Les probabilités ont été fixées uniformément sur la base d'un alphabet de cardinalité 4 ($\{p_0 = 3/4, p_1 = 1/4\}$). La figure 4.1 compare les résultats expérimentaux aux résultats théoriques. L'axe des abscisses représente la longueur n des mots dont on calcule le score. Pour chaque longueur de mot, 20 000 mesures de score ont été effectuées. L'axe des ordonnées correspond aux scores S_n calculés. Chaque point représente donc un mot et son score. La moyenne théorique obtenue selon la formule ci-dessus, ainsi que les deux bandes de tolérances induites par l'écart-type ($= \sqrt{\mathbf{Var}[S_n]}$) figurent en pointillés. L'expérimentation est satisfaisante dans la mesure où il apparaît clairement que les scores calculés sont en corrélation avec la moyenne et la variance calculées.

Si la figure 4.1 est une validation expérimentale des expressions de la moyenne et de la variance, la figure 4.2 correspond à la fonction de répartition expérimentale pour les mots de longueur 1 000, centrée par rapport à la moyenne (*i.e.*,, la moyenne expérimentale des 20 000 scores calculés pour ces mots a été soustraite à tous ces scores) et réduite par l'écart-type (*i.e.*, les scores centrés des 20 000 mots de longueur 1 000 ont été divisés par l'écart-type expérimental calculé pour ces mots). L'axe des abscisses est étiqueté par les scores centrés réduits[3], tandis que l'axe des ordonnées représente le pourcentage de tests ayant obtenu un score inférieur ou égal à la valeur en abscisse. La fonction de répartition de la loi normale centrée réduite – ou loi de GAUSS – apparaît en pointillé (*cf.* Annexe E.1). Il apparaît nettement

[2] Les simplifi cations s'obtiennent aisément en remplaçant systématiquement p_1 par $1 - p_0$ dans les formules.

[3] Cette notion est formellement introduite à la page 110.

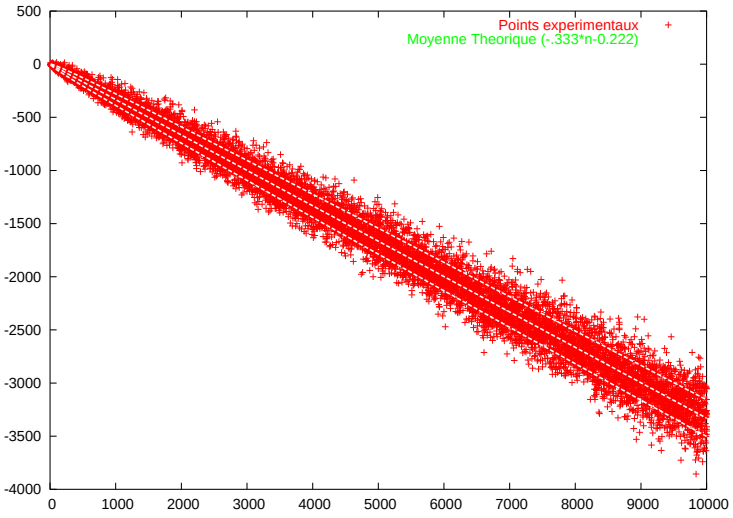


Figure 4.1 – Moyenne des scores obtenus avec des sources sans mémoire.

que les deux courbes se superposent. Ainsi, au vu des expérimentations, s'est renforcée la conviction que les fonctions de score *Block-Based* suivent une loi normale.

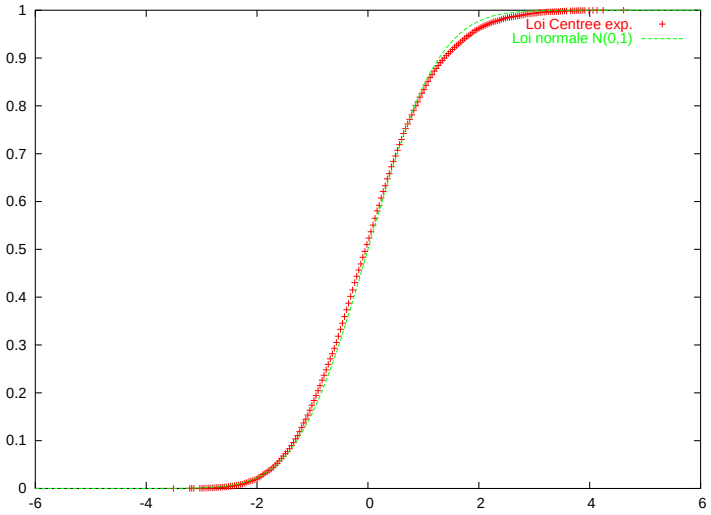


Figure 4.2 – Distribution des scores obtenus avec des sources sans mémoire.

Séquences biologiques

Les séquences ont été extraites du génome de *Bacillus subtilis* (environ quatre millions de nucléotides). Des résultats obtenus à partir de séquences provenant d'autres organismes sont présentés à la fin de cette section. La probabilité de « correspondances » a été calculée en considérant les fréquences d'apparition des quatre bases ($p_1 = 0{,}254\,188$ et $p_0 = 0{,}745\,812$, *cf.* Section 4.2.1 – page 90). Pour chaque calcul de score, deux séquences de même longueur n ont été choisies aléatoirement dans le génome, et ont été comparées base par base, afin de construire le mot alignement de taille n correspondant.

La figure 4.3 permet d'illustrer le fait que l'étude sur des sources sans mémoires donne une information relativement précise quant au comportement des fonctions de score lorsqu'elles sont appliquées sur des séquences biologiques. Comme pour la figure 4.1, il existe une forte corrélation entre les valeurs théoriques calculées, et les valeurs expérimentales.

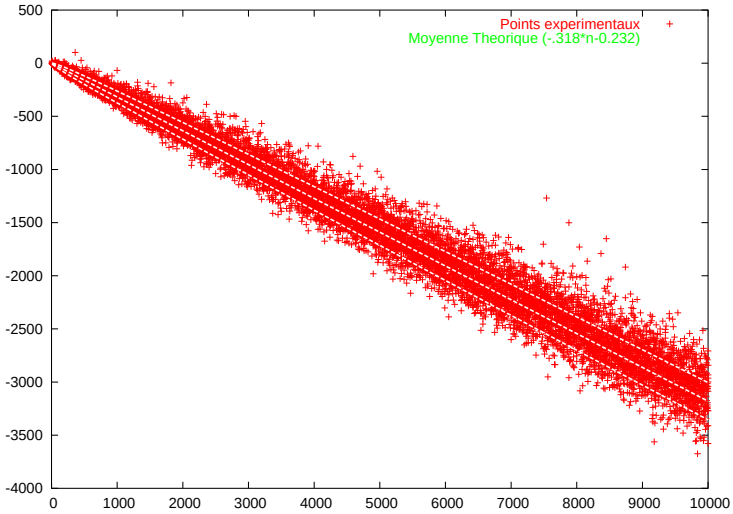


Figure 4.3 – Moyenne des scores obtenus avec des séquences issues de *B. subtilis*.

Comme lors de l'étude expérimentale sur les sources sans mémoires, la distribution des scores obtenus pour les séquences de taille 1 000 a été tracée (*cf.* Figure 4.4). Dans ce contexte également, 20 000 mesures de score ont été effectuées (*i.e.*, 20 000 mots générés).

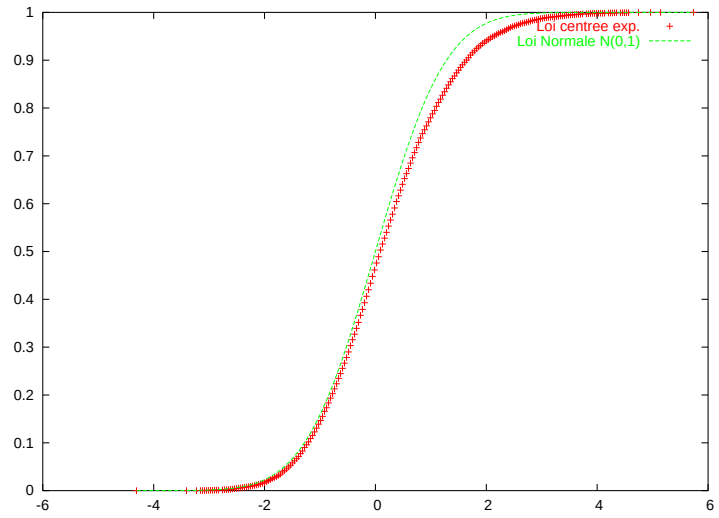


Figure 4.4 – Distribution des scores obtenus avec des séquences issues de *B. subtilis*.

Il est possible de remarquer que dans le cas des séquences réelles, la distribution semble rester gaussienne ; toutefois, un léger biais apparaît dans le cas des scores les plus élevés. Ce phénomène peut s'expliquer par le fait que la fonction de score favorise beaucoup les « correspondances ». Il n'est pas dé-

raisonnable de supposer qu'il y ait un peu plus de « correspondances » dans une séquence réelle que dans une séquence totalement prise au hasard. Cependant, ce biais n'est pas très important et ne remet pas en cause le caractère gaussien de la distribution. Enfin, la figure 4.5 illustre différents biais en fonction de l'origine des séquences biologiques[4] (*Arabidopsis thaliana*, *Bacillus subtilis* et plancton).

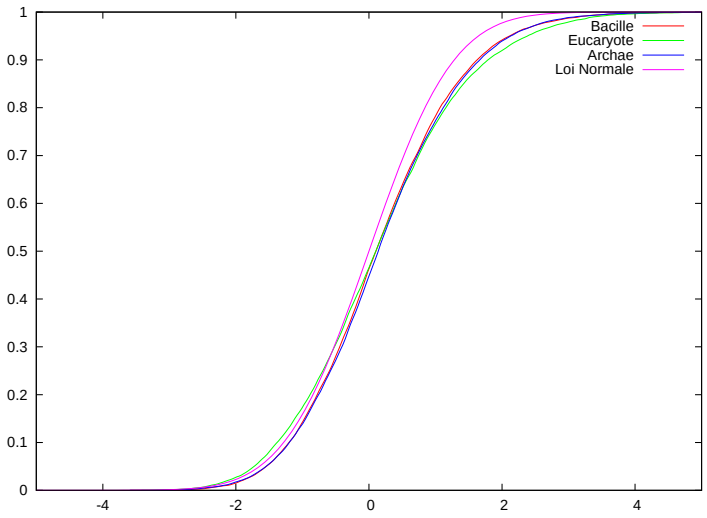


Figure 4.5 – Distributions des scores obtenus avec des séquences d'origines diverses.

L'étude des fonctions de cette famille de score a également été menée dans le contexte plus général des sources dynamiques (*cf.* [125] pour les définitions liées au sources dynamiques). Dans un travail ultérieur à celui présenté ici, il a été montré [15] que dans ce contexte, la moyenne et la variance ont une expression similaire à celle obtenue dans le cadre des sources sans mémoires. De surcroît, notre hypothèse concernant la normalité de la distribution des fonctions *Block-Based* dans le cas général a également été démontrée. La preuve utilise une décomposition de la variable aléatoire S_n en plusieurs variables aléatoires indépendantes et de même loi, puis se base sur le théorème de la limite centrale (*cf.* Nomenclature & Biographies) pour finir la preuve.

4.3 Algorithme

La famille de fonctions de score étudiée tout au long de la section précédente a été introduite dans [97], et utilisée dans l'algorithme STAMS. Dans cette seconde partie de chapitre, un nouvel algorithme, StatiSTAMS© [uses **S**tatistical **T**echniques for **A**nalysis & **R**esearch in **S**equences], est présenté, avec un triple objectif. Le premier étant d'obtenir des résultats au moins équivalents à ceux de STAMS (en termes de qualité). Le second consiste à obtenir ces résultats avec un temps d'exécution significativement plus rapide (de l'ordre de la seconde quand la première méthode est de l'ordre de la minute, ...). Le dernier objectif est d'élargir le champ d'action de l'algorithme, en répondant au problème plus général de l'extraction de motifs communs dans un ensemble de séquences (présenté à la section 2.5 – page 36),

[4] La courbe dont la partie supérieure est le plus à droite correspond à l'organisme eucaryote (*Arabidopsis thaliana*). La courbe la plus à gauche dans sa partie supérieure correspond à celle de la loi normale. Les deux dernières courbes (*Bacillus subtilis* et plancton – *archæ*) se confondent sur toute leur longueur. Leurs parties supérieures sont donc situées entre la courbe de la loi normale et celle de l'organisme eucaryote.

i.e., en intégrant une notion de quorum. La réalisation de ces objectifs est intrinsèquement liée aux connaissances apportées par les résultats de l'étude des fonctions *Block-Based*.

Avant de présenter l'algorithme, il est souhaitable de rappeler le problème général EM_q auquel il apporte une réponse :

Extraction sous contrainte de quorum de motifs communs dans un ensemble de séquences.

Données: un ensemble de séquences $\mathcal{S} = \{s_1, \ldots, s_t\}$ ($s_i \in \Sigma^*, 1 \leq i \leq t$), un quorum q ($0 \leq q \leq 100\%$), ainsi qu'une notion de similarité globale.

Problème: trouver toutes les collections de $R \geq \lceil q\,t \rceil$ motifs $(m_1 \in s'_1, \ldots, m_R \in s'_R)$ telles que $\{s'_1, \ldots, s'_R\} \subseteq \mathcal{S}$ ($s'_i \neq s'_j$) et que $m_1, \ldots, m_R$ soient similaires.

4.3.1 Étude préliminaires

L'idée directrice de l'algorithme est de tirer profit des propriétés statistiques des fonctions de score utilisées, et principalement les fonctions *Block-Based* dont les composantes de score sont d'ordre polynomial. Par conséquent, avant de détailler le fonctionnement de cette nouvelle méthode, il apparaît nécessaire de définir les propriétés statistiques qu'il est possible d'exploiter.

Pour chaque fonction de score *Block-Based*, telle que ses composantes soient d'ordre polynomial, la moyenne et la variance (*cf.* Théorèmes 4.11 – page 94 – et 4.13 – page 99) se calculent en un temps constant[5] en fonction des probabilités de « correspondances » (p_1) et de « non correspondances » (p_0) entre deux mots de même longueur n fixée. Ces fonctions suivent asymptotiquement une loi normale [15] (*cf.* Annexe E.1).

Propriété 4.14. *Étant donnée une variable aléatoire X suivant une loi normale $\mathcal{N}(\mu, \sigma)$, alors la variable aléatoire $\dfrac{X - \mu}{\sigma}$ suit également une loi normale de moyenne 0 et de variance 1.*

Cette propriété (*cf.* [36] pour sa preuve) permet de comparer entre elles des variables aléatoires différentes distribuées normalement. Cela peut être comparé à une mise à la même échelle de plusieurs cartes géographiques afin de pouvoir les superposer.

Définition 4.20. Étant donnée une variable aléatoire X suivant une loi normale $\mathcal{N}(\mu, \sigma)$, la variable aléatoire $\dfrac{X - \mu}{\sigma}$ est appelée la variable de X centrée par rapport à la moyenne et réduite par l'écart-type, ou en abrégé, la variable centrée réduite. Soit x une valeur de résultat pour X ; alors la valeur centrée réduite $\dfrac{x - \mu}{\sigma}$ est appelée Z-valeur.

Le terme de Z-score tire ses origines de cette notion de Z-valeur. Il est pourtant à manipuler avec précaution, comme l'illustre la remarque suivante.

Remarque 4.21. Étant donnée une valuation x d'un score d'espérance μ et d'écart-type σ, la dénomination Z-score représente la valeur centrée réduite $\dfrac{x - \mu}{\sigma}$ (quelle que soit la distribution de score). Ainsi, une Z-valeur est un Z-score, la réciproque est fausse.

[5] Pour les fonctions polynomiales, les fonctions $\mathcal{K}$ et $\mathscr{K}$ (*cf.* Équations 4.16 – page 96 – et 4.25 – page 103) permettent de donner des formules closes. Pour les fonctions sublinéaires, il est possible de définir une approximation en maîtrisant le terme d'erreur (*cf.* Équation 4.20 – page 98). Celui-ci converge à vitesse géométrique vers 0, et pour une précision fixée, le nombre de termes à calculer est majorable par une constante. Les constantes κ'_0, κ'_1, $\overline{\kappa'_0}$, $\overline{\kappa'_1}$, κ''_0 et κ''_1, le calcul de la moyenne et de la variance en fonction de n et des probabilités p_0 et p_1 s'obtient en temps constant.

Un des intérêts majeurs de l'utilisation des Z-valeurs est que cela simplifie le calcul des probabilités associées aux variables aléatoires normalement distribuées.

Propriété 4.15. *Étant donnée une variable aléatoire X suivant une loi normale $\mathcal{N}(\mu, \sigma)$, pour tout $x \in X(\Omega)$, la probabilité Prob$[X = x]$ est égale à la probabilité Prob* $\left[\dfrac{X-\mu}{\sigma} = \dfrac{x-\mu}{\sigma}\right]$.

Ainsi, calculer une probabilité liée à une variable aléatoire distribuée normalement, revient à déterminer la probabilité correspondante liée à la variable centrée réduite. Il suffit alors de disposer d'une table statique descriptive de la loi normale centrée réduite (*cf.* Table E.2 – page 214) pour effectuer le calcul en temps constant. Une autre notion importante, introduite dans [37] en 1943, qui est utilisée dans StatiSTAIRS est la notion de P-valeur.

Définition 4.22. Étant données une variable aléatoire X (quelle que soit sa distribution) et une valeur $x \in \mathbb{R}$, la probabilité Prob$[X \geq x]$ est également appelée la P-valeur de x (par rapport à X).

Définition 4.23. Étant données une variable aléatoire X distribuée normalement et une valeur seuil $x^* \in \mathbb{R}$, le réel α ($0 \leq \alpha \leq 1$) tel que $\alpha = 1 - \text{Prob}[X \leq x^*]$, est appelée erreur de type I ou encore erreur de première espèce. La valeur seuil centrée réduite est notée U_α.

De même qu'il existe une table pour déterminer la probabilité d'un évènement suivant une loi normale centrée réduite, il existe une table permettant de déterminer la valeur seuil U_α correspondant à une erreur de première espèce α donnée (*cf.* Table E.1 – page 213).

Il apparaît clairement que si l'application d'une fonction *Block-Based* S à un alignement de 2 mots de longueur n produit le score ς, la P-valeur de ς est Prob$[S_n \geq \varsigma]$, où S_n est la variable aléatoire associée à la fonction de score S. Comme S_n suit une loi normale [15] et que sa moyenne et sa variance (*cf.* Théorèmes 4.11 – page 94 – et 4.13 – page 99) sont connues et calculables en temps constant, il est possible de déterminer cette probabilité à partir du Z-score (qui ici est une Z-valeur), en utilisant la propriété 4.15. La P-valeur de ς est donc égale à

$$\begin{aligned}\text{Prob}[S_n \geq \varsigma] &= \text{Prob}\left[X \geq \frac{\varsigma - \mathbf{E}[Sn]}{\sqrt{\mathbf{Var}[S_n]}}\right] \\ &= 1 - \text{Prob}\left[X < \frac{\varsigma - \mathbf{E}[Sn]}{\sqrt{\mathbf{Var}[S_n]}}\right].\end{aligned}$$

En utilisant une table statique des probabilités Prob$[X < x]$ où X suit une loi $\mathcal{N}(0, 1)$ (*cf.* Table E.2 – page 214), il est possible de déterminer une valeur approchée de Prob$[S_n \geq \varsigma]$ en temps constant.

Remarque 4.24. La P-valeur associée à un score ς se lit comme « la probabilité qu'un score au moins égal à ς soit le fruit du hasard ».

4.3.2 Intégration des propriétés statistiques

La première idée qu'il est naturel de mettre en œuvre est d'utiliser le Z-score en lieu et place du score deux à deux dans STAIRS. Les tests réalisés n'ont eu d'impact notoire ni sur la qualité des résultats, ni sur le temps d'exécution. Ceci s'explique principalement par le fait que globalement, la relation d'ordre entre les solutions candidates ne s'en trouve pas (ou peu) modifiée (d'après les expérimentations). Les motifs consensuels (*i.e.*, modèles construits à partir de chaque solution pour les représenter) proposés

par cette variante sont souvent un peu plus petits (d'un ou deux symboles), ce qui a comme incidence d'augmenter légèrement (moins de 0,5%) la sensibilité moyenne, et de baisser tout aussi légèrement la valeur prédictive positive moyenne sur les séquences générées aléatoirement (*cf.* Section 3.4 – page 73).

À défaut d'améliorer la qualité des résultats, une seconde variante a été testée, visant à diminuer le temps de calcul. Afin de réaliser une économie de temps, la stratégie mise en place a consisté à diminuer l'espace de recherche en intégrant comme paramètre d'entrée une erreur de première espèce α, et en ne conservant que les paires candidates dont le score avait une P-valeur au plus égale à α. Les tests réalisés en fixant $\alpha = 5\%$ sont identiques en qualité à ceux de la première variante présentée ci-avant. Le gain en temps d'exécution n'est pas non plus significatif (environ 2% de gain de temps sur les séquences biologiques). En fixant l'erreur de première espèce plus petite, le gain de temps augmente peu, en revanche, la qualité des résultats en est très vite affectée.

STARS n'étant pas initialement conçu pour tenir compte d'une information sur la distribution des fonctions, il semble évident (au vu des résultats obtenus par ces adaptations) que le principe même de l'algorithme doit être complètement repensé. C'est pourquoi StatiSTARS n'est pas une version améliorée de STARS, mais bien une nouvelle méthode, offrant notamment la possibilité d'intégrer une contrainte de quorum.

Toutefois, dans la mesure où les résultats du premier algorithme sont tout de même très satisfaisants, plusieurs concepts de l'algorithme initial se retrouvent dans cette nouvelle méthode. Le cahier des charges établi pour le premier algorithme demeure le même (*cf.* Section 2.5.3 – page 42), à savoir une utilisation et une paramétrisation simple et intuitive, ainsi qu'une grande modularité dans le type de motifs à extraire.

Avant de détailler cette nouvelle méthode, il est nécessaire de succinctement rappeler le fonctionnement de STARS, et de mettre en exergue les implications dues à la méconnaissance des propriétés statistiques des fonctions de score qu'il utilise.

STARS est un algorithme probabiliste. Ceci est en grande partie dû au peu d'informations fournies par le score. En effet, le score ne permet dans cette méthode que de comparer les solutions entre elles au fil du déroulement de l'algorithme. La notion de score élevé ou faible n'apparaît alors qu'au fur et à mesure de l'exécution, d'où la nécessité d'effectuer plusieurs cycles afin de décider de l'importance d'un score donné. Comme le nombre de cycles qu'il est possible de réaliser est exponentiel, seul un sous-ensemble (choisi au hasard) de ces cycles est réellement effectué, conférant ainsi le caractère probabiliste de STARS. Le fait de disposer d'informations statistiques comme la moyenne et la variance sur les fonctions *Block-Based* permet de ne pas avoir à effectuer un grand nombre de cycles pour déterminer si le score est élevé ou pas. Donc cela apporte clairement un gain au niveau du temps d'exécution. De plus, l'apport des propriétés statistiques des fonctions de score rend l'algorithme non plus probabiliste, mais déterministe. Toutefois, une partie importante des concepts présents dans cette méthode sont repris dans StatiSTARS, et il apparaît judicieux de s'appuyer sur l'algorithme de STARS (*cf.* Algorithme 4.1 – page ci-contre) afin de la présenter.

L'approche de STARS consiste à considérer à tour de rôle chaque séquence comme étant une séquence de référence, supposée contenir toutes les solutions à extraire. Cette approche permet de considérer tous les motifs de la séquence de référence comme des modèles externes dont les occurrences sur les autres séquences sont recherchées. Cette solution (discutée à la section 3.3.1 – page 58) présente plusieurs avantages. Principalement, cela permet de définir un modèle consensuel représentatif d'une collection de motifs et de diminuer la complexité liée au calcul d'un score global à partir des scores deux à deux entre

Nom : STABS
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  $\Sigma, \mathcal{C}(\Sigma)$ % *Alphabet et couverture de l'alphabet utilisé.* %
  p % *Nombre de résultats à renvoyer.* %
  $f : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score.* %
Sortie : *Sol* % *Ensemble des résultats.* %
Variables : *OldSol* % *Ensemble des résultats avant le dernier cycle.* %
  ℓ % *Numéro de la séquence de référence* %
  k % *indice de la séquence en cours de traitement* %
  π % *Permutation sur* $\mathcal{S}$ ($s'i = s_{\pi_i}$). %
  nb_cycles % *Compteur de cycles.* %
  TStock % *Vecteur de taille* t *pour le pré−traitement des séquences.* %
  TShared % *Vecteur de taille* $2n-1$ *pour les décalages entre 2 séquences.* %
  TCand % *Matrice Carrée de taille* t *pour le traitement de TShared.* %
Début
  % *Pré−traitement des séquences.* %
  Pour ℓ de 1 à t **Faire**
    $TStock[\ell] \leftarrow$ Créer_Tableau_Stockage(s_ℓ). % *Algorithme* 3.3, *p.*64 %
  Fin Pour

  Pour ℓ de 1 à t **Faire**
    Pour k de 1 à t **Faire**
      Si $\ell \neq k$ **Alors**
        $TShared \leftarrow$ Créer_Tableau_Correspondances($s_\ell, TStock[k]$). % *Algorithme* 3.4, *p.*65 %
        $TCand[\ell, k] \leftarrow$ Créer_Tableau_Candidats($TShared$). % *Algorithme* 3.5, *p.*66 %
      Fin Si
    Fin Pour
  Fin Pour

  % *Processus d'extraction.* %
  $Sol \leftarrow \emptyset$, $OldSol \leftarrow \emptyset$.
  $\ell \leftarrow 0$, $nb\_cycles \leftarrow 0$.
  Répéter
    % *Choix de la séquence de référence, puis de la permutation.* %
    $\ell \leftarrow (\ell \mod t) + 1$.
    Choisir une permutation π de $\mathcal{S}$ telle que $s'_1 = s_\ell$.
    % *Calcul de la solution pour cette permutation.* %
    $Sol \leftarrow Sol \cup$ STABS-$\chi(\mathcal{S}, \pi, p, f)$. % *Algorithme* 3.2, *p.*63 %
    Trier *Sol* par scores $\searrow$ et ne conserver que les p meilleurs résultats.
    % *Mise à jour du nombre de cycles exempts de modification.* %
    Si $Sol \neq OldSol$ **Alors**
      $nb\_cycles \leftarrow 0$.
    Sinon
      $nb\_cycles \leftarrow nb\_cycles + 1$.
    Fin Si
  Tant Que ($nb\_cycles < t^2$)
  Afficher et **Retourner** *Sol*.
Fin

Algorithme 4.1 – Algorithme d'extraction de motifs STABS.

un motif consensuel et ses occurrences (*cf.* Section 3.2.2 – page 49). Cette approche est donc reprise dans StatiSTAMS. Aussi, la phase de pré-traitement des séquences, permettant d'optimiser l'extraction des solutions candidates (*cf.* Section 3.3.3 – page 62), est-elle conservée (lignes 17 à 28). Il en est de même pour la notion de similarité utilisée, à savoir la e-similarité (*cf.* Section 3.2.1 – page 46), dont les propriétés permettent d'une part le pré-traitement des séquences, et d'autre part l'usage de toutes les fonctions proposées à la section 3.2.2 – page 49. Celles-ci permettent d'évaluer le degré de similarité (et notamment *via* les fonctions *Block-Based*) entre deux motifs donnés. L'idée étant de fonder la méthode sur les propriétés statistiques des fonctions, seules les fonctions dont les P-valeurs (*cf.* Définition 4.22 – page 111) sont calculables sont considérées. Le calcul du score global d'une collection de motif dans la première méthode utilise les scores deux à deux entre le modèle consensuel (présent sur la séquence de référence) et ses occurrences sur les autres séquences. Cette approche en étoile (*cf.* Figure 2.2) est également reprise ici. Toutefois, dans STAMS, chaque séquence est considérée comme séquence de référence plusieurs fois, et ceci de manière cyclique (*i.e.*, tous les t cycles, t étant le nombre de séquences en entrée). Les séquences pré-traitées (le traitement dépendant de la séquence de référence) sont donc stockées dans une matrice $TCand$ telle que $TCand[\ell, k]$ représente l'ensemble des motifs e-similaires maximaux (*cf.* Définition 3.14 – page 56) entre la séquence de référence s_ℓ et la séquence s_k pour tous les décalages (*cf.* Définition 3.10 – page 54) entre ces deux séquences. Ici, chaque séquence n'étant considérée qu'une seule fois comme référence, il n'est pas besoin de conserver les tables $TCand[\ell, k]$, mais seulement la table pour la séquence de référence en cours. Le traitement doit alors être effectué entre les lignes 27 et 28 de l'algorithme 4.1 – page précédente. Ces modifications sont reportées sur l'algorithme 4.2 – page ci-contre.

Les propriétés statistiques utilisées par StatiSTAMS pour l'extraction des collections de motifs solutions du problème EM_q (*cf.* ligne 29 – page suivante) sont donc essentiellement les notions de Z-score et de P-valeur. La première différence fondamentale introduite par rapport à STAMS concerne la relation d'ordre entre les paires de motifs similaires. En effet, celle-ci n'est plus basée sur les valeurs des scores calculés, mais sur leurs Z-scores, et plus précisément sur leurs P-valeurs. Ce changement a pour conséquence immédiate que seules les paires de motifs candidates dont la mesure de similarité est significative (*i.e.*, ceux dont la P-valeur traduit un score exceptionnel[6]) sont conservés dans l'espace des solutions. La notion d'exceptionnel est définie par l'erreur de première espèce maximale autorisée (*cf.* Définition 4.23 – page 111). En effet, tout score ς, donné par une fonction de score suivant une loi $\mathcal{N}(\mu, \sigma)$, dont la P-valeur est inférieure ou égale à une erreur α donnée est tel que $\dfrac{\varsigma - \mu}{\sigma^2} \geq U_\alpha$.

Par conséquent, seules les paires de motifs ayant un Z-score supérieur à un seuil U_α sont traitées dans conservées dans les tables $TCand[k]$. Le seuil U_α est – par définition – tel que $\text{Prob}[x \leq U_\alpha] = 1 - \alpha$ et peut être approché en temps constant (à partir de la table E.1 – page 213).

Concernant le cas plus particulier des fonctions *Block-Based* dont les composantes sont d'ordre polynomial, afin de calculer $\mathbf{E}[S_n]$ et $\mathbf{Var}[S_n]$ pour une paire candidate de longueur n, il faudrait, en toute rigueur, effectuer le calcul des probabilités p_1 et p_0 pour la paire en question. Cela peut se faire en temps linéaire par rapport à n, ce qui ne change donc pas la complexité liée au calcul du score (comme pour STAMS, il est supposé que le calcul du score d'un couple de mots est linéaire en temps sur la taille des mots). Néanmoins, cela alourdit l'algorithme et le gain d'information n'est pas significatif.

Afin d'alléger les calculs, les probabilités p_1 et p_0 considérées sont respectivement les probabilités que deux symboles pris au hasard dans l'ensemble des séquences de $\mathcal{S}$ soient en « correspondance » ou

[6] Exceptionnel = peu fréquent.

Nom : StatiSTABS
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  $\Sigma, \mathcal{C}(\Sigma)$ % *Alphabet et couverture de l'alphabet utilisé.* %
  p % *Nombre de résultats à renvoyer.* %
  $f_n : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score distribuée normalement selon* $\mathcal{N}(\mu_n, \sigma_n)$. %
  α % *Erreur de première espèce.* %
  q % *Contrainte de quorum* $0 \leq q \leq 100$. %
Sortie : *Sol* % *Ensemble des résultats.* %
Variables : ℓ % *Numéro de la séquence de référence* %
  k % *indice de la séquence en cours de traitement* %
  TStock % *Vecteur de taille* t *pour le pré-traitement des séquences.* %
  TShared % *Vecteur de taille* $2n-1$ *pour les décalages entre 2 séquences.* %
  TCand % *Vecteur de taille* $2n-1$ *pour le traitement de TShared.* %
Début
  % *Pré-traitement des séquences.* %
  Pour ℓ de 1 à t **Faire**
    $TStock[\ell] \leftarrow$ Créer_Tableau_Stockage(s_ℓ). % *Algorithme* 3.3, *p.*64 %
  Fin Pour

  $Sol \leftarrow 0$
  Pour ℓ de 1 à t **Faire**
    Pour k de 1 à t **Faire**
      Si $\ell \neq k$ **Alors**
        $TShared \leftarrow$ Créer_Tableau_Correspondances$(s_\ell, TStock[k])$. % *Algorithme* 3.4, *p.*65 %
        $TCand[k] \leftarrow$ Créer_Tableau_Candidats$(TShared)$. % *Algorithme* 3.5, *p.*66 %
      Fin Si
    Fin Pour
    % *Processus d'extraction.* %
    $Sol \leftarrow Sol \cup \{$Solutions pour la séquence de référence $s_\ell.\}$
  Fin Pour
  Trier *Sol* par scores $\searrow$ et ne conserver que les p premiers éléments.
  Afficher et **Retourner** *Sol*.
Fin

Algorithme 4.2 – Ébauche de l'algorithme d'extraction de motifs StatiSTABS.

bien au contraire en « non correspondance ». La probabilité p_1 est alors la moyenne des probabilités de « correspondances » entre toutes les paires distinctes de séquences.

$$p_1 := \frac{2}{t\,(t-1)} \sum_{i=1}^{t} \sum_{j=i+1}^{t} \sum_{\alpha \in \Sigma} p_\alpha^{(i)}\, p_\alpha^{(j)}, \qquad\qquad p_0 := 1 - p_0,$$

où $p_\alpha^{(k)}$ représente la probabilité d'apparition du symbole α dans la séquence s_k.

Il est possible d'approximer raisonnablement ces valeurs en posant que pour toute séquence s_k, la

probabilité d'apparition du symbole α est p_α. La valeur p_1 est alors sensiblement égale à

$$\begin{aligned} p_1 &\simeq \frac{2}{t\,(t-1)} \sum_{i=1}^{t} \sum_{j=i+1}^{t} \sum_{\alpha\in\Sigma} p_\alpha^2, \\ &\simeq \frac{2}{t\,(t-1)} \sum_{i=1}^{t} (t-i) \sum_{\alpha\in\Sigma} p_\alpha^2, \\ &\simeq \frac{2}{t\,(t-1)} \left(t^2 - \frac{t\,(t-1)}{2}\right) \sum_{\alpha\in\Sigma} p_\alpha^2, \\ &\simeq \sum_{\alpha\in\Sigma} p_\alpha^2. \end{aligned}$$

4.3.3 Extraction des solutions

L'intégration d'une contrainte de quorum q implique que pour un motif de la séquence de référence s_ℓ ayant une occurrence dans au moins $\lceil q\,t \rceil$ séquences, il faut tenir compte du fait qu'il existe peut-être des séquences dans lesquelles ce motif n'a pas d'occurrence. Bien qu'élémentaire, cette remarque est capitale. En effet, à chaque cycle, STAMS présuppose qu'il existe une solution (*i.e.*,, un motif de la séquence de référence ayant au moins une occurrence dans chaque autre séquence). De plus, l'ordre de traitement des séquences a une influence nécessitant d'effectuer de nombreux essais (*cf.* Section 3.3.4 – page 67). Essayer d'intégrer une contrainte de quorum dans de telles conditions équivaudrait alors à lancer l'algorithme sur tous les sous-ensemble d'au moins $\lceil q\,t \rceil$ séquences de $\mathcal{S}$. Il apparaît clairement qu'il est préférable

1. que l'ordre de traitement des séquences n'ait pas ou peu d'influence ;
2. de ne pas présupposer de la présence d'au moins une occurrence des motifs dans chaque séquence.

L'approche choisie consiste à ne pas effectuer de comparaisons sur les motifs, mais sur les positions de chaque symbole des motifs. En effet, si w_1 et w_2 sont deux motifs distincts de s_ℓ, avec $w_1 \pitchfork w_2$, et tels que w_1 ait une occurrence sur une séquence s_{k_1} ($k_1 \neq \ell$) et que w_2 ait une occurrence sur une séquence s_{k_2} ($k_2 \neq \ell$), alors il est préférable de considérer différemment les positions de w_1 qui ne sont pas dans w_2, les positions de w_2 qui ne sont pas dans w_1, et les positions communes à w_1 et w_2. C'est pourquoi, à chaque position de la séquence de référence est associée la liste des motifs maximaux contenant cette position, et cela pour chaque alignement avec chaque autre séquence de l'ensemble $\mathcal{S}$. Cette information permet de déterminer, pour une position donnée, et par un simple comptage, le nombre de séquences ayant une occurrence d'un motif incluant ladite position.

À chaque position i de la séquence de référence s_ℓ est donc associée un vecteur T_i de taille t (où $t = |\mathcal{S}|$) ainsi qu'un compteur τ_i. Ainsi, pour chaque séquence $s_k \neq s_\ell$, $T_i[k]$ est la liste des triplets $([a;b], z, \delta)$, tels que $[a;b] \in TCand_{s_\ell}[k][\delta]$, et tel que le Z-score z de $S(s_\ell[a..b], s_k[a+\delta..b+\delta])$ est supérieur au seuil U_α ; le compteur τ_i correspond au nombre de listes non vides de T_i (pour des raisons algorithmiques, la liste correspondant au cas $s_k = s_\ell$ contient un seul élément : $([1..|s_\ell|], z, 0)$, où z est le Z-score de $S(s_\ell, s_\ell)$). Une condition nécessaire pour qu'un motif de s_ℓ ait une occurrence dans au moins $\lceil q\,t \rceil$ séquences de $\mathcal{S}$ est que :

Propriété 4.16. *Étant donné un motif $s_\ell[a..b]$ de la séquence de référence exceptionnellement présent dans au moins $\lceil q\,t \rceil$ séquences, toutes les positions $i \in [a;b]$ de s_ℓ sont telles que $\tau_i \geq \lceil q\,t \rceil$.*

Démonstration. Puisque $s_\ell[a..b]$ est exceptionnellement présent dans au moins $\lceil q\,t \rceil$ séquences, pour tout $i \in [a;b]$, il y a au moins $\lceil q\,t \rceil - 1$ listes de T_i qui contiennent un triplet de la forme $([a;b],\_,\_)$. Comme la liste de T_i correspondant à la séquence s_ℓ est supposée non nulle par convention, le compteur τ_i est bien au moins égal à $\lceil q\,t \rceil$. □

L'algorithme (*cf.* Algorithme 4.3 – page suivante) permettant de créer les tables T_i consiste à calculer, pour chaque table $TCand_{s_\ell}[k]$ donnée ($k \neq \ell$) et pour chaque motif de s_ℓ (représenté par un intervalle $[a;b]$) présent dans cette table, le Z-score associé à la paire similaire représentée par $[a;b]$. Si ce Z-score est supérieur au seuil U_α, la paire de motifs est alors référencée sous la forme d'un triplet (intervalle, Z-score et décalage) dans toutes les listes $T_i[k]$ telles que $a \leq i \leq b$. Le compteur τ_i est mis à jour dans le même temps.

Pour une séquence s_ℓ considérée comme référence, les tableaux de positions intéressantes sont illustrés par la figure 4.6.

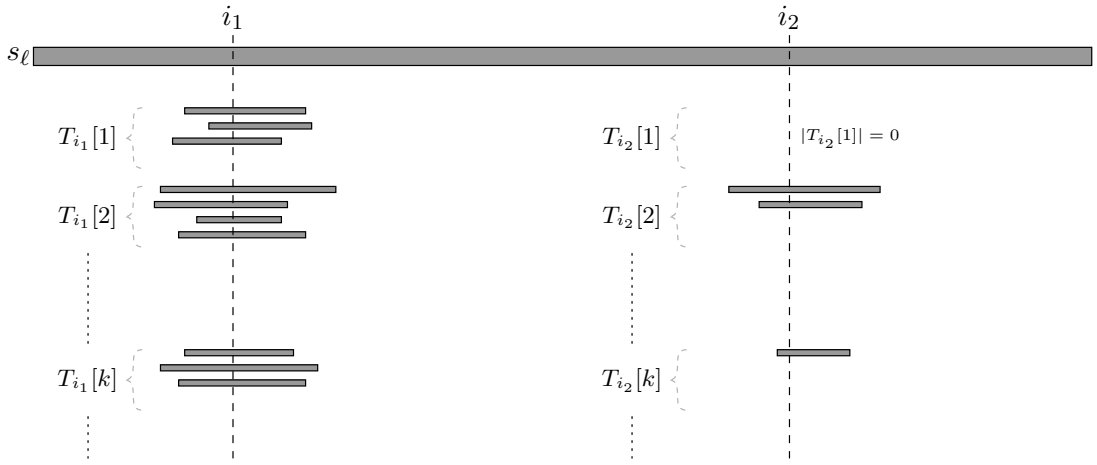


Figure 4.6 – Tableaux de positions intéressantes d'une séquence de référence.

Pour chaque position i de la séquence de référence s_ℓ, le compteur τ_i permet de savoir le nombre de séquences ayant au moins une occurrence d'un motif de s_ℓ passant par la position i. Sur la base de la propriété 4.16 – page ci-contre, seules les positions dont le compteur est supérieur à $\lceil q\,t \rceil$ sont considérées comme intéressantes ou valides. Étant données n positions valides consécutives $i_1, \ldots, i_n$, le motif $s_\ell[i_1..i_n]$ est présent dans les séquences s_k telles que l'intersection des listes $T_{i_1}[k], \ldots, T_{i_n}[k]$ est non nulle. Si au moins $\lceil q\,t \rceil$ listes résultant de ces intersections sont non nulles, alors il existe au moins une collection associée au motif $s_\ell[i_1..i_n]$ solution de EM_q. Comme pour STAMS, il n'est pas question de renvoyer toutes les collections de motifs. Il est donc nécessaire de valuer les solutions. Dans la première méthode, le score global associé à un motif consensuel est le minimum des scores entre les occurrences et le consensus. Ce score n'est pas repris ici, essentiellement pour deux raisons : tout d'abord, le score doit rendre compte – en raison de l'ajout de la gestion de la contrainte de quorum – du nombre de séquences concernées par le motif (*i.e.*, les séquences ayant au moins une occurrence dudit motif) ; d'autre part, il est très largement souhaitable que le score global puisse fournir une information statistique. Étant donnée une suite de Z-scores $v_1, \ldots, v_n$ de même loi (et de mêmes paramètres), le produit des compléments à 1 de leurs P-valeurs (*i.e.*, $\prod_{i=1}^{n}(1 - \text{Prob}[V_i > v_i]) = \prod_{i=1}^{n} \text{Prob}[V_i \leq v_i]$) est la probabilité d'obtenir une

```
Nom : Créer_Tableaux_Positions_Intéressantes
Entrées : TCand_{s_ℓ} % Table des ensemble de motifs e−similaires entre s_ℓ %
                      % et chaque autre séquence k de S = {s_1, ..., s_t}. %
  α % Erreur de première espèce permettant de déterminer U_α. %
  f_n : Σ* × Σ* → ℝ % Fonction de score distribuée normalement selon N(μ_n, σ_n). %
Sorties : T % Matrice de taille |s_ℓ| × t de listes de %
                % triplets (motif, Z−score, décalage). %
  τ % Tableau de |s_ℓ| compteurs pour la contrainte de quorum. %
Variables : δ % décalage entre deux séquences. %
  k % Numéro de la séquence en cours de traitement. %
  i % Positions concernées par un motif exceptionnel dans s. %
  a, b % Positions de début et fin des occurrences des motifs dans s. %
  Z % Z−score calculé entre deux motifs. %
Début

  Initialiser les compteurs τ[i] à 0.
  % Traitement des séquences s_k. %
  Pour k de 1 à t Faire
    Si k = ℓ Alors
      % Création des listes T_i[ℓ] et des compteurs τ_i[ℓ]. %
      Z ← (f_{|s_ℓ|}(s_ℓ, s_ℓ) − μ_{|s_ℓ|}) / σ_{|s_ℓ|}
      Pour i de 1 à |s_ℓ| Faire
        Initialiser T[i][ℓ] avec ([1..s_ℓ], Z, 0)
        τ[i] ← τ[i] + 1
      Fin Pour
    Sinon
      Initialiser les listes T[i][k] à vide.
      % Recherche des motifs exceptionnels entre s_ℓ et s_k. %
      Pour δ de 1 − |s_ℓ| à |s_k| − 1 Faire
        Pour Chaque [a; b] ∈ TCand_{s_ℓ}[k][δ] Faire
          Z ← (f_{b−a+1}(s_ℓ[a..b], s_k[a + δ..b + δ]) − μ_{b−a+1}) / σ_{b−a+1}
          Si Z > U_α Alors % Le score est significatif. %
            Pour i de a à b Faire
              % Mise à jour du compteur τ_i. %
              Si T[i][k] est vide Alors
                τ[i] ← τ[i] + 1
              Fin Si
              Ajouter ([a; b], Z, δ) à la liste T[i][k].
            Fin Pour
          Fin Si
        Fin Pour
      Fin Pour
    Fin Si
  Fin Pour

  Retourner (T, τ).

Fin
```

Algorithme 4.3 – Extraction des motifs exceptionnels entre deux séquences.

série de n scores au plus aussi élevés (dans le cas où les Z-scores sont produits par une source sans mémoire). Ainsi, il est possible de calculer la probabilité d'obtenir une série de n scores dont au moins un est plus élevé que dans la série $v_1, \ldots, v_n$, ce qui est donc une P-valeur (*cf.* Notation 4.22 – page 111). Toutefois, par souci d'homogénéité, il est préférable qu'un score élevé traduise une meilleure solution, aussi le score global choisi pour une collection de motifs de scores centrés réduits respectifs $Z_1, \ldots, Z_n$ est le complément à 1 de cette P-valeur (*cf.* Équation 4.26).

$$SG(Z_1, \ldots, Z_n) := 1 - \underbrace{(1 - \prod_{i=1}^{n} \text{Prob}[X \leq v_i])}_{p\text{-valeur de } (v_1,\ldots,v_n)} = \prod_{i=1}^{n} \text{Prob}[X \leq v_i]. \tag{4.26}$$

La reconstruction des solutions dans sa version naïve consiste à construire tous les motifs constitués de positions valides, à vérifier pour chacun qu'ils sont des solutions, et le cas échéant à calculer les Z-scores deux à deux des occurrences puis le score global, en ne conservant (comme pour la première méthode) que les p meilleures solutions. Il semble évident que le nombre de motifs à construire et à tester est très majoritairement exponentiel par rapport à la taille de la séquence de référence ($O\left(\sum_{i=1}^{|s|} \binom{|s|}{i}\right) = O\left(2^{|s|}\right)$). Il est donc nécessaire de limiter le nombre de motifs à construire. Pour cela, il faut déterminer

1. quels motifs construire en priorité ;
2. quand arrêter la phase de reconstruction.

Priorité des motifs à reconstruire :

Afin de déterminer les motifs à reconstruire en priorité, chaque position i de s_ℓ est valuée par un score $\mathcal{H}$, et les motifs sont reconstruits à partir des positions de meilleurs scores. Le score $\mathcal{H}$ doit posséder à la fois une dimension quantitative sur le nombre de solutions potentielles qu'il est possible de construire (le support), et une dimension qualitative sur le score global qu'il est raisonnable d'espérer obtenir pour une solution (estimateur du score global).

Définition 4.25. Étant données une position i de la séquence de référence s_ℓ et ses t listes T_i. Le support de la position i est égal au nombre de collections potentielles dont le consensus partage la position i sur s_ℓ. Le support est noté, et mathématiquement défini par

$$support(i) := \prod_{k=1}^{t} \max(1, |T_i[k]|). \tag{4.27}$$

La notion de support traduit le « potentiel » de solutions qu'il est possible de construire à partir d'une position donnée. La figure 4.7 reprend la figure 4.6 en la complétant afin d'illustrer la sémantique liée à cette notion. Sur l'exemple, il apparaît intuitivement que la position i_1 est plus « prometteuse » que la position i_2 en nombre de solutions.

Toutefois, il est possible que le score global de la meilleur solution obtenue à partir de la position i_1 soit inférieur au score global d'une des solutions obtenue à partir de la position i_2. Une estimation du score global des solutions obtenues à partir d'une position donnée dans la séquence de référence

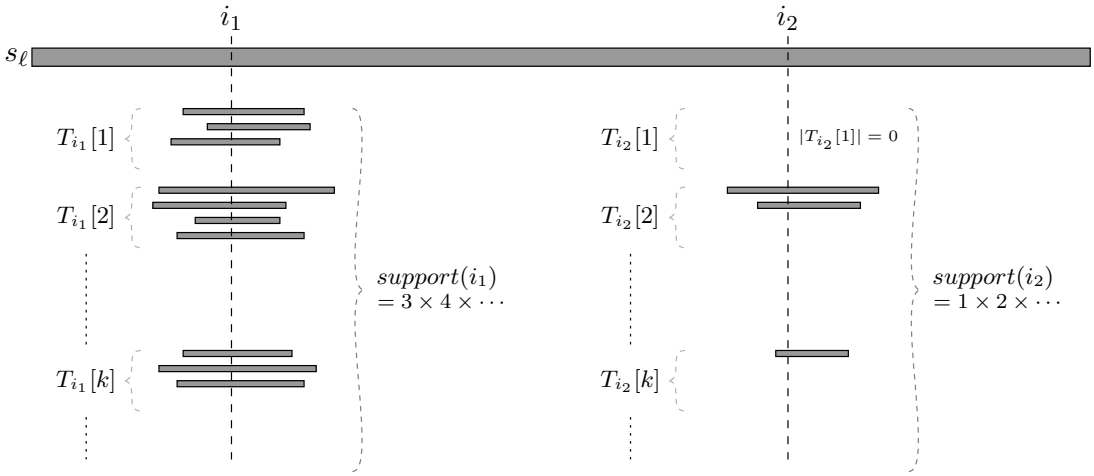


Figure 4.7 – Illustration de la notion de support pour plusieurs positions.

permet d'atténuer l'aspect quantitatif lié au support en intégrant une dimension qualitative à l'heuristique utilisée.

Pour une position valide i de la séquence de référence s_ℓ, une liste $T_i[k]$ représente l'ensemble de paires maximales de motifs similaires et de scores significatifs entre les séquences s_ℓ et s_k. Chaque triplet de la liste correspond donc à un motif (ou un sous-motif englobant la position i) de s_ℓ ayant une occurrence sur s_k pour un décalage donné participant à une solution au moins. Chacune de ces paires apporte donc une contribution au score global (*cf.* Équation 4.26 – page précédente) de ces solutions. En considérant, pour chaque liste $T_i[k]$ non nulle le Z-score le plus élevé (respectivement le plus bas) parmi ceux des triplets de $T_i[k]$, il est possible de définir une indication optimiste (respectivement pessimiste) sur la contribution apportée au score global d'une solution dont une occurrence est sur s_k. Ainsi, il est possible de définir deux estimateurs du score global des solutions obtenues à partir de la position i.

Définition 4.26. Étant donnée une position valide i de la séquence de référence, l'estimateur optimiste du score global des solutions obtenues à partir de la position i est défini par

$$SG^+(i) := \prod_{\substack{1 \leq k \leq t \\ T_i[k] \neq \emptyset}} \max_{(\_,Z,\_) \in T_i[k]} (Z).$$

De la même manière, l'estimateur pessimiste du score global des solutions obtenues à partir de la position i est défini par

$$SG^-(i) := \prod_{\substack{1 \leq k \leq t \\ T_i[k] \neq \emptyset}} \min_{(\_,Z,\_) \in T_i[k]} (Z).$$

Ainsi, en combinant les définitions 4.25 – page précédente – et 4.26, il est possible de définir un score $\mathcal{H}$ associé à chaque position valide i de la séquence de référence, mêlant les aspects quantitatifs et qualitatifs. Après plusieurs expérimentations, la solution retenue consiste à associer aux positions i le score

$$\mathcal{H}(i) := \frac{SG^+(i) + SG^-(i)}{2} \log(support(i)). \tag{4.28}$$

Arrêt de la phase de reconstruction :

Pour une position i donnée, l'ensemble des collections d'au moins $\lceil q\,t \rceil$ motifs e-similaires et de score significatif par rapport à un motif de la séquence de référence englobant la position i est un sous-ensemble des solutions au problème EM$_q$. Cet ensemble correspond aux collections qu'il est possible de reconstruire à partir des listes $T_i[k]$ ($1 \leq k \leq t$), définissant ainsi l'espace de construction. La limite inférieure de cet espace est donc représentée par le plus grand motif de s qui soit un sous-motif de tous les consensus qu'il est possible de définir. Quant à la limite supérieure, elle est représentée par le plus grand motif de s_ℓ contenant tous les consensus qu'il est possible de définir à partir d'un ensemble de motifs pris parmi les motifs référencés dans $\lceil q\,t \rceil$ listes non vides de T_i, dont la liste $T_i[\ell]$. La limite inférieure s'obtient simplement en réalisant l'intersection de tous les motifs de chaque liste non vide référencée par T_i. La limite supérieure s'obtient, quant à elle, en deux étapes. Dans un premier temps, pour chaque liste $T_i[k]$, la plus petite borne inférieure et la plus grande borne supérieure des intervalles référencés sont extraits. Ensuite, les limites inférieure et supérieure de l'espace de construction s'obtiennent en considérant respectivement la $\lceil q\,t \rceil^{ème}$ plus petite borne inférieure et la $\lceil q\,t \rceil^{ème}$ plus grande borne supérieure extraites. La limite inférieure de l'espace de construction est représentée par l'intervalle $[x_i; y_i]$ tandis que la limite supérieure est représentée par l'intervalle $[X_i; Y_i]$ (*cf.* Figure 4.8).

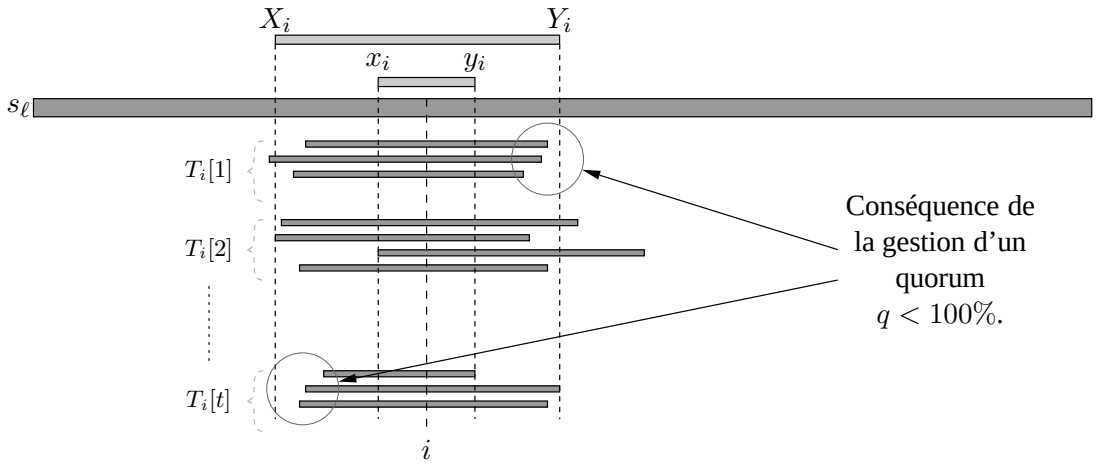


Figure 4.8 – Délimitation de l'espace de construction des solutions.

Une fois l'espace de construction autour de la position i délimité, il reste *a priori* à énumérer les motifs de s_ℓ, facteurs de $s_\ell[X_i..Y_i]$ dont $s_\ell[x_i..y_i]$ est un facteur[7]. Pour chacun de ces motifs, il faut ensuite vérifier qu'ils ont une occurrence sur au moins $\lceil q\,t \rceil$ séquences, et le cas échéant calculer le score de la solution obtenue. Toutefois, une telle énumération n'est pas envisageable en raison de l'explosion combinatoire du nombre de solutions potentielles. Aussi est-il nécessaire de définir une nouvelle heuristique permettant de limiter l'énumération des motifs. Cette heuristique est basée sur la définition de trois motifs particuliers dits caractéristiques de cet espace de construction.

Définition 4.27. Étant donnés une position i de la séquence de référence s_ℓ, un quorum q et les limites inférieures $[x_i; y_i]$ et supérieures $[X_i; Y_i]$ de l'espace de construction, le motif caractéristique central, est défini par le motif $\overleftrightarrow{m_i} := s_\ell[x_i..y_i]$, les motifs caractéristiques gauche et droit sont respectivement

[7] Par construction, $s_\ell[x_i..y_i]$ est facteur de tous les consensus des solutions qu'il est possible de construire à partir de la position i.

les plus grands motifs ayant une occurrence sur au moins $\lceil q\,t \rceil$ séquences, tels que $\overleftarrow{m_i} = s[X_i..Y'_i]$ et $\overrightarrow{m_i} = s[X'_i..Y_i]$, avec $Y'_i \geq y$ et $X'_i \leq x$.

Le choix de ces motifs caractéristiques est guidé par leurs propriétés particulières. En effet, le motif caractéristique central $\overleftrightarrow{m_i}$ est un sous-motif de tous les consensus des solutions qu'il est possible de construire à partir de la position i. Dans la suite, tous ces consensus sont appelés les i-consensus. Par construction, tout motif débutant avant l'indice X_i et terminant au plus tôt à la position i ne peut être un i-consensus. L'indice de début d'un i-consensus est minoré par X_i. Les motifs $s_\ell[X_i..i]$ et $s_\ell[i..y_i]$ étant des i-consensus, le motif $s_\ell[X_i..y_i]$ est également un i-consensus. De la même manière, il n'existe pas de i-consensus se terminant après la position Y_i. Ainsi, il existe nécessairement un i-consensus débutant à la position X_i et s'achevant à la position Y'_i telle que $y_i \leq Y'_i \leq Y_i$. De façon symétrique, il existe nécessairement un i-consensus s'achevant à la position Y_i et débutant à la position X'_i telle que $X_i \leq X'_i \leq x_i$ (*cf.* Figure 4.9).

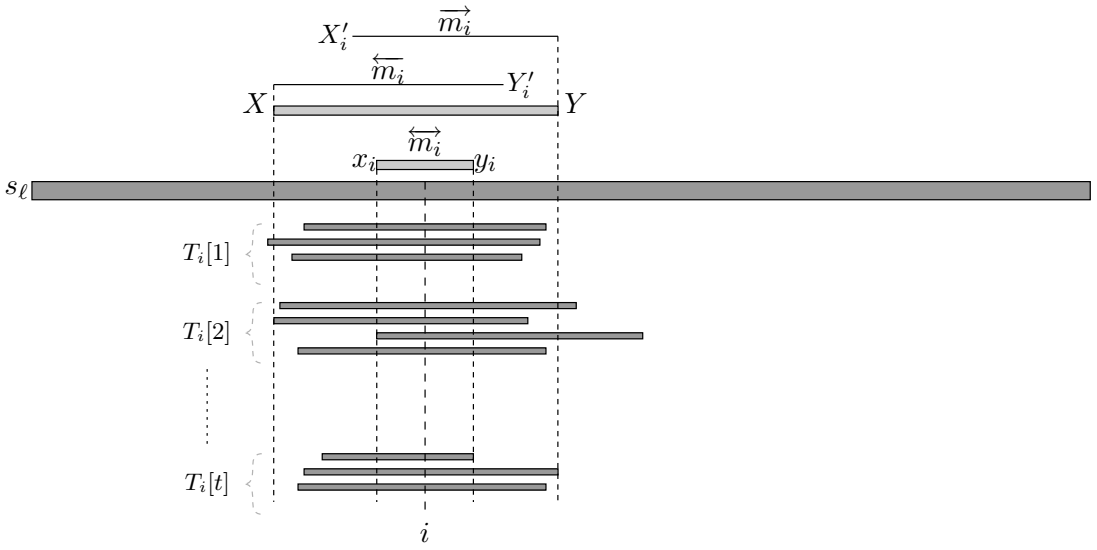


Figure 4.9 – Motifs caractéristiques de l'espace de construction autour de la position i.

La recherche de $\overleftarrow{m_i}$ (respectivement $\overrightarrow{m_i}$) revient donc à trouver la plus grande valeur de Y'_i (respectivement la plus petite valeur de X'_i) permettant de satisfaire la contrainte de quorum (la contrainte de quorum est satisfaite pour un motif $s_\ell[a..b]$ s'il existe au moins $\lceil q\,t \rceil$ listes $T_i[k]$ telles que $[a..b]$ soit inclus dans un des intervalles référencés – la liste $T_i[\ell]$ satisfait nécessairement cette condition). Les recherches de X'_i et de Y'_i s'effectuent par dichotomie sur les intervalles respectifs $[X_i; x_i]$ et $[y_i; Y_i]$.

Si l'énumération de tous les i-consensus n'est pas envisageable, il n'est pas non plus raisonnable de se contenter de ces trois motifs caractéristiques. Le critère de choix retenu est de ne conserver que les p plus grand i-consensus. Pour cela, l'heuristique consiste à énumérer, à partir des motifs caractéristiques gauche et droit, les i-consensus les plus grands. Ces i-consensus, appelées motifs caractéristiques secondaires, s'obtiennent en suivant un principe de décalage/extension des motifs caractéristiques gauche et droit (*cf.* Algorithme 4.4 – page 124). En effet, le motif $\overleftarrow{m_i} = s_\ell[X_i..Y'_i]$ est décalé position après position vers la droite pour obtenir les motifs $s_\ell[X_i + 1..Y'_i + 1]$, $s_\ell[X_i + 2..Y'_i + 2]$, …, $s_\ell[X_i + \min(x_i - X_i, Y_i - Y'i)..Y'_i + \min(x_i - X_i, Yi - Y'_i)]$. Chacun de ces motifs est testé en vue de déterminer s'ils possèdent une occurrence sur au moins $\lceil q\,t \rceil$ séquences, et le cas échéant, les motifs sont éten-

dus (par dichotomie sur l'intervalle $[Y'_i; y_i]$) au maximum sur la droite. De la même manière, le motif $\overrightarrow{m_i} = s_\ell[X'_i..Y_i]$ est décalé position après position vers la gauche pour obtenir les motifs $s_\ell[X'_i-1..Y_i-1]$, $s_\ell[X'_i-2..Y_i-2]$, ..., $s_\ell[X'_i-\min(X'_i-X_i, Y_i-y_i)..Y_i-\min(X'_i-X_i, Y_i-y_i)]$. Tous ces motifs sont également testés afin de déterminer s'ils sont des i-consensus, et le cas échéant, ils sont étendus au maximum sur la gauche.

Une fois l'énumération effectuée, au plus p motifs consensuels sont conservés (les p plus grands motifs), et les positions (valides) de l'espace de construction sont exclues d'un traitement ultérieur. En effet, la plupart de ces positions, bien qu'ayant souvent un score $\mathcal{H}$ élevé (d'après les expérimentations), n'aboutissent généralement qu'à reconstruire plus ou moins les mêmes consensus, ce qui par ailleurs explique leur score élevé. La reconstruction s'arrête lorsque au plus p positions valides restantes ont été traitées. En effet, pour chaque position traitée, au moins un motif consensuel existe, celui représenté par l'intervalle $[x; y]$. L'heuristique basée sur $\mathcal{H}$ permet de surcroît de guider la reconstruction vers des motifs consensuels significatifs en pratique, ce qui permet de mettre un terme à cette phase rapidement. De plus, comme chaque séquence est considérée une fois comme séquence de référence, le nombre de solutions obtenues en ne considérant systématiquement que p positions par séquence est déjà suffisamment important (*i.e.*, $p\,t$) pour renvoyer p motifs consensuels très exceptionnels (au sens statistique) sur la globalité de l'exécution (*cf.* Algorithme 4.5 – page 125).

Une fois que les motifs consensuels sont extraits d'une séquence de référence donnée, il reste à calculer pour chacun leurs occurrences sur les différentes séquences, ainsi que le score global associé à la collection (*cf.* Algorithme 4.6 – page 126). Pour un consensus $s_\ell[a..b]$ donné, ses différentes occurrences s'obtiennent en parcourant l'un des vecteurs de listes T_i tel que $i \in [a; b]$ (*cf.* Propriété 4.16 – page 116). En effet, pour chaque séquence s_k, un triplet (m, Z, δ) de $T_i[k]$ tel que l'intervalle $[a; b]$ est inclus dans m indique que $s_k[a+\delta..b+\delta]$ est une occurrence de $s_\ell[a..b]$. Il suffit alors de calculer le Z-score des paires $(consensus, occurrence)$, et de ne conserver pour chaque séquence que l'occurrence renvoyant le meilleur score. Une fois la collection de motifs définie, il rester à calculer la P-valeur associée à chaque Z-score afin de déterminer le score global SG (*cf.* Équation 4.26 – page 119).

Afin d'optimiser cette phase de calcul, la liste T_i utilisée est celle qui limitera le plus le nombre de collections potentielles, à savoir la liste associée à la position i de plus petit support (*cf.* Définition 4.25 – page 119).

L'extraction des solutions n'a pas été détaillée dans l'algorithme 4.2 – page 115 (ligne 29). Avant d'étudier la complexité de StatiSTAMS, il est donc souhaitable de compléter l'algorithme en y intégrant ce qui vient d'être présenté dans cette section (*cf.* Algorithme 4.7 – page 127).

4.3.4 Complexité

Avant d'entamer l'étude de la complexité, l'algorithme 4.2 – page 115 – est repris en modifiant la ligne 29 afin d'y intégrer l'algorithme 4.7 – page 127.

Pré-traitement des séquences (ligne 16 à 18 et 22 à 27)

La construction des tables $TStock$, $TShared$ et $TCand$ a été présentée à la section 3.3.5 – page 68. La complexité temporelle et spatiale de la construction de la table $TStock$ (lignes 16 à 18) est donc

$$O\left(|\Sigma|\, t\, n\right), \tag{4.29}$$

Nom : Reconstitution_Motifs_Consensuels
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  ℓ % *Numéro de la séquence de référence.* %
  T_i % *Vecteur de taille* t *de listes de triplets* %
    % *(motif, Z–score, décalage).* %
  q % *Contrainte de quorum à satisfaire.* %
  $[x;y], [X;Y]$ % *Limites inf. et sup. de l'espace de construction.* %
Sorties : $iMotifs$ % *Ensemble des* i*–consensus reconstruits.* %
Variables :
  X', X'', Y', Y'' % *Bornes pour les motifs caractéristiques.* %
  d % *Décalage appliqué aux motifs caractéristiques gauche et droit.* %
  r % *Extension gauche ou droite à appliquer en sus du décalage.* %
Début

  % *Construction des motifs caractéristiques gauche et droit.* %
  Calculer Y' maximal par dichotomie sur $[y;Y]$ (via T_i)
    tel que $s_\ell[X..Y']$ soit un i–consensus.
  Calculer X' minimal par dichotomie sur $[X;x]$ (via T_i)
    tel que $s_\ell[X'..Y]$ soit un i–consensus.
  $iMotifs \leftarrow \{[X;Y'], [X';Y]\}$.

  % *Construction des motifs caractéristiques secondaires.* %
  $r \leftarrow 0, d \leftarrow 1$
  Tant Que $d \leq \min(x - X, Y - Y' - r)$ **Faire**
    Si $s_\ell[X + d; Y' + d + r]$ est un i–consensus **Alors**
      Calculer Y'' maximal par dichotomie sur $[Y' + d + r; Y]$ (via T_i)
        tel que $s_\ell[X + d..Y'']$ soit un i–consensus.
      $iMotifs \leftarrow iMotifs \cup \{[X + d; Y'']\}$
      $r \leftarrow Y'' - Y' - d$
    Fin Si
    $d \leftarrow d + 1$
  Fin Tant Que

  $r \leftarrow 0, d \leftarrow 1$
  Tant Que $d \leq \min(X' - X - r, Y - y)$ **Faire**
    Si $s_\ell[X' - d - r; Y - d]$ est un i–consensus **Alors**
      Calculer X'' minimal par dichotomie sur $[X; X' - d - r]$ (via T_i)
        tel que $s_\ell[X''..Y - d]$ soit un i–consensus.
      $iMotifs \leftarrow iMotifs \cup \{[X''; Y - d]\}$
      $r \leftarrow X' - X'' + d$
    Fin Si
    $d \leftarrow d + 1$
  Fin Tant Que

  Retourner $iMotifs$.

Fin

Algorithme 4.4 – Reconstitution des i-consensus de la séquence de référence.

```
1  Nom : Calcul_Consensus_Séquence
2  Entrées : S = {s_1, ..., s_t} % Ensemble des séquences à traiter. %
3    ℓ % Numéro de la séquence de référence. %
4    H % Scores associés aux positions de la séquence s_ℓ. %
5    p % Nombre de positions de la séquence s_ℓ à examiner. %
6    q % Contrainte de quorum à satisfaire. %
7    T % Matrice de taille |s_ℓ| × t de listes de %
8      % triplets (motif, Z−score, décalage). %
9  Sortie : Res % Ensemble des consensus construits pour la séquence s_ℓ. %
10 Variables : i % Positions de s_ℓ. %
11   [x;y], [X;Y] % Limites inf. et sup. de l'espace de construction courant. %
12   τ % Tableau de |s_ℓ| compteurs pour la contrainte de quorum. %
13   L % Positions de s_ℓ restant à traiter. %
14 Début
15
16   L ← [1; |s_ℓ|].
17   Tant Que (p > 0) ∧ (L ≠ ∅) Faire
18     Choisir i ∈ L tel que i maximise H[i].
19     Si H[i] > 0 Alors
20       % Délimitation de l'espace de construction autour de la position i. %
21       Calculer les bornes [x;y], [X;Y] à partir de T[i] et de q.
22       % Calcul des p meilleurs i−consensus (Algorithme 4.4, p.124). %
23       Res ← Res ∪ Reconstitution_Motif_Consensuel(S, ℓ, T[i], q, [x;y], [X;Y]).
24       L ← L \ [X;Y].
25       p ← p − 1.
26     Sinon
27       p ← 0.
28     Fin Si
29   Fin Tant Que
30
31   Trier Res par taille d'intervalles décroissantes,
32   Res ← {I ∈ Res | ∀ I' ∈ Res, I ⊄ I'}.
33   Ne conserver que les p premiers éléments de Res.
34
35   Retourner Res.
36
37 Fin
```

Algorithme 4.5 – Calcul des consensus pour une séquence de référence fixée.

```
1  Nom : Calcul_Collection
2  Entrées : T_i % Tableau de taille t de listes de triplets %
3            % (motifs, décalages, Z−score). %
4    ℓ % Numéro de la séquence de référence. %
5    α % Erreur de première espèce permettant de déterminer U_α. %
6    f_n : Σ* × Σ* → ℝ % Fonction de score distribuée normalement selon N(μ_n, σ_n). %
7  Sorties : SG % Score global de la solution. %
8    D % Tableau de taille t des décalages associés aux occurrences. %
9  Variables :
10   k % Numéro de la séquence en cours d'analyse. %
11   Z, tmp % Z−scores. %
12   m, δ % Motif et décalage associé. %
13 Début
14
15   SG ← −1.
16   Pour k de 1 à t Faire
17     D[k] ← −∞, Z ← U_α.
18     Pour Chaque (m, δ, _) de T_i[k] Faire
19       Si [a; b] ⊆ m Alors
20         tmp ← (f_{b−a+1}(s_ℓ[a..b], s_k[a + δ..b + δ]) − μ_{b−a+1})/σ_{b−a+1}
21         Si Z < tmp Alors
22           D[i] ← δ
23           Z ← tmp
24         Fin Si
25       Fin Si
26     Fin Pour
27
28     Si Z > U_α Alors
29       Si SG < 0 Alors
30         SG ← Prob[X ≤ Z]
31       Sinon
32         SG ← SG ∗ Prob[X ≤ Z]
33       Fin Si
34     Fin Si
35   Fin Pour
36
37   Retourner (SG, D).
38
39 Fin
```

Algorithme 4.6 – Calcul des occurrences d'un consensus et du score global de la collection.

Nom : Extraction_Solutions
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  ℓ % *Numéro de la séquence de référence.* %
  p % *Nombre de résultats à renvoyer.* %
  $f_n : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score distribuée normalement selon* $\mathcal{N}(\mu_n, \sigma_n)$. %
  α % *Erreur de première espèce.* %
  q % *Contrainte de quorum* $0 \leq q \leq 100$. %
Sortie : Sol % *Ensemble des résultats obtenus à partir de* s_ℓ. %
Variables : $Motifs$ % *Ensemble de consensus.* %
  i, j % *positions de la séquence de référence.* %
  $TCand_{s_\ell}$ % *Table des ensemble de motifs* $e-$*similaires entre* s_ℓ %
    % *et chaque autre séquence* k *de* $\mathcal{S} = \{s_1, \ldots, s_t\}$. %
  $\mathcal{H}, support$ % *Vecteurs des scores et des supports de chaque position.* %
  T, τ % *Tableaux d'informations sur les positions de* s_ℓ. %
  $[a; b]$ % *Intervalle représentant un consensus.* %
  SG, D % *Score global et occurrences (décalages associés) d'une solution.* %
Début

  % *Calcul des consensus.* %
  $(T, \tau) \leftarrow$ Créer_Tableaux_Positions_Intéressantes$(TCand, \alpha, f_n)$. % *Algorithme* 4.3, *p.*118 %
  Pour i de 1 à $|s_\ell|$ **Faire**
    Si $\tau[i] \geq \lceil q\,t \rceil$ **Alors**
      Calculer $support[i]$ et $\mathcal{H}[i]$
    Sinon
      $support[i] \leftarrow 0, \mathcal{H}[i] \leftarrow 0$
    Fin Si
  Fin Pour
  $Motifs \leftarrow$ Calcul_Consensus_Séquence$(\mathcal{S}, \ell, \mathcal{H}, p, q, T)$ % *Algorithme* 4.5, *p.*125 %

  % *Calcul des solutions.* %
  $Sol \leftarrow \emptyset$
  Pour Chaque $[a; b] \in Motifs$ **Faire**
    Choisir $i \in [a; b]$ tel que $support[i] = \min_{a \leq j \leq b} (support[j])$.
    $(SG, D) \leftarrow$ Calcul_collection$(T[i], \ell, \alpha, f_n)$ % *Algorithme* 4.6, *p.*126 %
    $Sol \leftarrow Sol \cup ([a; b], \ell, SG, D)$.
  Fin Pour

  Retourner Sol

Fin

Algorithme 4.7 – Extraction des solutions pour une séquence de référence fixée.

```
1  Nom : StatiSTARS
2  Entrées : S = {s_1, ..., s_t} % Ensemble des séquences à traiter. %
3    Σ, C(Σ) % Alphabet et couverture de l'alphabet utilisé. %
4    p % Nombre de résultats à renvoyer. %
5    f_n : Σ* × Σ* → ℝ % Fonction de score distribuée normalement selon N(μ_n, σ_n). %
6    α % Erreur de première espèce. %
7    q % Contrainte de quorum 0 ≤ q ≤ 100. %
8  Sortie : Sol % Ensemble des résultats. %
9  Variables : ℓ % Numéro de la séquence de référence %
10   k % indice de la séquence en cours de traitement %
11   TStock % Vecteur de taille t pour le pré−traitement des séquences. %
12   TShared % Vecteur de taille 2n−1 pour les décalages entre 2 séquences. %
13   TCand % Vecteur de taille 2n−1 pour le traitement de TShared. %
14 Début
15   % Pré−traitement des séquences. %
16   Pour ℓ de 1 à t Faire
17     TStock[ℓ] ← Créer_Tableau_Stockage(s_ℓ). % Algorithme 3.3, p.64 %
18   Fin Pour
19
20   Sol ← 0
21   Pour ℓ de 1 à t Faire
22     Pour k de 1 à t Faire
23       Si ℓ ≠ k Alors
24         TShared ← Créer_Tableau_Correspondances(s_ℓ, TStock[k]). % Algorithme 3.4, p.65 %
25         TCand[k] ← Créer_Tableau_Candidats(TShared). % Algorithme 3.5, p.66 %
26       Fin Si
27     Fin Pour
28     % Processus d'extraction. %
29     Sol ← Sol ∪ Extraction_Solutions(S, ℓ, p, f_n, α, q). % Algorithme 4.7, p.127 %
30   Fin Pour
31   Trier Sol par scores SG ↘ et ne conserver que les p premiers éléments.
32   Afficher et Retourner Sol.
33 Fin
```

Algorithme 4.8 – Algorithme d'extraction de motifs StatiSTARS.

où t désigne le nombre de séquences de $\mathcal{S}$ et n leur longueur moyenne. Ces conventions sont adoptées dans les calculs suivants. La complexité engendrée par les lignes 22 à 27 est

$$O\left(t\,n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right), \tag{4.30}$$

tant en espace qu'en temps. Il n'est pas superflus toutefois de rappeler (pour la suite des calculs) que la longueur d'une liste $TCand_{s_\ell}[k][\delta]$ est en

$$O\left(\min\left(\frac{n}{e}, n\sum_{\alpha\in\Sigma} p_\alpha^2\right)\right),$$

et que la taille d'une table $TCand_{s_\ell}[k]$ est $2\,n-1$.

Extraction des solutions (ligne 29)

La complexité de cette étape est donnée par la complexité de l'algorithme 4.7 – page 127.

Calcul de T et de τ (ligne 20) :

L'initialisation des compteurs $\tau[i]$ (*cf.* Algorithme 4.3 – page 118, ligne 16) se fait trivialement en $O(|s_\ell|) = O(n)$. La boucle principale (lignes 18 à 44) implique exactement t itérations, dont une seule satisfaisant la condition $k = \ell$. Considérons ce cas unique. Le calcul du Z-score s'effectue en temps constant à partir du score. Celui-ci est calculé en $O(n)$. Le temps nécessaire à l'exécution de la boucle des lignes 22 à 25 est également en $O(n)$. Dans tous les autres cas (*i.e.*, $k \neq \ell$), il est nécessaire d'initialiser les listes $T[i][k]$ à vide, donc à effectuer $O(n)$ affectations. La boucle de la ligne 29 à la ligne 41 permet de tester pour chaque intervalle $[a; b]$ de $TCand_{s_\ell}[k][\delta]$, et cela pour chaque décalage δ (il y en a $2\,n-1$), si la paire $(s_\ell[a..b], s_k[a+\delta..b+\delta])$ est candidate. Le Z-score d'une telle paire se calcule en $O(b-a)$. Les éléments d'une liste $TCand_{s_\ell}[k][\delta]$ étant disjoints, il sont inclus dans l'intervalle $[1; |s_\ell|]$, le calcul de tous les Z-scores de toutes les paires obtenues à partir d'une liste $TCand_{s_\ell}[k][\delta]$ requiert un temps $O(n)$. De même, la mise à jour de la liste $T[i][k]$ et du compteur $\tau[i]$ s'effectuant en temps constant, le coût engendré par les passages dans la boucle de la ligne 33 à la ligne 39 pour une liste $TCand_{s_\ell}[k][\delta]$ donnée est en $O(n)$. Ainsi, le coût de la boucle de la ligne 29 à la ligne 42 est en $O((2\,n-1)\,n) = O(n^2)$. La construction de T et τ pour une séquence de référence donnée requiert donc un temps

$$O(n + (t-1)\,n^2) = O(t\,n^2). \tag{4.31}$$

L'espace requis pour le vecteur τ est en $O(n)$, tandis que celui requis par T est proportionnel à l'espace requis par la table $TCand_{s_\ell}$ multiplié par n (un motif $[a; b]$ étant référencé dans $b - a + 1$ listes). Soit une complexité spatiale pour la construction de T et τ (*cf.* Équation 4.29 – page 123) en

$$O\left(t\,n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right), \tag{4.32}$$

Calcul des supports et des scores $\mathcal{H}$ de chaque position (lignes 21 à 27) :

Le calcul de $\mathcal{H}[i]$ requiert de parcourir chaque liste $T[i][k]$ afin d'extraire les scores minimum et maximum. Le calcul de la longueur de la liste $T[i][k]$ peut être fait en parallèle. Ainsi, la complexité de cette étape correspond en temps pour chaque i à la complexité spatiale de $T[i]$. Donc pour l'ensemble des calculs de $\mathcal{H}$ et de $support$, la complexité temporelle est dans $O(|T|)$, soit

$$O\left(t\,n^2 \sum_{\alpha\in\Sigma} p_\alpha^2\right). \tag{4.33}$$

Calcul des consensus (ligne 28) :

Pour l'étude de la complexité de l'algorithme 4.5 – page 125, nous poserons que le choix de la position i qui maximise le score $\mathcal{H}$ (ligne 18) s'effectue au pire en temps linéaire sur la taille de la séquence de référence.

Le calcul de la limite inférieure de l'espace de construction autour d'une position i (ligne 21) requiert un temps de calcul linéaire par rapport au nombre d'éléments référencés par le vecteur $T[i]$ passé en

entrée de l'algorithme (intersection des éléments) et un espace constant. Le calcul de la limite supérieure de l'espace de construction autour d'une position i (ligne 21) nécessite également de parcourir toutes les listes $T[i][k]$ afin d'extraire les bornes minimum et maximum pour chaque liste, puis de trier l'ensemble des bornes minimum et l'ensemble des bornes maximum afin d'extraire leur $\lceil q\,t \rceil$ éléments (impliquant une complexité spatiale en $O(t)$). La taille de chaque liste $T[i][k]$ est bornée par le nombre d'intervalles incluant la position i dans la table $TCand_{s_\ell}[k]$. Or il ne peut y en avoir au plus qu'un seul par décalage. Donc $|T[i][k]| = O(|TCand_{s_\ell}[k]|) = O(n)$. Les algorithmes de tri utilisés sont des variantes du tri par tas et s'effectuent chacun en $O(t\,\log q\,t)$. Par conséquent, le calcul des limites de l'espace de construction se fait en $O(t\,(n + \log q\,t))$.

Il reste principalement à déterminer la complexité temporelle et spatiale requise par la construction des i-consensus (ligne 23). Pour une position i de s_ℓ, les i-consensus sont construits par l'algorithme 4.4 – page 124. Le temps de construction des motifs caractéristiques dépend des limites inférieures et supérieures. Dans le pire des cas, la limite inférieure est $[i;i]$ et la limite supérieure est $[1;n]$. Afin de vérifier si la contrainte de quorum est satisfaite pour un motif de s_ℓ et ses occurrences, il suffit de parcourir les t listes $T[i][k]$. Cette vérification implique donc un temps en $O(\sum_{i=1}^{t} |T[i][k]|) = O(t\,n)$. la maximisation par dichotomie (au pire $O(\log n)$ tests) à droite comme à gauche requière alors un temps en $O(t\,n\,\log n)$. Enfin, le nombre de motifs caractéristiques secondaires testés et étendus par les boucles des lignes 24 à 32 et des lignes 35 à 43 est majoré par n (cas où $r = 0$, $[x_i;y_i] = [i;i]$ et $[X_i;Y_i] = [1;n]$). Pour chacun d'eux, il y a une phase d'extension. La complexité temporelle totale de construction des consensus pour une position i donnée est donc $O(t\,n^2\,\log n)$.

La soustraction des positions déjà traitées (*cf.* Algorithme 4.5 – page 125, ligne 24) se fait en trivialement en $O(n)$. Enfin, le nombre d'itérations de la boucle (lignes 17 à 29) est borné par p, ce qui permet d'établir que la complexité temporelle du calcul des consensus est

$$\begin{aligned} & O\left(p\,\left(n + t\,(n + \log q\,t) + t\,n^2\,\log n + n\right)\right), \\ \text{soit} \quad & O\left(p\,t\,\left(\log q\,t + n^2\,\log n\right)\right). \end{aligned}$$

À cette complexité, il faut ajouter le coût engendré par les étapes de la ligne 31 à la ligne 33. Ces trois étapes s'effectuent en pratique simultanément. En effet, la suppression de solutions (ligne 32) peut se faire pendant le tri (partiel par tas). Ainsi, lorsque qu'une solution est sélectionnée pour être « rangée à sa place » durant le tri, il suffit de vérifier qu'elle n'est pas incluse dans les solutions déjà ordonnées (ce test requiert pour chaque solution conservée un temps en $O(p)$). Ainsi en plus du coût du tri partiel en $O(|Res| + p\,\log|Res|)$, il faut ajouter $O(p^2)$. Or le nombre de consensus construits à chaque itération est borné par n. Donc à l'issue de la construction des consensus, $|Res| = O(p\,n)$.

La complexité temporelle de cette phase demeure par conséquent

$$O\left(p\,t\,\left(\log q\,t + n^2\,\log n\right)\right). \tag{4.34}$$

La complexité spatiale engendrée par cette étape est de l'ordre du nombre de consensus construits et résidant en mémoire en même temps, soit

$$O(p\,n). \tag{4.35}$$

Calcul des collections et de leur score (lignes 32 à 36).

Le nombre de consensus de l'ensemble $Motifs$ est borné par p, et pour chacun, leur taille est au plus n. Ainsi, Pour un consensus $s_\ell[a..b]$, la recherche de la position i de plus petit support s'effectue en $O(n)$ (lignes 32 à 36).

La complexité temporelle du calcul de la collection associée au consensus et de son score global (*cf.* Algorithme 4.6 – page 126) est directement liée à la valeur du support des listes du vecteur T_i passée en paramètre (*cf.* Définition 4.25 – page 119). Dans ce dernier algorithme, la boucle principale (lignes 16 à 35) effectue t itérations. Chaque itération correspond au traitement d'une liste des triplets de cardinalité en $O(n)$. Le calcul d'un Z-score (ligne 20) d'une paire de motifs de même longueur s'effectue en temps linéaire sur cette longueur, soit en $O(n)$. Ainsi le traitement effectué de la ligne 18 à la ligne 26 requiert un temps en $O(n^2)$. Le calcul de la P-valeur associée au Z-score (ligne 29) s'effectue en temps constant au moyen d'une table statique (*cf.* Table E.2 – page 214). Ainsi, le calcul de la collection de motifs et du score globale associés à un consensus donné requiert un temps total en $O(t\,n^2)$, d'où une complexité temporelle pour cette phase en

$$O(p\,t\,n^2). \tag{4.36}$$

Une collection de motifs comprends au plus t motifs représentés chacun par l'intervalle de référence $[a;b]$ et un décalage δ. Un solution est un quadruplet comprenant l'intervalle de référence $[a;b]$, le numéro de la séquence de référence ℓ, un score global SG et un tableau d'au plus t décalages permettant de retrouver les occurrences à partir de l'intervalle $[a;b]$ et du numéro de la séquence de référence ℓ. Chaque solution engendre donc une complexité spatiale en $O(t)$, et par conséquent l'ensemble des résultats occupe un espace en

$$O(p\,t). \tag{4.37}$$

Complexité globale de StatiSTAMS La complexité temporelle engendrée par le traitement d'une séquence de référence est la somme des complexités temporelles calculées ci-avant. Chaque séquence de $\mathcal{S}$ est considérée une fois comme séquence de référence (boucle de la ligne 21 à la ligne 30), et à l'issue des traitements de chaque séquence de référence, l'ensemble Sol (de taille $t\,p$) est restreint au p collections de meilleurs scores globaux (tri partiel par tas en $O(t\,p + p\,\log(t\,p))$). L'algorithme 4.8 – page 128 – a donc une complexité temporelle en

$$O\left(\begin{array}{ll} & (t\,p + p\,\log(t\,p)) \qquad + |\Sigma|\,t\,n \quad (4.29) \\ +t\left(\right. & \underbrace{t\,n^2 \sum_{\alpha\in\Sigma} p_\alpha^2}_{(4.30)} + \underbrace{t\,n^2}_{(4.31)} + \underbrace{t\,n^2 \sum_{\alpha\in\Sigma} p_\alpha^2}_{(4.33)} + \underbrace{p\,t\,\left(\log q\,t + n^2\,\log n\right)}_{(4.34)} + \underbrace{p\,t\,n^2}_{(4.36)} \left.\right) \end{array}\right),$$

soit

$$O\left(p\,t^2\,\left(\log q\,t + n^2\,\log n\right)\right). \tag{4.38}$$

Pour le calcul de la complexité spatiale, il suffit de remarquer que seules la table $TStock$ et la liste des solutions sont conservées ou mises à jour à chaque itération de la boucle principale. Toutes les autres variables sont réinitialisées à chaque changement de séquence de référence. Ainsi, la liste Sol requiert

un espace mémoire en $O(p\,t^2)$, d'où la complexité spatiale engendrée par l'intégralité de l'algorithme en

$$O\left(\underbrace{|\Sigma|\,t\,n}_{(4.29)} + \underbrace{t\,n^2\sum_{\alpha\in\Sigma}p_\alpha^2}_{(4.30)} + \underbrace{t\,n^2\sum_{\alpha\in\Sigma}p_\alpha^2}_{(4.32)} + \underbrace{p\,n}_{(4.35)} + \underbrace{p\,t}_{(4.37)} + p\,t^2\right),$$

soit

$$O\left(t\,n^2\sum_{\alpha\in\Sigma}p_\alpha^2 + p\,(n+t^2)\right) \qquad \left(= o\left(t^2\,n^2\right)\right). \tag{4.39}$$

4.3.5 Développement

StatiSTARS utilise une grande partie des librairies écrites pour STARS. L'algorithme a été implémenté en C++, en respectant la norme ANSI. Le programme n'est pas encore distribué. Toutefois, il est prévu de le mettre à disposition prochainement sous une licence de type GPL. De plus, en raison de sa forte compatibilité avec STARS (*i.e.*, le format des entrées et des sorties du programme sont quasiment identiques), il est prévu de faire évoluer l'interface WebStars© (*cf.* Section 3.3.7 – page 73) afin d'ajouter la prise en charge de StatiSTARS, ce qui permettra notamment de générer automatiquement des « *Sequence Logo* » de chaque motif consensuel extrait.

4.4 Tests & Résultats

Les tests présentés ci-dessous ont été réalisés sur la même plate-forme que ceux réalisés pour STARS (*cf.* Section 3.4 – page 73) ; à savoir, sur un serveur de calcul quadri-processeur *Pentium*® III, cadencés chacun à $700MHz$, et disposant de $4Go$ de mémoire vive.

Le protocole d'expérimentation de l'algorithme est également le même, composé de deux parties. Dans un premier temps les tests ont été réalisés sur des séquences générées aléatoirement dans lesquelles des motifs ont été insérés, puis sur des séquences biologiques, contenant des motifs validés expérimentalement. En outre, une étude comparative des deux algorithmes est proposée à la fin des tests.

4.4.1 Séquences générées aléatoirement

Les tests sur les séquences aléatoires ont été menés afin d'estimer la performance de l'algorithme, et par conséquent, pour valider le choix des heuristiques utilisées (score $\mathcal{H}$, motifs caractéristiques, troncature des résultats). Pour cela, tout comme pour STARS, l'algorithme est considéré comme un test statistique (*cf.* Table 4.4 – page suivante), dont les solutions renvoyées sont soit correctes (Vrai Positifs VP), soit erronées (Faux Positifs FP). Les solutions non détectées (Faux Négatifs FN) sont également comptabilisées. Le reste correspond bien évidemment aux Vrai Négatifs (VN). Les séquences utilisées sont les mêmes que celles utilisées pour évaluer le premier algorithme, à savoir des séquences générées aléatoirement sur un alphabet à quatre symboles par une source uniforme et équiprobable (chaque symbole à la même probabilité d'être émis : $1/4$). Plusieurs motifs sont également générés par cette même source, puis altérés pour enfin être insérés à une position au hasard dans les séquences. Lorsqu'une position d'un

	Critère validé	Critère non validé
Test positif	Vrais Positifs (VP)	Faux Positifs (FP)
Test négatif	Faux Négatifs (FN)	Vrais Négatifs (VN)

Table 4.4 – Classification des résultats dans les tests statistiques.

motif retourné par l'algorithme correspond à une position d'un motif inséré, cette position est comptabilisée parmi les VP, sinon elle est comptabilisée parmi les FP. Idem pour la détermination des FN et des VN.

À partir de ces données sont calculés la sensibilité et la valeur prédictive positive (VPP) du test [128] (*cf.* Table 4.5).

$$\text{Sensibilité} = \frac{\text{VP}}{\text{VP} + \text{FN}} \qquad \text{VPP} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

Table 4.5 – Calcul de la sensibilité et de la valeur prédictive positive d'un test statistique.

Chaque jeu de tests comprend au moins 10 séquences de taille au moins 100 ; 1 000 jeux de tests ont été engendrés ; trois collections de motifs 5-similaires de tailles minimales fixées (25, 15 et 10) et donnant chacun un score différent fixé sont ensuite insérés dans chacun des 1 000 jeux d'essai.

L'algorithme a été exécuté sur chaque ensemble avec en paramètres $p = 1$ (recherche du meilleur motif), $Simil = 5$-similarité, $q = 100$ (nombre total de séquences), $\alpha = 1\%$, $f^{=}(k) = 2\,k$ et $f^{\neq}(k) = -k$, ce qui correspond aux conditions les plus proches du paramétrage utilisé par STAMS. Pour chaque jeu d'essai, la VPP et la sensibilité ont été calculées, et pour ce paramétrage, la VPP moyenne observée sur l'ensemble des jeux de tests est 0,82 tandis que la sensibilité moyenne observée sur ce même ensemble est 0,83 (*cf.* Figure 4.10). Toutefois, l'extraction de motifs 4-similaires dans ces mêmes ensembles de séquences (les autres paramètres demeurant inchangés), améliore la VPP moyenne observée, qui devient 0,98 ainsi que la sensibilité moyenne qui passe à 0,90 (*cf.* Figure 4.10). Pour chacun des deux diagrammes, l'axe des abscisses correspond à la VPP et à la sensibilité exprimée en pourcentage, et l'axe des ordonnées au nombre de jeux de tests ayant obtenu un pourcentage donné.

La différence de résultats observable entre l'extraction de motifs 4-similaires et l'extraction de motifs 5-similaires s'explique en partie du fait que lors de la découverte de motifs 4-similaires, les effets de bords sont plus limités, et surtout que les collections de motifs insérés recouvrent généralement plusieurs collections de motifs 4-similaires (*cf.* Figure 3.1). Enfin, les mêmes tests réalisés en n'insérant qu'une seule collection de motifs donnent pour la découverte de motifs 5-similaires une VPP et une sensibilité égales toutes deux en moyenne à 0,90.

D'autres séries de tests ont été réalisés afin de mesurer la validité de StatiSTAMS lorsque les motifs insérés ne sont pas présents dans toutes les séquences. Ainsi, 4 série de jeux de tests supplémentaires ont été élaborés. Chaque série de jeux comprend 100 ensembles de 10 séquences de longueur au moins 100. Pour la première série de jeux de test, pour chaque ensemble de séquences, une collection de motifs 5-similaires de longueur minimale 20 a été insérée dans au moins 7 des 10 séquences (contrainte 70%). Les trois autres série de jeux de séquences correspondent respectivement aux contraintes de quorum 80%, 90% et enfin 100%. De la même manière que précédemment, les VPP et sensibilités moyennes ont été calculées (*cf.* Table 4.6 – page suivante).

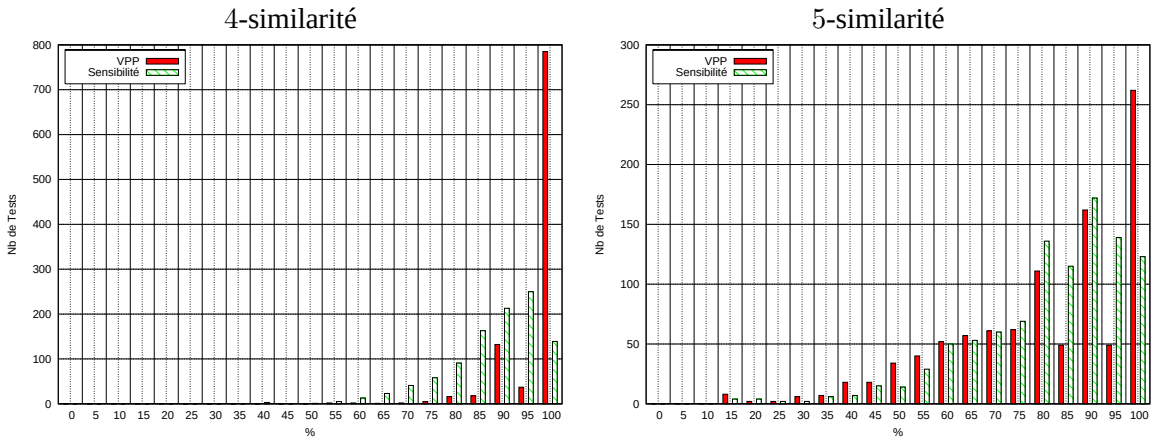


Figure 4.10 – Sensibilité et VPP de StatiSTARS mesurées sur 1 000 jeux de séquences générés aléatoirement.

4-similarité

Contrainte	VPP	Sensibilité
70%	0,83	0,87
80%	0,92	0,88
90%	0,96	0,92
100%	0,99	0,90

5-similarité

Contrainte	VPP	Sensibilité
70%	0,42	0,65
80%	0,60	0,76
90%	0,71	0,81
100%	0,90	0,90

Table 4.6 – Sensibilité et VPP de StatiSTARS mesurées sur 100 jeux de 10 séquences générés aléatoirement pour différentes contraintes de quorum.

Comme précédemment, les résultats obtenus en recherchant des motifs 4-similaires sont plus satisfaisant que ceux obtenus en recherchant des motifs 5-similaires. Les raisons en sont les mêmes que celles évoquées précédemment. En considérant maintenant l'usage d'autres fonctions de score, les meilleurs résultats obtenus parmi l'ensemble des tests effectués sont obtenus en recherchant des motifs 4-similaires avec comme fonctions composantes de score $f^{=}(k) = k^2$ et $f^{\neq}(k) = -k$, avec une VPP moyenne supérieure à 0,90 et une sensibilité supérieure à 0,85 et ce pour toutes les contraintes de 70% à 100%.

4.4.2 Séquences biologiques

Les tests sur des séquences biologiques ont été réalisés sur les 5 ensembles de séquences d'ADN d'*Escherichia coli*, décrits à la section 3.4.2 – page 76. Il s'agit d'ensembles construits à partir des sites de liaison ADN-protéines présents dans le génome d'*E. coli*. Ces ensembles sont présentés à nouveau dans la table 4.7.

Ensemble	Nb de séquences	Longueur $[\min, \max]$	Protéine régulatrice
1	9	$[53, 806]$	ArgR
2	8	$[53, 806]$	ArgR (sites en tandem)
3	16	$[100, 3\,841]$	LexA
4	18	$[299, 5\,795]$	PurR
5	8	$[251, 2\,594]$	TyrR

Table 4.7 – Synthèse de 5 ensembles de séquences d'*E. coli*.

Les jeux de séquences ont été traités par StatiSTARS, d'abord en utilisant le paramétrage le plus proche de STARS, *i.e.*, $e = 5$, $f^{=}(k) = 2\,k$, $f^{\neq}(k) = -k$, $q = 100\%$ et $\alpha = 5\%$. Les résultats obtenus sont synthétisés dans la table 4.8.

Ensemble	Protéine régulatrice	Motif consensuel extrait	« *Sequence Logo* » associé
1	ArgR	ATGAATAATTACACATATAAAGTGAATTTTAATTCAATAA	Figure 4.11
2	ArgR en tandem	AAATGAATAATCATCCATATAAATTGAATTTTAATTCATTGAG	Figure 4.12
3	LexA	GTTCACGCCGGATGCGCGTGAACGCCTTATTCGACCTATAA	Figure 4.13
4	PurR	TGGCGCAGGTTTATCGTCAGCTTGGCGACAAACCGGCAGATGTACGCGATGTCGCGCAA	Figure 4.14
5	TyrR	GAACCCTGGGAAGTGTTTTTATCGTGGCGGGAACCACAATT	Figure 4.15

Table 4.8 – Motifs extraits par StatiSTARS ($e = 5$) pour les jeux de séquences d'*E. coli*.

Le résultat présenté pour les deux premiers ensembles sont très satisfaisants. Le « débordement » à droite du motif consensuel du premier jeu d'essais par rapport au « *Sequence Logo* » est dû au fait que 8 des 9 séquences contiennent ce motif en tandem, et que la dernière séquence contient seulement le début de la seconde occurrence, comme cela a déjà été souligné dans le chapitre précédent. Les occurrences de ces motifs sont correctement détectées.

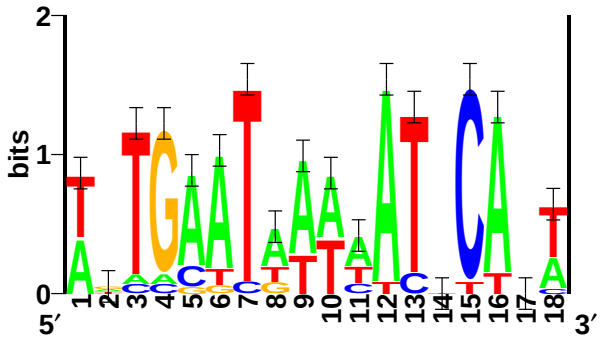


Figure 4.11 – « *Sequence Logo* » du site de liaison à ArgR.

A contrario, les trois autres jeux de séquences donne des résultats très peu satisfaisant. Les motifs proposés [114] par les « *Sequence Logo* » ont peu de similitudes avec les motifs extraits par StatiSTARS.

Cependant, en faisant varier les différents paramètres (e, α, $f^{=}$,$f^{\neq}$ et q). La majorité des combinaisons effectuées donnent de très bons résultats. Pour la fonction de score construite avec $f^{=}(k) = 2\,k$ et $f^{\neq}(k) = -k$, une erreur de première espèce $\alpha = 5\%$ et en fixant le quorum q à 100%, les résultats obtenus pour chaque jeu de séquence sont synthétisés dans la table 4.9 – page suivante, et détaillés ci-après. Les « *Sequence Logo* » associés aux jeux d'essais (*cf.* Figures 3.10 à 3.14) sont de nouveau fournis dans ce chapitre afin d'en faciliter la consultation. D'ores et déjà, il est possible de constater que les résultats obtenus sont quasiment identiques à ceux obtenus par STARS (*cf.* Table 3.6 – page 76). De manière générale, les motifs consensuels fournis par StatiSTARS sont ceux fournis par STARS, étendus d'un ou deux symboles à droite ou à gauche.

Le résultat présenté pour le premier ensemble a été obtenu en fixant $e = 4$. Le « débordement » à droite du motif consensuel par rapport au « *Sequence Logo* » trouve exactement la même explication que précédemment. En fixant une contrainte de quorum $q = 80\%$, le motif consensuel extrait est

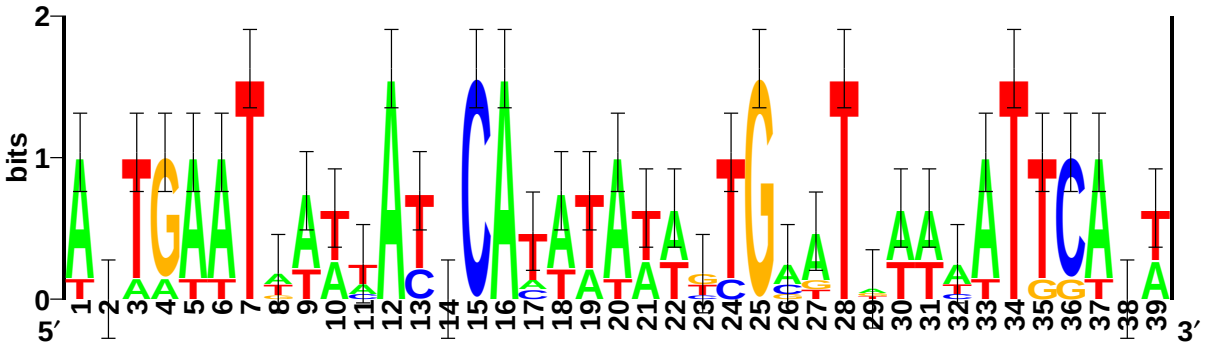


Figure 4.12 – « *Sequence Logo* » du site de liaison (en tandem) à ArgR.

Ensemble	Protéine régulatrice	Motif consensuel extrait	« *Sequence Logo* » associé
1	ArgR	AATGAATAATTACACATATAAAGTGAATTT	Figure 4.11
2	ArgR en tandem	TAAATGAAAACTCATTTATTTTGCATAAAAATTCAGT	Figure 4.12
3	LexA	TCCTGTTAATCCATACAG	Figure 4.13
4	PurR	GCAAACGTTTTC	Figure 4.14
5	TyrR	TGTATTGAGATTTTCACTTT	Figure 4.15

Table 4.9 – Motifs extraits par StatiSTABS (e variable) pour les jeux de séquences d'*E. coli*.

alors ATGAATAATTACACATATAAAGTGAATTTTAATTCAATAAGTG. Ce qui correspond bien au site de liaison en tandem (*cf.* Figure 4.12). Les occurrences de ce motif sont bien détectées, et la séquence qui ne possède effectivement pas d'occurrence du motif en tandem est bien décelée.

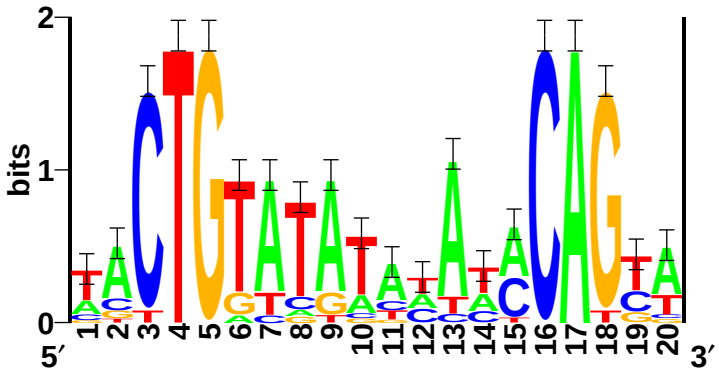


Figure 4.13 – « *Sequence Logo* » du site de liaison à LexA.

Pour les jeux d'essais 2, 3 et 5, les motifs consensuels extraits sont encore plus proches de la réalité que ceux extraits par STABS. Les différentes occurrences sont bien identifiées. Ils ont été obtenus en choisissant $e = 4$ pour le second jeu de séquences, et $e = 3$ pour les jeux 3 et 5.

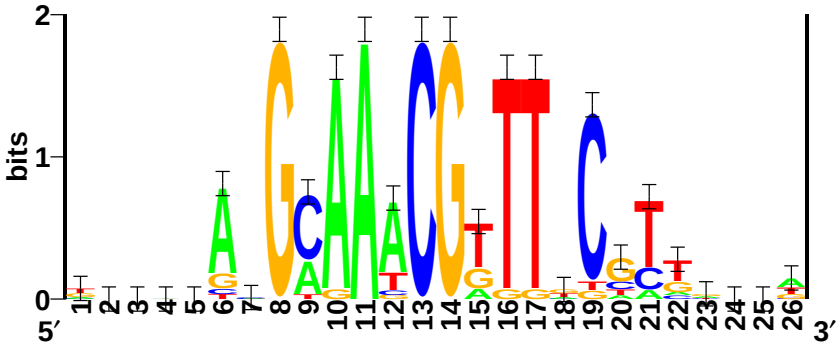


Figure 4.14 – « *Sequence Logo* » du site de liaison à PurR.

Concernant le $4^{ème}$ ensemble, le résultat est absolument identique à celui fourni par STARS. Les caractères conservés sont tous retranscrits, et les caractères non conservés ne le sont pas. Il a été obtenu en fixant $e = 2$.

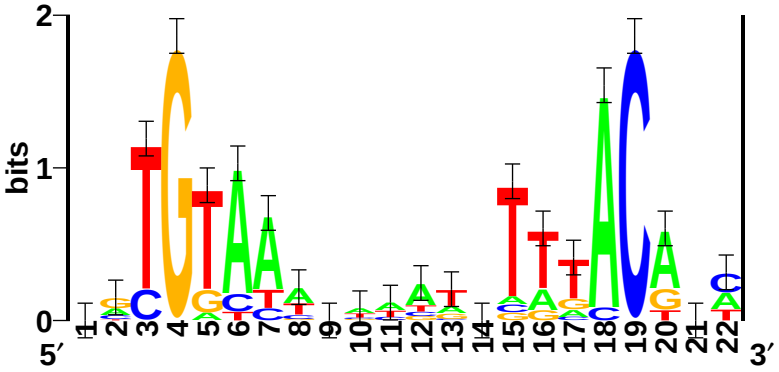


Figure 4.15 – « *Sequence Logo* » du site de liaison à TyrR.

4.4.3 Étude Comparative entre STARS et StatiSTARS

Le chapitre précédent se concluait par le constat que les résultats de STARS sont très proches de la réalité biologique et comparables à ceux des algorithmes les plus performants. Les résultats de StatiSTARS sont non seulement comparables à ceux obtenus par STARS sur les séquences biologiques, mais ils sont encore plus fins. Bien que sur les séquences générées aléatoirement, les indicateurs statistiques soient moins élevés que ceux de STARS (dont la VPP moyenne observée est 0,95 et la sensibilité moyenne 0,91, *cf.* Section 3.4.1 – page 74), StatiSTARS intègre la gestion d'une contrainte de quorum, et de surcroît, StatiSTARS offre une indication statistique sur la pertinence des motifs extraits. En effet, tous les motifs biologiques présentés dans la table 4.9 – page ci-contre – ont un score global compris entre 0,99 et 1, indiquant donc une P-valeur de moins de 1%. Enfin, un autre avantage non négligeable de StatiSTARS

par rapport à STARS provient de sa complexité, tant spatiale que temporelle, qui lui confère une plus grande rapidité d'exécution. À titre de rappel, la complexité temporelle moyenne observée de STARS est $O\left(p^2\,t^3\,n^2\right)$ contre $O\left(p\,t^2\,\left(\log q\,t + n^2\,\log n\right)\right)$ dans le pire des cas pour StatiSTARS ; la complexité spatiale de STARS étant au pire en $O\left(t^2\,n^2/e\right)$ contre $o\left(t^2\,n^2\right)$ pour StatiSTARS. Ces différences de complexités ont un impact notable sur les temps d'exécution. En effet, là où le premier algorithme met parfois plusieurs heures de calcul, StatiSTARS a besoin de seulement quelques minutes pour afficher ses résultats. À titre d'exemple, sur le jeu d'essai $ArgR$, STARS s'exécute en plus de 15 secondes sur un *Pentium*® III cadencés à $700MHz$, alors que StatiSTARS s'exécute en moins de 3 secondes.

Du point de vue applicatif, les deux intérêts majeurs de ce nouvel algorithme par rapport à STARS sont d'une part sa vitesse d'exécution, et d'autre part la gestion de la contrainte de quorum. Cependant, si ces deux avantages ne sont pas primordiaux, et que le souhait de l'utilisateur est de répondre au problème EM en privilégiant par dessus tout l'aspect qualitatif des résultats, STARS est alors la solution qui prime. Concernant le paramétrage de StatiSTARS, dans la mesure où les temps d'exécution sont très faibles, il est clairement envisageable d'essayer plusieurs combinaisons. Néanmoins, une erreur de première espèce au dessus de 5% ou en deçà de 1% n'apporteront probablement pas une réponse pertinente, de même une valeur élevée de e risque fort d'être une entrave à la qualité des résultats.

CHAPITRE 5

Étude des Oracles des Facteurs et des Suffixes

ὁ ἄναξ οὗ τὸ Μαντειόν ἐστι τὸ ἐν Δελφοῖς, οὔτε λέγει οὔτε κρύπτει ἀλλὰ σημαίνει.
Le dieu dont l'oracle est à Delphes ne révèle ni ne cache, il indique.

— HÉRACLITE [540–480 av. JC], Fragment 93 (traduction de Paul TANNERY – 1887).

L'objet de ce chapitre est l'étude d'une structure d'index, appelée Oracle, ayant deux versions : l'Oracle des Facteurs et l'Oracle des Suffixes. Le résultat principal, la caractérisation combinatoire du langage obtenu par chacun de ces Oracles, a été publié dans [96]. Il se base sur la définition de facteurs particuliers, appelés facteurs remarquables, et d'une opération spécifique, la contraction, qui, ensemble, permettent d'engendrer les langages sus-cités.

L'introduction d'une structure d'index dans les algorithmes vus dans les chapitres précédents apporterait avec elle tous les deux principaux avantages des index : la compacité de la représentation des données à traiter, et les facilités d'analyse (du point de vue de la complexité). Dans ce but, notre intérêt s'est rapidement orienté vers les Oracles, deux structures séduisantes tant par leur étonnante simplicité de construction que par leur faible complexité. De surcroît, les méthodes et outils actuels basés sur ces structures offrent de bons résultats. Toutefois, avant de les intégrer dans un quelconque algorithme, il est nécessaire d'approfondir le champ de connaissance de ces index. En effet, lorsque nous avons commencé notre étude, plusieurs questions ouvertes à leur sujet demeuraient, dont la détermination du langage que ces structures reconnaissent. Sans cette connaissance, la fiabilité des Oracles ne peut pas être rigoureusement établie, et entraînerait une fiabilité méconnue de nos algorithmes. Nous avons donc effectué une étude approfondie des Oracles, afin d'en dégager leurs propriétés, leurs modalités d'utilisations, leurs avantages et leurs inconvénients.

Les deux résultats principaux de ce chapitre (*cf.* Théorèmes 5.12 – page 160 – et 5.13 – page 161) permettent, en caractérisant les langages reconnus par les Oracles, à la fois de mieux comprendre la nature intrinsèque des Oracles, et surtout de mettre en évidence leur manque de fiabilité en terme de « prédicteurs ». Fort de ces connaissances, une déclinaison de ces structures est alors présentée, offrant des performances très nettement supérieures, les Oracles à transitions gardées.

5.1 Principales structures d'index

Les structures d'index présentées ici sont soit des arbres (*Trie*, PATRICIA *Trie*, Arbres des Suffixes), soit des automates (Automates des Facteurs, Automates des Suffixes, Oracles des Facteurs, Oracles des Suffixes). Les définitions de base sur les arbres et les automates sont données dans l'annexe D.

5.1.1 Les [PATRICIA] *Trie*

Le *Trie* est un arbre enraciné construit à partir d'un ensemble de mots. Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble. Dans cet arbre, tous les mots de l'ensemble sont ainsi représentés et aucun mot supplémentaire n'y est présent (*cf.* Figure 5.1). Cette structure a été proposée en 1960 par FREDKIN [46]. L'idée de la construction est basée sur le fait que tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe. Le terme *Trie* provient du mot anglais *retrieval*. Afin de gérer le cas où un mot et un de ses préfixes propres apparaissent dans l'ensemble à représenter, l'alphabet est étendu avec un nouveau symbole (généralement \$) ajouté à chaque fin de mot. Ainsi, un ensemble de n mots sera représenté par un *Trie* à n feuilles, et de profondeur égale à la longueur du plus grand mot de l'ensemble.

Le PATRICIA *Trie* est une version compacte d'un *Trie*. Cette structure a été proposée en 1968 par MORRISON [101]. Son appellation est l'acronyme de *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*. Dans cette version, seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés (*cf.* Figure 5.1).

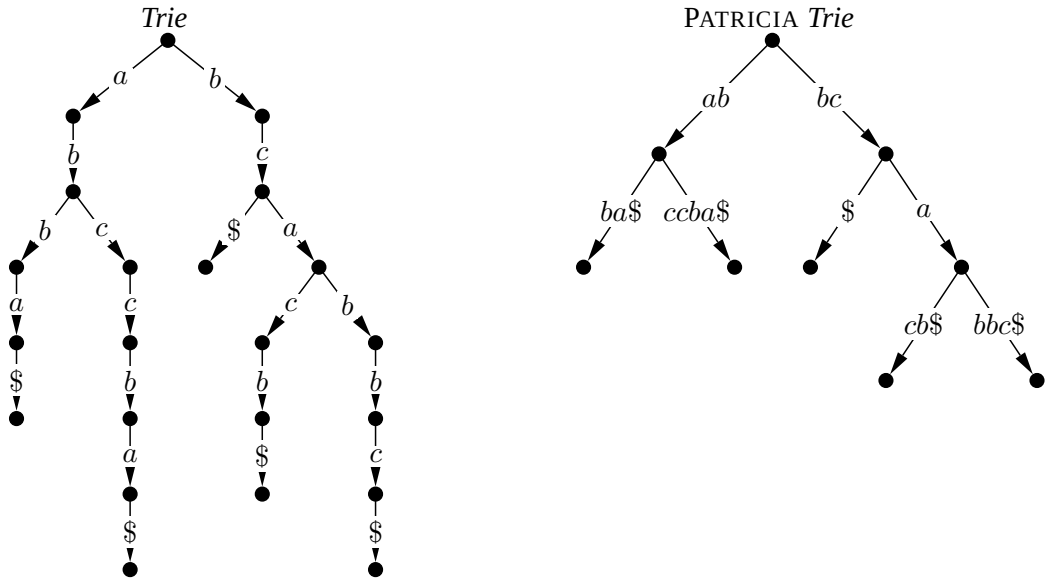


Figure 5.1 – *Trie* et PATRICIA *Trie* de l'ensemble de mots $\{abba, abccba, bc, bcacb, bcabbc\}$.

5.1.2 Les Arbres des Suffixes

Ces arbres sont des *Trie*, construits à partir de l'ensemble des suffixes d'un mot. Les arbres des suffixes compacts sont alors les PATRICIA *Trie* obtenus à partir de l'ensemble des suffixes d'un mot (*cf.* Figure 5.2). Il est possible de construire un tel arbre en temps linéaire. En effet, les suffixes ne sont pas indépendants les uns des autres, et cette information est exploitée pour construire l'arbre en temps linéaire par rapport à la taille du mot représenté. Le premier algorithme à avoir atteint cette complexité pour la construction d'un tel arbre a été proposé par WEINER en 1973 [136]. Dans cet algorithme, la construction s'effectue en lisant le mot à représenter de la droite vers la gauche. Il faut attendre 1995 pour voir apparaître le premier algorithme linéaire permettant de construire cette structure en lisant le mot de la gauche vers la droite [127]. C'est en effet à cette date qu'UKKONEN a présenté son célèbre algorithme de construction *on-line* de l'arbre des suffixes (*i.e.*, la structure construite à la $i^{ème}$ étape de l'algorithme appliqué à un mot de longueur n est l'arbre des suffixes du préfixe de longueur i du mot représenté).

5.1.3 Les automates des Facteurs/Suffixes

L'automate des Facteurs est un automate reconnaissant exactement et uniquement tous les facteurs du mot à partir duquel il a été élaboré (*cf.* Figure 5.3, où tous les états sont considérés finaux). Il est également connu sous le nom plus générique de DAWG (pour *Directed Acyclic Word Graph*). La construction de l'automate des Facteurs d'un mot donné peut s'effectuer en temps linéaire [33]. De plus, cette structure possède la propriété d'avoir au plus $2\,n - 1$ états pour un mot de longueur n. Tous les états sont finaux dans cet automate.

L'automate des Suffixes est un automate reconnaissant exactement et uniquement tous les suffixes du mot à partir duquel il a été construit. Sa structure est identique à celle de l'Automate des Facteurs, excepté

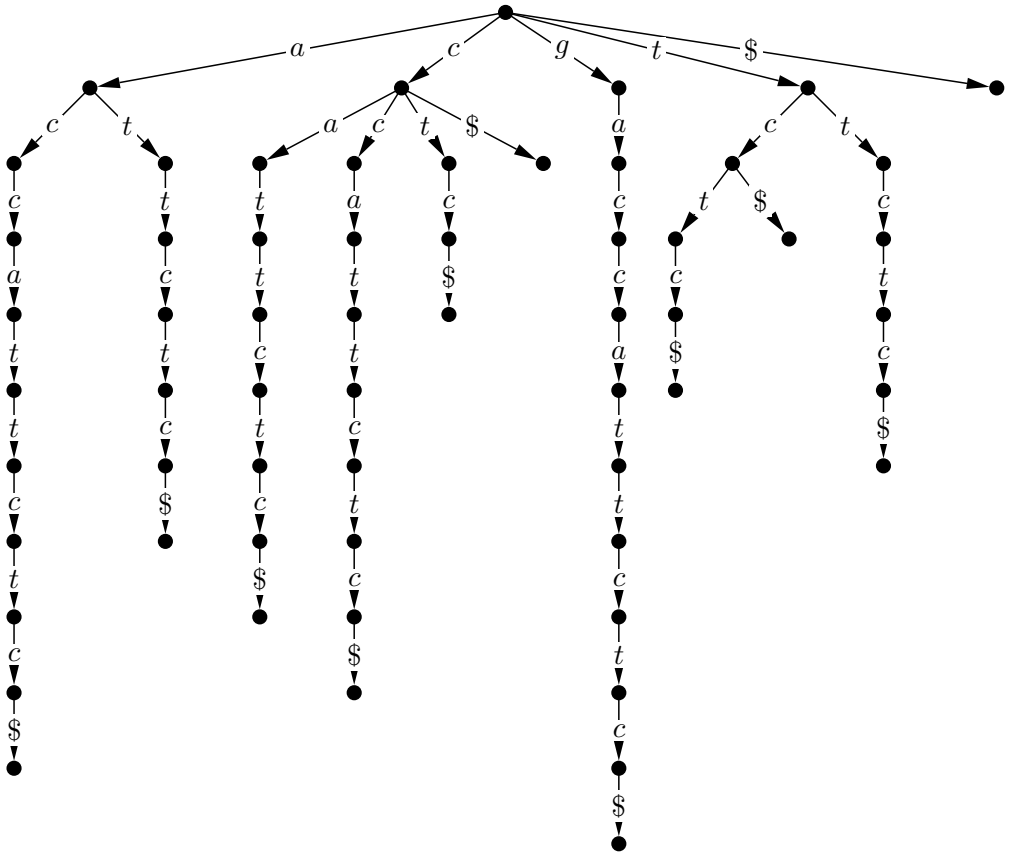


Figure 5.2 – Arbre des Suffixes du mot $gaccattctc$.

que seuls les états de reconnaissance des suffixes du mot représenté sont finaux (*cf.* Figure 5.3). Les états finaux sont déterminés lors de la construction, ce qui ne modifie pas la complexité de l'algorithme. De surcroît, cet automate est minimal en nombre d'état (*cf.* Définition D.24 – page 207).

5.1.4 Les Oracles de Facteurs/Suffixes

L'Oracle des Facteurs est également un automate. Il reconnait au moins tous les facteurs du mot à partir duquel il a été construit (*cf.* Figure 5.4, où tous les états sont considérés finaux). Cette structure a été introduite en 1999 par ALLAUZEN *& al.* [3]. L'objectif de leurs travaux était d'élaborer une méthode permettant de construire un automate *acyclique*, reconnaissant **au moins** tous les facteurs d'un mot s, ayant le moins d'état possible, et qui soit *linéaire* en nombre de transitions. Ainsi cette structure possède-t-elle exactement $n+1$ états pour un mot de longueur n et au plus $2\,n-1$ transitions. De plus, l'automate est homogène (*i.e.,* toutes les transitions arrivant dans un même état ont la même étiquette).

L' Oracle des Suffixes est également un automate. Il reconnait au moins tous les suffixes du mot à partir duquel il a été élaboré. Sa structure est identique à celle de l'Oracle des Facteurs pour le même mot,

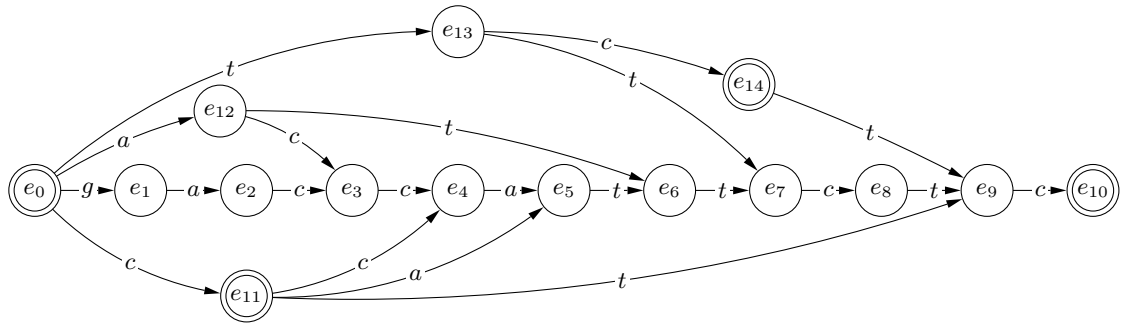


Figure 5.3 – Automate des Facteurs/Suffixes du mot $gaccattctc$. Dans l'Automate des Suffixes, seuls les états représentés par un double cercle sont finaux.

excepté que seuls les états de reconnaissance des suffixes du mot représenté sont finaux (*cf.* Figure 5.4). Les états finaux sont ici aussi déterminés lors de la construction, ce qui ne modifie pas la complexité de l'algorithme.

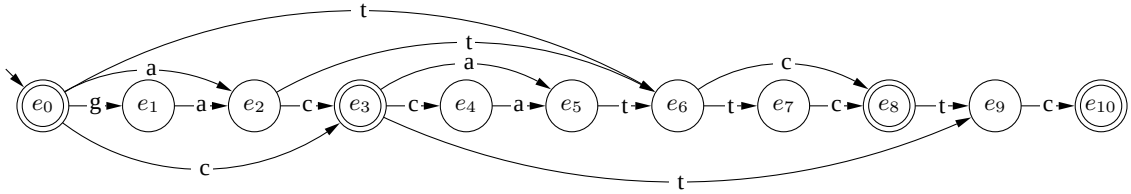


Figure 5.4 – Oracle des Facteurs/Suffixes du mot $gaccattctc$. Dans l'Oracle des Suffixes, seuls les états représentés par un double cercle sont finaux.

5.2 Quelques particularités sur les Oracles

Rappelons ici trois notations qui seront beaucoup utilisées par la suite : $Fact(s)$ note l'ensemble de facteurs du mot s, $Suff(s)$ son ensemble de suffixes et $Pref(s)$ son ensemble de préfixes. Notons également, pour un mot w reconnu par un automate, $Etat(w)$ l'état d'acceptation de ce mot (en supposant que l'automate est bien identifié, la notation n'est pas ambiguë).

En vue de présenter plus précisément les Oracles, il et nécessaire de définir les deux fonctions suivantes :

Définition 5.1. Étant donnés un mot $s \in \Sigma^+$ et un facteur x de s, la fonction $\underline{Pos}$ est définie comme étant la position de la première occurrence de x dans s, *i.e.*, s peut être écrit $s = u\,x\,v$ ($u, v \in \Sigma^*$, $x \notin Fact(s[1..|u\,x| - 1])$) et alors $Pos_s(x) = |u| + 1$. La fonction $\underline{poccur}$ est alors définie telle que $poccur_s(x) = |u\,x| = Pos_s(x) + |x| - 1$. Cette fonction renvoie la position de la fin de la première occurrence de x dans s.

L'Oracle d'un mot donné est défini par l'algorithme 5.1 – page ci-contre.

```
1  Nom : Construit_Oracle
2  Entrées : Σ % Alphabet (supposé minimal) %
3            s ∈ Σ* % Le mot à traiter %
4  Sortie : Oracle % L'Oracle des Facteurs de s %
5
6  Début
7    Créer l'état initial étiqueté par e_0
8
9    Pour i de 1 à |s| Faire
10     Créer un état étiqueté par e_i
11     Ajouter une transition de l'état e_{i-1} vers l'état e_i étiquetée par s[i]
12   Fin Pour
13
14   Pour i de 0 à |s| - 1 Faire
15     Soit u un mot de longueur minimale reconnu à l'état e_i
16     Pour Tout α ∈ Σ \ {s[i+1]} Faire
17       Si u α ∈ Fact(s[i - |u| + 1..|s|]) Alors
18         j ← poccur_{s[i-|u|+1..|s|]}(u α) - |u|
19         Ajouter une transition de l'état e_i vers l'état e_{i+j} étiquetée par α
20       Fin Si
21     Fin Pour Tout
22   Fin Pour
23 Fin
```

Algorithme 5.1 – Construction de l'Oracle des Facteurs d'un mot

Lorsque tous les états de cet automate sont finaux, cette structure est appelée l'*Oracle des Facteurs.* Lorsque seul un sous-ensemble « particulier » de ces états sont finaux, l'automate devient alors l'*Oracle des Suffixes.*

Définition 5.2. Étant donné un mot $s \in \Sigma^*$, l'*Oracle des Facteurs* de s est défini comme étant l'automate obtenu par l'algorithme 5.1, dont tous les états sont considérés comme finaux ; il est noté $FO(s)$. De même, l'*Oracle des Suffixes* de s est défini comme étant l'automate obtenu par ce même algorithme, où seuls les états e_i ($0 \leq i \leq |s|$) tels qu'il existe un chemin issu de l'état initial vers e_i étiquetant un suffixe de s sont finaux ; il est noté $SO(s)$.

L'automate obtenu par cet algorithme pour un mot s comporte exactement $|s|+1$ états, ce qui correspond au plus petit nombre d'états qu'il est possible d'obtenir pour un automate acyclique reconnaissant tous les facteurs ou suffixes de s. Le nombre de transitions de cet automate est borné par $2\,|s| - 1$. En outre, cet automate est homogène, *i.e.,* toutes les transitions arrivant dans le même état sont étiquetées par le même symbole. Ainsi, il n'est pas nécessaire d'étiqueter les arêtes lors de la représentation de l'automate en machine. Enfin, cette structure nécessite moins de mémoire que les Arbres des Suffixes ou encore les *Tries* [51].

Définition 5.3. Toute transition issue d'un état e_i vers un état e_j tel que $j - i > 1$ ($0 \leq i < j \leq |s|$) est dite externe, et toute transition issue d'un état e_i vers un état e_{i+1} ($0 \leq i < |s|$) est dite interne.

L'algorithme 5.1 est quadratique. Il existe un autre algorithme linéaire produisant exactement la même structure. Cet algorithme est détaillé à la section 5.5 – page 163. Il est à noter que ce dernier est

également aisé à implémenter. Néanmoins, la simplicité de construction et d'utilisation des Oracles est contrebalancée par un inconvénient majeur lié à cette structure. En effet, si tous les facteurs (respectivement suffixes) d'un mot s sont reconnus par l'Oracle des Facteurs (respectivement Suffixes) de s, d'autres mots qui ne sont pas facteurs (respectivement suffixes) de s sont également acceptés. La seule certitude que confère l'Oracle est qu'un mot rejeté n'est pas facteur (respectivement suffixe) de s. La caractérisation du langage reconnu par les Oracles était, au moment où nous nous sommes intéressés aux Oracles, un problème ouvert. Avant d'utiliser cette structure pour l'algorithme STABS, il est donc apparu nécessaire de s'intéresser à cette question, afin de comprendre sa nature intrinsèque d'une part, et surtout de mesurer le risque encouru par des réponses erronées lors de son utilisation.

5.2.1 Quelques définitions

Dans la suite, l'usage du terme Oracle, sans davantage de précision sur sa nature, signifie que les faits énoncés sont valables tant pour la version Facteur que la version Suffixe.

Le lemme suivant résume plusieurs propriétés sur les Oracles énoncées par ALLAUZEN *& al.* dans [3].

Lemme 5.1. *Étant donnés un mot $s \in \Sigma^*$ et son Oracle, un mot unique de longueur minimum est accepté à chaque état e_i ($0 \leq i \leq |s|$) de l'Oracle de s. Il est noté $min(e_i)$. De plus, pour tout entier i ($0 \leq i \leq |s|$), $min(e_i) \in Fact(s)$ et $i = poccur_s(min(e_i))$.*

5.2.2 Facteurs Remarquables

La caractérisation du langage reconnu par les Oracles nécessite de définir certains facteurs particuliers.

Définition 5.4. Étant donnés un mot $s \in \Sigma^*$ et son Oracle, l'ensemble des facteurs remarquables de s, noté $\mathcal{F}_s$, est défini par :

$$\mathcal{F}_s = \{min(e_i) \,|\, 1 \leq i < |s| \ \wedge \ (\#_{out}(e_i) > 1 \ \vee \ min(e_i) \in Suff(s))\},$$

où $\#_{out}(e_i)$ désigne le degré sortant de l'état e_i.

Dans la suite, la première occurrence d'un facteur remarquable doit se distinguer de ses éventuelles autres occurrences. Par conséquent, il est nécessaire de définir une nouvelle notion.

Définition 5.5. Étant donné un suffixe v de s (tel que $s = u\,v$) et un facteur remarquable f de s, f est un facteur remarquable conservé de s dans v si la première occurrence de f dans s apparaît dans v. L'ensemble des facteurs remarquables conservés de s dans v est noté $\mathcal{F}_{s,v}$ (ainsi $\mathcal{F}_{s,v} \subseteq \mathcal{F}_s$).

L'exemple 5.1 illustre les notions de facteurs remarquables et de facteurs remarquables conservés.

Exemple 5.1 (Facteurs remarquables dans un Oracle).
En considérant le mot $gaccattctc$ et son Oracle (cf. Figure 5.5), $Etat(cat) = e_6$, $min(e_6) = t$ et $\#_{out}(e_6) = 2$. Le facteur t est un facteur remarquable de $gaccattctc$ et est également un facteur remarquable conservé de $gaccattctc$ dans le suffixe $attctc$. Il ne l'est plus pour le suffixe $tctc$.

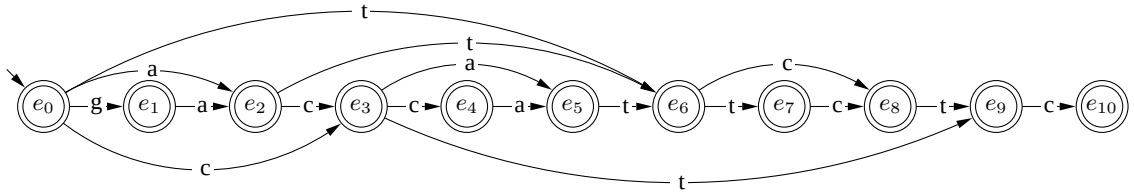


Figure 5.5 – Oracle des Facteurs du mot $gaccattctc$.

L'ensemble des facteurs remarquables du mot $gaccattctc$ est $\mathcal{F}_{gaccattctc} = \{a, c, t, tc\}$. De surcroît, les ensembles des facteurs remarquables conservés de ce mot dans ses différents suffixes sont :

$$\left\{\begin{array}{llllll}
\mathcal{F}_{gaccattctc,gaccattctc} & = & \mathcal{F}_{gaccattctc,accattctc} & = & \mathcal{F}_{gaccattctc} & = \{a,c,t,tc\},\\
\mathcal{F}_{gaccattctc,ccattctc} & = & & & & = \{c,t,tc\},\\
\mathcal{F}_{gaccattctc,cattctc} & = & \mathcal{F}_{gaccattctc,attctc} & = & \mathcal{F}_{gaccattctc,ttctc} & = \{t,tc\},\\
\mathcal{F}_{gaccattctc,tctc} & = & & & & = \{tc\},\\
\mathcal{F}_{gaccattctc,ctc} & = & \mathcal{F}_{gaccattctc,tc} & = & \mathcal{F}_{gaccattctc,c} & = \{\}.
\end{array}\right.$$

5.2.3 Opération de Contraction

Les facteurs remarquables d'un mot donné sont utilisés comme des marqueurs permettant de supprimer certaines portions de ce mot. Ceci est fait à l'aide de l'opération de contraction, qui considère la première occurrence d'un facteur remarquable et une autre occurrence de ce même facteur, puis identifie caractère par caractère ces deux occurrences en éliminant tous les caractères intermédiaires.

Définition 5.6. Étant donnés un mot $s \in \Sigma^*$ et un facteur remarquable f de s ayant au moins deux occurrences dans s, supposons que :

$$\left\{\begin{array}{lll}
s & = & u\,f\,v \qquad\qquad (u, v \in \Sigma^*)\\
f\,v & = & w\,f\,x \qquad (w \in \Sigma^+, x \in \Sigma^*)\\
Pos_s(f) & = & |u| + 1.
\end{array}\right.$$

La paire $(|u| + 1, |u\,w| + 1)$ est alors appelée une contraction de s par f. L'opération de contraction est l'action qui, étant données les factorisations $s = u\,f\,v$ et $f\,v = w\,f\,x$, crée le mot $y = u\,f\,x$.

Remarque 5.7. On remarque ici que la première occurrence de f est à la même position $|u| + 1$ dans le mot initial s et dans le mot y obtenu par contraction. Une contraction préserve donc l'emplacement du facteur remarquable qui lui correspond.

Notation 5.8. Étant donnés un mot $s \in \Sigma^*$ et un facteur remarquable f de s, l'ensemble des contractions de s par f est noté $\mathcal{C}_s^f$. L'ensemble des contractions qu'il est possible de réaliser sur s est noté $\mathcal{C}_s^*$ ($\equiv \bigcup_{f \in \mathcal{F}_s} \mathcal{C}_s^f$). Étant donné v, un suffixe de s tel que $s = u\,v$, le sous-ensemble des contractions de $\mathcal{C}_s^*$ effectué à partir des facteurs remarquables conservés de s dans v est noté $\mathcal{C}_{s,v}^*$. Ce sous-ensemble est formellement défini par

$$\mathcal{C}_{s,v}^* = \{(p, q) \mid (p, q) \in \mathcal{C}_s^* \ \wedge\ p > |u|\}.$$

L'exemple 5.2 permet d'illustrer l'opération de contraction.

Exemple 5.2 (Ensemble de contractions).
En reprenant le mot $gaccattctc$ et son Oracle (cf. Exemple 5.1 – page146), les ensembles de contractions qu'il est possible de lui appliquer sont :

$$\left.\begin{array}{rcl} \mathcal{C}^{a}_{gaccattctc} & = & \{(2,5)\} \\ \mathcal{C}^{c}_{gaccattctc} & = & \{(3,4),(3,8),(3,10)\} \\ \mathcal{C}^{t}_{gaccattctc} & = & \{(6,7),(6,9)\} \\ \mathcal{C}^{tc}_{gaccattctc} & = & \{(7,9)\} \end{array}\right\} \Rightarrow \mathcal{C}^{*}_{gaccattctc} = \left\{ \begin{array}{l} (2,5),(3,4),(3,8),(3,10), \\ (6,7),(6,9),(7,9) \end{array} \right\}$$

Il est facile de voir que tous les ensembles de contractions ne sont pas valides, dans le sens où l'application d'une contraction peut rendre inapplicables d'autres contractions de l'ensemble. Aussi, deux ou plusieurs contractions peuvent parfois être remplacées par une seule sans changer l'effet cumulé obtenu.

Définition 5.9. Un ensemble de contractions $\mathcal{C}$ est cohérent s'il ne contient pas deux contractions (i_1, j_1), (i_2, j_2) telles que : $i_1 < i_2 < j_1 < j_2$. Un ensemble de contractions cohérent est dit minimal s'il ne contient pas deux contractions (i_1, j_1) et (i_2, j_2) telles que $i_2 = i_1$ ou $i_2 = j_1$.

Remarque 5.10. Notons ici que, pour un ensemble cohérent $\mathcal{C}$, le résultat de la contraction de s par $\mathcal{C}$ est le même quel que soit l'ordre d'application des contractions. Ceci est dû, d'une part, au fait que les contractions ne se chevauchent pas et, d'autre part, au fait que la première occurrence d'un facteur remarquable f est conservée par la contraction correspondant à ce facteur (*cf.* Remarque 5.7 – page précédente). Ce dernier argument implique également que tous les facteurs remarquables dont la première occurrence a une intersection non-nulle avec la première occurrence de f gardent la position de leur première occurrence.

Il est donc possible de définir le mot résultant de l'application d'un ensemble cohérent de contractions à un mot s.

Définition 5.11. Étant donnés un mot $s \in \Sigma^*$ et l'ensemble de contractions cohérent (mais pas forcément minimal) $\mathcal{C} = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ associé aux facteurs remarquables $\{f_1, \ldots, f_k\}$, la fonction Mot (*cf.* Figure 5.6) est définie par :

$$Mot(s, \mathcal{C}) \quad = \quad s[1..p_1 - 1]\, s[q_1..p_2 - 1] \ldots s[q_{k-1}..p_k - 1]\, s[q_k..|s|].$$

Remarque 5.12. Par abus de langage et pour allèger l'écriture, la notation étendue $Mot(w, \mathcal{C})$ sera utilisée avec la même interprétation que ci-dessus pour tout sous-mot w de s bien identifié par ses positions de début et de fin sur s, et pour tout ensemble $\mathcal{C} \subseteq \mathcal{C}^*_s$. Ceci sous-entend que les contractions de $\mathcal{C}$ appliquées sur w sont celles de $\mathcal{C} \cap \mathcal{C}^*_{s,w}$.

Exemple 5.3 (Opération de contraction).
Le mot obtenu par la contraction de $s = gaccattctc$ par $\mathcal{C} = \{(3,4),(6,9)\}$ ($\mathcal{C} \subseteq \mathcal{C}^*_{gaccattctc}$) est $gaccattctc = gacatc$. Comme l'ensemble de contractions est cohérent et minimal, les parties grisées (qui sont éludées) sont disjointes. Par conséquent, l'ordre d'application des contractions n'influe pas sur le résultat.

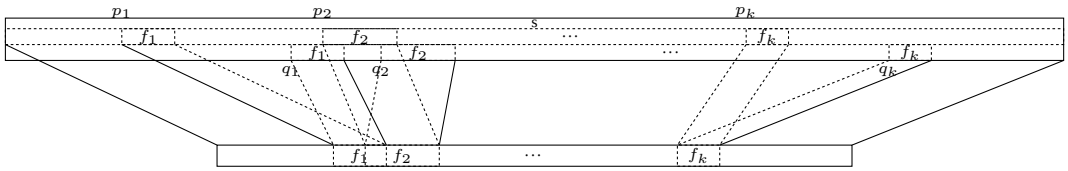


Figure 5.6 – Mots obtenus par l'opération de contraction.

5.2.4 Fermeture d'un mot

Les mots obtenus par l'opération de contraction sont utilisés afin de déterminer le langage reconnu par les Oracles. Les ensembles de contractions considérés sont donc uniquement les ensembles cohérents et minimaux, les autres ensembles ne permettant pas d'engendrer de nouveaux mots.

Définition 5.13. Étant donné un mot $s \in \Sigma^*$, la fermeture de s est l'ensemble noté $\mathcal{E}(s)$ défini par

$$\mathcal{E}(s) = \bigcup_{\substack{\mathcal{C} \subseteq \mathcal{C}_s^* \\ \mathcal{C} \text{ cohérent et minimal}}} Mot(s, \mathcal{C}).$$

Le concept de fermeture est illustré par l'exemple 5.4.

Exemple 5.4 (Fermeture d'un mot).
En reprenant l'exemple du mot $gaccattctc$, il est possible de construire 14 ensembles distincts de contractions cohérents et minimaux, permettant de générer 14 mots distincts à partir de $gaccattctc$. Bien que ça ne soit pas le cas dans cet exemple, il peut arriver que plusieurs ensembles de contractions cohérents et minimaux distincts génèrent le même mot.

$$\left\{\begin{array}{lcll}
\mathcal{C}_0 = \{\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_0) = & = gaccattctc \\
\mathcal{C}_1 = \{(2,5)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_1) = g\boldsymbol{a}cc\boldsymbol{a}ttctc & = gattctc \\
\mathcal{C}_2 = \{(3,4)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_2) = ga\boldsymbol{c}\boldsymbol{c}attctc & = gacattctc \\
\mathcal{C}_3 = \{(3,8)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_3) = gaccatt\boldsymbol{c}tc & = gactc \\
\mathcal{C}_4 = \{(3,10)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_4) = gaccattct\boldsymbol{c} & = gac \\
\mathcal{C}_5 = \{(6,7)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_5) = gacca\boldsymbol{t}\boldsymbol{t}ctc & = gaccatctc \\
\mathcal{C}_6 = \{(6,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_6) = gaccat\boldsymbol{t}c\boldsymbol{c} & = gaccatc \\
\mathcal{C}_7 = \{(7,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_7) = gaccat\boldsymbol{t}\boldsymbol{c}\boldsymbol{t}\boldsymbol{c} & = gaccattc \\
\mathcal{C}_8 = \{(2,5),(6,7)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_8) = g\boldsymbol{a}cc\boldsymbol{a}\boldsymbol{t}\boldsymbol{t}ctc & = gatctc \\
\mathcal{C}_9 = \{(2,5),(6,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_9) = g\boldsymbol{a}cc\boldsymbol{a}t\boldsymbol{t}c\boldsymbol{c} & = gatc \\
\mathcal{C}_{10} = \{(2,5),(7,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_{10}) = g\boldsymbol{a}cc\boldsymbol{a}t\boldsymbol{t}\boldsymbol{c}\boldsymbol{t}\boldsymbol{c} & = gattc \\
\mathcal{C}_{11} = \{(3,4),(6,7)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_{11}) = ga\boldsymbol{c}\boldsymbol{c}a\boldsymbol{t}\boldsymbol{t}ctc & = gacatctc \\
\mathcal{C}_{12} = \{(3,4),(6,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_{12}) = ga\boldsymbol{c}\boldsymbol{c}attc\boldsymbol{t}c & = gacatc \\
\mathcal{C}_{13} = \{(3,4),(7,9)\} & \Rightarrow & Mot(gaccattctc, \mathcal{C}_{13}) = ga\boldsymbol{c}\boldsymbol{c}at\boldsymbol{t}\boldsymbol{c}\boldsymbol{t}\boldsymbol{c} & = gacattc
\end{array}\right.$$

Ainsi, la fermeture du mot $gaccattctc$ est :

$$\mathcal{E}(gaccattctc) = \left\{ \begin{array}{l} gac, gacatc, gacatctc, gacattc, gacattctc, gaccatc, gaccatctc, \\ gaccattc, gaccattctc, gactc, gatc, gatctc, gattc, gattctc \end{array} \right\}.$$

La fermeture d'un mot va permettre, dans la suite de ce chapitre, de montrer que l'Oracle des Suffixes reconnaît exactement tous les suffixes des mots de $\mathcal{E}(s)$. Ce résultat sera ensuite utilisé pour démontrer qu'à l'instar de l'Oracle des Suffixes, l'Oracle des Facteurs reconnaît exactement tous les facteurs des mots de $\mathcal{E}(s)$. En guise d'exemple, remarquer que le mot atc est accepté par l'Oracle des Facteurs (respectivement Suffixes) de $gaccattctc$ (*cf.* Figure 5.5), alors qu'il ne s'agit pas d'un facteur (donc ni d'un suffixe) de ce mot. En revanche, atc est suffixe (et donc facteur) du mot $gacatc$ de $\mathcal{E}(s)$.

5.2.5 Propriétés des Oracles

La preuve du rôle déterminant de la notion de fermeture dans la caractérisation des ensembles de mots reconnus par les Oracles est basée sur une série de lemmes énoncés par ALLAUZEN *& al.* dans [3]. Il est donc nécessaire de les rappeler.

Lemme 5.2. *Étant donnés un mot $s \in \Sigma^*$ et un entier i ($0 \leq i \leq |s|$), le mot $min(e_i)$ est suffixe de tout mot reconnu à l'état e_i de l'Oracle de s.*

Lemme 5.3. *Étant donnés un mot $s \in \Sigma^*$ et un facteur w de s, le mot w est reconnu à l'état e_i ($1 \leq i \leq poccur_s(w)$) par l'Oracle de s.*

Lemme 5.4. *Étant donnés un mot $s \in \Sigma^*$ et un entier i ($0 \leq i \leq |s|$), tout chemin dans l'Oracle de s dont l'étiquette contient $min(e_i)$ comme suffixe arrive dans un état e_j tel que $j \geq i$.*

Lemme 5.5. *Étant donnés un mot $s \in \Sigma^*$ et $w \in \Sigma^*$ un mot reconnu par l'Oracle de s à l'état e_i, tout suffixe de w est reconnu par l'Oracle dans un état e_j tel que $j \leq i$.*

La preuve de ce dernier lemme donnée dans [3] n'est valable que dans le cas de l'Oracle des Facteurs. Étendons la preuve au cas de l'Oracle des Suffixes.

Démonstration. Le lemme original permet d'affirmer que si x est un suffixe du mot w, alors $Etat(x) \leq Etat(w)$. Il faut donc montrer que si $Etat(w)$ est final, alors $Etat(x)$ l'est également. De fait, deux cas sont à considérer. Lorsque $|x| \geq |min(e_i)|$, on obtient $min(e_i) \in Suff(x)$ (*cf.* Lemme 5.2) et, d'après le lemme 5.4, $Etat(x) \geq Etat(min(e_i))$. Or $Etat(min(e_i)) = e_i = Etat(w)$. Il s'en déduit donc que $Etat(x) = Etat(w)$. Le second cas à étudier apparaît lorsque $|x| < |min(e_i)|$. Comme l'état e_i est final, il existe nécessairement un suffixe v de s tel que $Etat(v) = e_i$. Il se déduit du lemme 5.2 que $min(e_i) \in Suff(v) \subseteq Suff(s)$. Comme x et $min(e_i)$ sont suffixes de w, la relation $|x| < |min(e_i)|$ implique nécessairement que $x \in Suff(min(e_i))$. Il en découle par conséquent que x est également suffixe du mot s, et par définition de l'Oracle des Suffixes, l'état $Etat(x)$ est final. □

Les lemmes suivants sont utilisés dans les démonstrations concernant la caractérisation des ensembles de mots reconnus. Ils mettent en avant des propriétés relatives aux facteurs remarquables et aux mots obtenus par contraction.

Lemme 5.6. *Soient un mot s, un facteur remarquable f de s tel que $s = u\,f\,v$ (avec $u, v \in \Sigma^*$) et $poccur_s(f) = |u\,f|$, et un ensemble cohérent $\mathcal{C}$ de contractions ($\mathcal{C} \subseteq \mathcal{C}_s^*$) tel que $Mot(u\,f, \mathcal{C}) = w\,f$. Alors l'Oracle de s reconnaît les mots $w\,f$ et f dans le même état.*

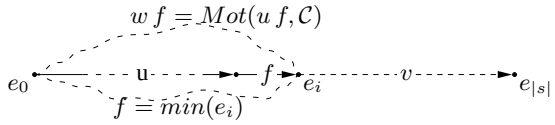


Figure 5.7 – Illustration du lemme 5.6.

Démonstration. Supposons que les contractions de $\mathcal{C}$ sont ordonnées par ordre croissant des indices. Pour tout i de 0 à $|\mathcal{C}|$, soit $\mathcal{C}_i$ l'ensemble des i premières contractions de $\mathcal{C}$. Il est important de noter ici que, puisque $poccur_s(f) = |u\,f|$ et que $Mot(u\,f, \mathcal{C}) = w\,f$, tous les mots intermédiaires donnés par les ensembles de contractions $\mathcal{C}_1, \mathcal{C}_2, \ldots$ seront de la forme $w_i\,f$. En effet, si par l'absurde on suppose l'occurrence de f est détruite par la $i^{ème}$ contraction, on remarque immédiatement que la partie détruite se trouve au début de l'occurrence de f. Il est alors facile de déduire qu'aucune des contractions ultérieures ne corrigera le préfixe détruit de f, puisque ces contractions ont des indices plus grands que la contraction destructrice.

Par induction sur la taille de l'ensemble $\mathcal{C}_i$, montrons que $Etat(w_i\,f) = Etat(f)$ quel que soit $f \in \mathcal{F}_s$ et $\mathcal{C}_i \subseteq \mathcal{C}$.
Soient $f \in \mathcal{F}_s$, $e_x = Etat(f)$ ($f = min(e_x)$ par définition de f) et $e_{x'_i} = Etat(w_i\,f)$. En considérant $\mathcal{C}_0$, on obtient trivialement $Mot(u\,f, \mathcal{C}_0) = u\,f$, donc $w_0\,f = u\,f$. Or d'après le lemme 5.4 – page ci-contre, $x'_0 \geq x$. En outre, selon le lemme 5.3 – page précédente – appliqué au mot $u\,f$, il découle que $x'_0 \leq poccur_s(u\,f)$. Pourtant, selon la définition de f, $poccur_s(f) = |u\,f| = poccur_s(u\,f)$. Il s'en déduit que $x'_0 \leq x$ et donc que $x'_0 = x$.
Montrons que si ce lemme est vérifié pour tout facteur remarquable de s et pour tout ensemble $\mathcal{C}_i \subset \mathcal{C}$ de contractions, il l'est également pour tout facteur remarquable et tout ensemble $\mathcal{C}_{i+1} \subseteq \mathcal{C}$. Soit (p, q), la dernière contraction de l'ensemble $\mathcal{C}_{i+1}$ (par ordre ascendant sur les positions). L'ensemble $\mathcal{C}_{i+1}$ s'écrit également $\mathcal{C}_i \cup \{(p, q)\}$. Soit b le facteur remarquable utilisé par (p, q). Le mot $u\,f$ peut s'écrire $u\,f = s[1..p-1]\,s[p..q-1]\,s[q..|u\,f|]$. Comme (p, q) est choisie comme étant la dernière contraction, toutes les contractions de $\mathcal{C}_i$ sont appliquées sur le mot $s[1..p-1]$. Il est alors possible de définir $a, c \in \Sigma^*$ tels que $w_i\,f = a\,s[p..|u\,f|] = a\,b\,c$ et $d \in \Sigma^*$ tel que $w_{i+1}\,f = a\,s[q..|u\,f|] = a\,b\,d$. Il s'en déduit que $a\,b = Mot(s[1..p-1]\,b, \mathcal{C}_i)$ et d'après l'hypothèse d'induction, $Etat(a\,b) = Etat(b)$. Ceci induit que $Etat(a\,b\,c) = Etat(b\,c)$ et que $Etat(a\,b\,d) = Etat(b\,d)$. Or le mot $b\,d$ ($= s[q..|u\,f|]$) est un suffixe du mot $b\,c$ ($= s[p..|u\,f|]$), et selon le lemme 5.5 – page ci-contre :

$$\begin{array}{lrcl} & Etat(b\,d) & \leq & Etat(b\,c) \\ \Leftrightarrow & Etat(a\,b\,d) & \leq & Etat(a\,b\,c) \\ \Leftrightarrow & Etat(w_{i+1}\,f) & \leq & Etat(w_i\,f) \\ \Leftrightarrow & Etat(w_{i+1}\,f) & \leq & Etat(f). \end{array}$$

De plus, le lemme 5.4 – page précédente – permet d'affirmer que $Etat(w_{i+1}\,f) \geq Etat(f)$ et donc que $Etat(w_{i+1}\,f) = Etat(f)$. Le lemme 5.6 – page ci-contre – est donc avéré pour tout $\mathcal{C}_i \subseteq \mathcal{C}_s^*$ et pour tout facteur remarquable. □

Le lemme suivant permet d'établir la correspondance entre les contractions et les transitions externes.

Lemme 5.7. *Soit un mot s et son Oracle, un entier i ($1 \leq i \leq |s|$) tel que $\#_{out}(e_i) > 1$ et le facteur $f = min(e_i)$. Alors, toute transition externe p issue de e_i d'étiquette α vers un état e_{i+j} ($j > 1$) est telle que $s[i + j - |f|..i + j] = f\,\alpha$ et $poccur_s(f\,\alpha) = i + j$. En conséquence, f est un facteur remarquable*

de s dont la première occurrence est à la position $i-|f|+1$ et la contraction $(i-|f|+1, i-|f|+j)$ de s par f existe.

Démonstration. D'après l'algorithme de construction de l'Oracle (*cf.* Algorithme 5.1 – page 145), la transition p de l'état e_i vers l'état e_{i+j} par α a été ajoutée (ligne 19) parce que $poccur_{s[i-|f|+1..|s|]}(f\,\alpha) - |f| = j$ (ligne 18). En outre, le mot $f\,\alpha$ est un facteur de s puisque $f\,\alpha \in Fact(s[i-|f|+1..|s|])$ (ligne 17). La première occurrence de $f\,\alpha$ dans s ne saurait apparaître avant la première occurrence de f dans s (*cf.* Lemme 5.1 – page 146), donc la première occurrence de $f\,\alpha$ dans s apparaît nécessairement après la position $i-|f|+1$, et coïncide par conséquent (à la translation des indices près) avec la première occurrence de $f\,\alpha$ dans $s[i-|f|+1..|s|]$. Il s'en déduit que $poccur_s(f\,\alpha) = i-|f| + poccur_{s[i-|f|+1..|s|]}(f\,\alpha)$, et donc que $poccur_s(f\,\alpha) = i+j$. Par conséquent, f possède au moins deux occurrences ($s[i-|f|+1..i]$ et $s[i-|f|+j..i+j-1]$) sur s. Vu que $f \in \mathcal{F}_s$ par la définition de $\mathcal{F}_s$ et parce que $\#_{out}(e_i) > 1$, l'existence de la contraction $(i-|f|+1, i-|f|+j)$ se déduit. □

5.3 Caractérisation du langage engendré par les Oracles

La caractérisation du langage reconnu par l'Oracle d'un mot s est basée en partie sur l'algorithme *Contractor* (*cf.* Algorithme 5.2 – page suivante). Cet algorithme consiste à déterminer, à partir d'un mot s, de son Oracle des Suffixes, et d'un mot w accepté par cet Oracle, un ensemble cohérent de contractions $\mathcal{C} \subseteq \mathcal{C}_s^*$ tel que w est un suffixe de $Mot(s, \mathcal{C})$. En pratique, l'algorithme ne prend pas en entrée le mot s, mais le plus grand suffixe v de $s = u\,v$ commençant par le premier symbole de w. Le fonctionnement de l'algorithme est décrit en détail par la suite, y compris avec un exemple.

5.3.1 Fonctionnement de l'algorithme Contractor

Dans cette section, le déroulement de l'algorithme est décrit informellement. Les preuves qui assurent qu'il est correct seront fournies dans la section suivante.

Selon la définition 5.11 – page 148, étant donnés un mot $s \in \Sigma^*$ tel que $s = u\,v$ ($u, v \in \Sigma^*$) et un ensemble $\mathcal{C} \subseteq \mathcal{C}_{s,v}^*$ de contractions, le mot $u\,w = Mot(s, \mathcal{C})$ est alors une concaténation de sous-mots de s. Ces sous-mots peuvent alors être vus comme des préfixes de suffixes de s, une contraction étant alors une concaténation de deux tels préfixes successifs. L'idée principale de l'algorithme *Contractor* consiste à parcourir les mots v et w de la gauche vers la droite, afin de calculer – à une étape précise de ce parcours – le plus long préfixe commun à deux suffixes donnés de v et de w (ligne 10) ; ce préfixe n'est rien d'autre que l'un des sous-mots concaténés, et correspond donc à une contraction qui sera elle aussi déterminée par l'algorithme. L'état de reconnaissance du préfixe (ligne 11) dans l'Oracle de s et le plus petit mot reconnu à cet état (ligne 12) sont nécessaires pour l'obtention de la contraction permettant d'atteindre soit les suffixes suivants de w et de v à considérer (lignes 14, 15, 16 et 17), soit la fin des deux mots v et w (ligne 21).

L'algorithme est récursif, le paragraphe ci-dessous, ainsi que la figure 5.8, illustrent une itération de *Contractor*. Les entrées de l'algorithme sont les mots S_i, S_i^w ($i \geq 0$) et un ensemble $\mathcal{C}_i$ de contractions. Au premier appel, les entrées sont donc initialisées avec $S_0 = v$, $S_0^w = w$, $\mathcal{C}_0 = \emptyset$ et $sdec = |u|$. La variable p_i étant le plus long préfixe commun à S_i et S_i^w (ligne 10), il est possible d'écrire :

$$S_i = p_i\,S_i' \qquad \text{et} \qquad S_i^w = p_i\,S_i'^w. \tag{5.1}$$

```
1  Nom : Contractor
2  Initialisation : lors du premier appel S_0 = v, S_0^w = w, C_0 = ∅, sdec = |u|
3    % s = u v tel que v est le plus grand suffixe de s commençant par w|1]. %
4  Entrées : S_i ∈ Σ* % Un suffixe de s devant encore être "contracté". %
5    S_i^w ∈ Σ* % Le suffixe de w qui est visé par les contractions sur S_i. %
6    C_i % L'ensemble de contractions déjà identifiées. %
7  Sortie : C_{i+1} % Ensemble de contractions applicables à s, mis à jour. %
8
9  Début
10   p_i ← plus grand préfixe commun à S_i et S_i^w
11   e_{q_i} ← Etat(p_i)
12   f_i ← min(e_{q_i})
13   Si (|p_i| < |S_i^w|) Alors
14     e_{r_i} ← état d'arrivée de la transition de e_{q_i} par α = S_i^w[|p_i| + 1]   (cf. Propriété 5.8 – page 155, item 3)
15     c_i ← (q_i − |f_i| + 1, r_i − |f_i|), C_{i+1} ← C_i ∪ {c_i}                    (cf. Propriété 5.9 – page 157, item 2)
16     S_{i+1}^w ← S_i^w[|p_i| − |f_i| + 1..|S_i^w|]                                  (cf. Propriété 5.8 – page 155, item 6)
17     S_{i+1} ← v[r_i − |f_i| − sdec..|v|]                                           (cf. Propriété 5.8 – page 155, item 6)
18     Retourner Contractor(S_{i+1}, S_{i+1}^w, C_{i+1})
19   Sinon
20     Si (|S_i| > |S_i^w|) Alors
21       c_i ← (q_i − |f_i| + 1, |s| − |f_i| + 1), C_{i+1} ← C_i ∪ {c_i}             (cf. Propriété 5.10 – page 158)
22     Sinon
23       C_{i+1} ← C_i                                                              (cf. Propriété 5.10 – page 158)
24     Fin Si
25     Retourner C_{i+1}
26   Fin Si
27 Fin
```

Algorithme 5.2 – Récupération d'un ensemble de contractions qui transforment $s = u\,v$ en $u\,w$.

Soient $e_{q_i} = Etat(p_i)$ et $f_i = min(e_{q_i})$ (lignes 11 et 12). Le lemme 5.2 – page 150 – permet d'affirmer que f_i est un suffixe de p_i, ce qui peut s'écrire :

$$p_i = p'_i\, f_i \quad (p'_i \in \Sigma^*). \tag{5.2}$$

Deux cas se profilent alors lors de cette itération (ligne 13). Le cas $|p_i| = |S_i^w|$ marque la fin de la récursion, puisque le parcours de w est terminé. Le cas contraire ($|p_i| \neq |S_i^w|$) implique qu'au moins une autre contraction est nécessaire afin d'obtenir $|p_j| = |S_j^w|$ ($j < i$). La variable e_{r_i} (ligne 14) identifie l'état d'arrivée de la transition issue de e_{q_i} et étiquetée par $\alpha = S_i^w[|p_i| + 1] = S_i'^w[1]$ (*cf.* Figure 5.8). L'ensemble $\mathcal{C}_{i+1}$ de contractions est de cardinalité $i + 1$. Quant à la constante $sdec = |u|$, elle est nécessaire afin de déterminer le mot S_{i+1} (ligne 17). En effet, chaque état e_i est associé au $i^{ème}$ caractère du mot s, et donc au caractère $(i - sdec)$ du suffixe v.

Exemple 5.5 (Exécution de l'algorithme *Contractor*).
Soit le mot $s = gaccattctc$. L'Oracle de s (cf. Figure 5.4) reconnaît $w = acatc$. Le plus grand suffixe de s commençant par $w[1] = a$ est $v = accattctc$ (donc $u = g$). Les mots v et w vont être parcourus simultanément, en appliquant si nécessaire des contractions au mot $s = u\,v$ afin de lire la même suite de caractères que celle composant le mot $u\,w$. L'algorithme ne s'arrêtera que lorsque le mot w aura été entièrement parcouru et reconnu. Les contractions données par Contractor *se basent sur les positions dans*

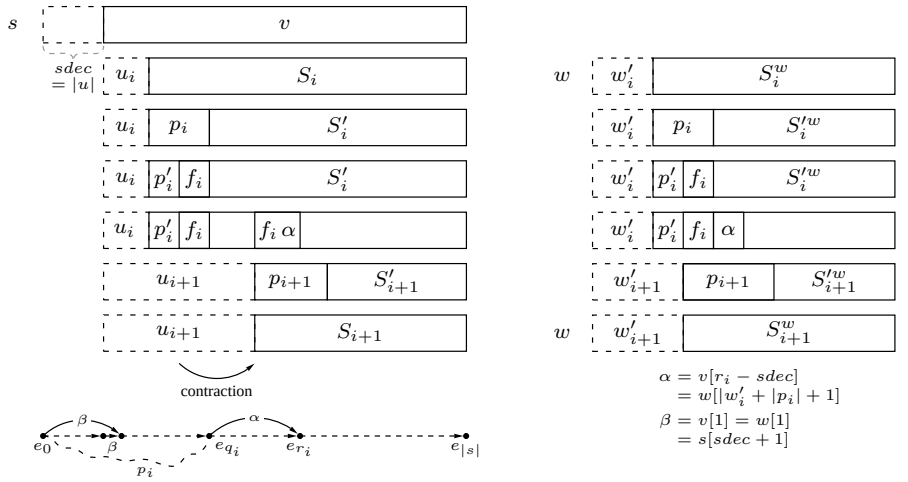


Figure 5.8 – Visualisation du déroulement de l'algorithme *Contractor* sur les mots S_i et S_i^w.

v, et sont calculées avec les indices des états, qui correspondent aux positions dans s, d'où l'intervention de la constante $sdec$.

Sur cet exemple l'algorithme est donc initialisé avec

$$\begin{cases} S_0 &= gaccattctc \quad (\Rightarrow sdec = |g| = 1) \\ S_0^w &= acatc \end{cases}$$

Rappelons ici que les caractères grisés ne font pas partie des mots, mais qu'il est utile de les voir pour mieux comprendre le déroulement de l'algorithme.

La variable S_i^w ($i \geq 0$) permet d'identifier le suffixe de w qui n'a pas encore été traité dans w après la $i^{ème}$ itération, autrement dit ce qu'il reste à lire dans le mot w. À la première itération, le plus long préfixe commun aux mots S_0 et S_0^w est $p_0 = a\,c$; l'état $e_{q_0} = e_3$, d'où $f_0 = min(e_{q_0}) = c$. Dans le cas présent, $|p_0| < |S_w^0|$ (le contraire signifierait que la fin du mot w est atteinte). Donc $e_{r_0} = e_5$ (transition de $e_{q_0} = e_3$ par a, cf. Figure 5.8). La contraction $c_1 = (3, 4)$ est alors ajoutée à $\mathcal{C}_0$ (l'application de c_1 à s donne le mot $gaccattctc$). À l'itération suivante, les mots en entrées sont $S_1 = accattctc$ et $S_1^w = acatc$. Les autres variables sont initialisées, $p_1 = cat$, $e_{q_1} = e_6$, $f_1 = t$, et puisque $|p_1| < |S_1^w|$, alors $e_{r_1} = e_8$ (transition de $e_{q_1} = e_6$ par c). La contraction $c_2 = (6, 7)$ est par conséquent ajoutée à $\mathcal{C}_1$ pour obtenir $\mathcal{C}_2$ et la troisième itération est initialisée avec $S_2 = cattctc$ et $S_2^w = catc$. Ainsi $p_2 = tc$, $e_{q_2} = e_8$ et $f_2 = tc$. Ici $|p_2| = |S_2^w|$ et $|S_2| > |S_2^w|$, ce qui signifie que w a été lu en intégralité, mais qu'une ultime contraction est nécessaire afin de lire le reste de S_2 (cela correspond au cas où w est accepté dans un état terminal autre que $e_{|s|}$, cf. Propriété 5.10 – page158). Cette dernière contraction est donc $(7, 9)$. L'ensemble des contractions ainsi obtenu est $\mathcal{C} = \mathcal{C}_3 = \{(3, 4), (6, 7), (7, 9)\}$. L'ensemble $\mathcal{C}$ est cohérent, et son application au mot $gaccattctc$ produit le mot $Mot(gaccattctc, \mathcal{C}) = gaccattctc$.

5.3.2 Étude de la validité de l'algorithme

Après avoir illustré le fonctionnement de l'algorithme, il est nécessaire de s'assurer de sa validité ; *i.e.*, vérifier que *Contractor* renvoie, à partir des mots $s = u\,v$ et w donnés (v étant le plus long suffixe de s tel que $v[1] = w[1]$, et w étant un mot reconnu par l'Oracle des Suffixes de s), un ensemble cohérent

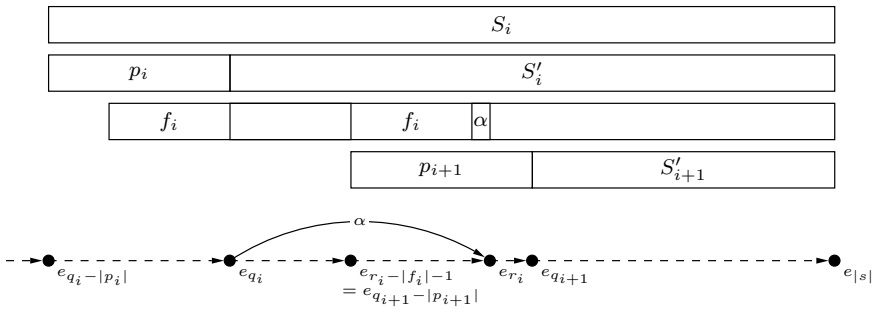


Figure 5.9 – Détail de la relation entre l'Oracle et les suffixes de s pour une itération de *Contractor*.

de contractions $\mathcal{C}$ tel que $Mot(s, \mathcal{C}) = u\,w$. Pour cela, plusieurs propriétés de l'algorithme doivent préalablement être établies. Les notations utilisées sont celles de l'algorithme *Contractor* et de la section précédente.

Propriété 5.8. *Les propositions suivantes sont vérifiées pour toute étape $i \geq 0$ de l'algorithme* Contractor *:*

1. $|p_i| > 0$ *;*
2. *si* $|p_i| < |S_i^w|$*, alors* $poccur_s(p_i) = q_i$*. De plus, si* $i > 0$ *alors* $q_i - |p_i| = r_{i-1} - |f_{i-1}| - 1$ *;*
3. *si* $|p_i| < |S_i^w|$*, alors il existe nécessairement une transition externe issue de l'état* e_{q_i} *vers un état* e_{r_i} *étiquetée par le symbole* α*. Par conséquent,* f_i *est un facteur remarquable de* s *et la contraction* $c_i = (q_i - |f_i| + 1, r_i - |f_i|)$ *existe. De plus,* $poccur_s(f_i\,\alpha) = r_i$ *;*
4. *si* $|p_i| < |S_i^w|$*, alors* $f_i\,\alpha \in Pref(p_{i+1})$ *;*
5. $S_i = v[q_i - |p_i| + 1 - sdec..|v|]$ *et* $S'_i = v[q_i + 1 - sdec..|v|]$ *;*
6. *si* $|p_i| < |S_i^w|$*, alors le mot* S_{i+1} *est un suffixe propre de* S_i *et le mot* S_{i+1}^w *est un suffixe de* S_i^w*. Les mots* S_i *et* S_i^w *(*$i \geq 0$*) sont respectivement suffixes des mots* v *et* w *;*
7. *le chemin dans l'Oracle des Suffixes commençant à l'état* e_0 *et étiqueté par le mot* S_0 *se termine à l'état* $e_{|s|}$ *et n'utilise que des transitions internes à l'exception de la première transition, menant de* e_0 *à* e_{sdec+1}*. Si* $|p_i| < |S_i^w|$*, alors le chemin dans l'Oracle des Suffixes commençant à l'état* e_0 *et étiqueté par le mot* S_{i+1} *se termine à l'état* $e_{|s|}$ *et n'utilise que des transitions internes à partir de l'état* e_{r_i}*.*

Cette propriété est illustrée par les figures 5.8 et 5.9.

Démonstration. La preuve utilise l'induction sur i globalement pour tout les items de 1 à 7. La condition $|p_i| < |S_i^w|$ correspond à la première alternative (ligne 13) de l'algorithme, et traduit donc l'existence d'une prochaine itération.

Cas $i = 0$.

1. L'affirmation est évidente puisque $v[1] = w[1]$ par l'hypothèse.

2. Puisque $i = 0$, on a $S_i = v$ et v est le plus long suffixe de s commençant par le premier symbole de w. Le mot $S_0[1]$ est de longueur 1. Il existe donc nécessairement un unique état e_x de l'Oracle de s tel que $min(e_x) = S_0[1]$. Selon le lemme 5.1 – page 146, $x = poccur_s(S_0[1])$. Par construction de S_0, la première occurrence de $S_0[1]$ se produit à la position $sdec + 1$, donc $x = sdec + 1$. Il en découle que $Etat(p_0) = e_{x+|p_0|-1} = e_{q_0}$ et que $poccur_s(p_0) = q_0$.

3. Le mot S_1^w est un suffixe de $S_0^w = w$ par construction (ligne 16). Comme w est reconnu par l'Oracle des Suffixes de s, tous ses suffixes, dont le mot S_0^w, sont également reconnus (*cf.* Lemme 5.5 – page 150). L'équation 5.1 – page 152 – avec $S_0'^w[1] = \alpha$ implique l'existence d'une transition de l'état e_{q_0} vers l'état e_{r_0} étiquetée par α. Cette transition est forcément externe. Dans le cas contraire, on aurait $s[q_0 + 1] = \alpha$ et donc $s[q_0 - p_0 \dots q_0 + 1] = p_0\alpha$ (*cf.* item 2). Par conséquent, $p_0\alpha$ serait un préfixe commun de S_0 et de S_0^w plus long que p_0, contradiction. La transition est donc externe ; le lemme 5.7 – page 151 – assure que $f_0 = min(e_{q_0})$ est un facteur remarquable et que la contraction c_0 existe.

4. De nouveau par le lemme 5.7 – page 151 – on obtient $s[r_0 - |f_0|..r_0] = v[r_0 - |f_0| - sdec..r_0 - sdec] = f_0\,\alpha$. Il en découle que le mot S_1 a pour préfixe $f_0\,\alpha$ (ligne 17). Le mot S_1^w a pour préfixe f_0 (ligne 16). Or le symbole $S_0^w[|p_0|+1] = \alpha$ (ligne 14) se trouve à la position $|f_0|+1$ dans S_1^w. Donc $f_0\,\alpha$ est également préfixe de S_1^w.

5. Comme pour l'item 2, il s'en déduit que $Etat(p_0) = e_{x+|p_0|-1} = e_{q_0}$, la reconnaissance de p_0 étant réalisée le long de la transition issue de e_0 et étiquetée $S_0[1]$ suivie de $|p_0| - 1$ transitions internes successives. Il en découle que $S_0 = s[q_0 - |p_0| + 1..|s|] = v[q_0 - |p_0| + 1 - sdec \dots |v|]$. Concernant le mot S_0', par construction $S_0' = S_0[|p_0| + 1..|S_0|]$, donc $S_0' = v[q_0 + 1 - sdec..|v|]$.

6. Le mot S_1^w est un suffixe de S_0^w par construction (ligne 16) ; or $S_0^w = w$, ce qui prouve une partie de l'affirmation. Ensuite, $S_1 = v[r_0 - |f_0| - sdec..|v|]$ (ligne 17) et $S_0 = v[q_0 - |p_0| + 1 - sdec..|v|]$ (item 5). Comme la transition entre e_{q_0} et e_{r_0} existe (item 3), et que f_0 est suffixe de p_0, les deux inégalités $r_0 > q_0$ et $|f_0| \leq |p_0|$ sont vérifiées. Finalement, $r_0 - |f_0| > q_0 - |f_0| \geq q_0 - |p_0|$, ce qui induit que S_1 est un suffixe propre de S_0.

7. Comme on l'a déjà vu, le mot $S_0 = v$ est construit en choisissant le plus long suffixe de s commençant par $w[1]$. Ainsi $poccur_s(S_0[1]) = sdec+1$, et par construction, l'arête issue de e_0 étiquetée par $S_0[1]$ aboutit à l'état e_{sdec+1} (*cf.* Figure 5.8). Le plus long chemin menant de cet état à $e_{|s|}$ est unique et composé uniquement de transitions internes. Il est de longueur $|s| - sdec - 1$, soit $|v| - 1$. Donc le mot S_0 étiquette le chemin composé de la transition menant de e_0 vers e_{sdec+1}, puis des transitions internes menant de l'état e_{sdec+1} à l'état $e_{|s|}$. Maintenant, supposons que $|p_0| < |S_0^w|$. Le mot $f_0\,\alpha$ est préfixe de S_1 (item 4). Le facteur f_0 est reconnu à l'état e_{q_0} (ligne 12), et la transition de e_{q_0} étiquetée par α mène à l'état e_{r_0} (item 3). Le plus long chemin menant de e_{r_0} à $e_{|s|}$ est unique et correspond au chemin composé uniquement de transitions internes, de longueur $|s| - r_0$. De plus, $|S_1| - |f_0\,\alpha| = |s| - r_0$ (ligne 17), donc le chemin étiqueté par S_1 n'est composé que de transitions internes à partir de l'état e_{r_0}.

Cas général $i > 0$.
Les items de 1 à 7 sont supposés vrais pour $i - 1$; leur preuve pour i est la suivante.

1. Se déduit immédiatement de l'item 4 pour $i - 1$.

2. Notons que $|p_i| < |S_i^w|$ implique $|p_{i-1}| < |S_{i-1}^w|$, sinon l'étape $i - 1$ aurait été terminale et l'étape i n'existerait plus. Maintenant, selon l'item 7 pour $i - 1$, le chemin dans l'Oracle des Suffixes commençant à l'état e_0 et étiqueté par le mot S_i se termine à l'état $e_{|s|}$ et n'utilise que des transitions internes à partir de l'état $e_{r_{i-1}}$. Donc p_i étiquette le sous-chemin de ce chemin qui commence à la position e_0 et se termine à la position e_{q_i}. De plus, par l'item 4, p_i contient $f_{i-1}\,\alpha'$ en tant que préfixe, où $\alpha' = S_{i-1}'^w[1]$ est le caractère similaire à α mais identifié à l'étape $i - 1$ (*cf.* Figures 5.8 et 5.9 en remplaçant i par $i - 1$ et α par α'). Puisque $f_{i-1}\,\alpha'$ est accepté en $e_{r_{i-1}}$, par la définition de $e_{r_{i-1}}$, on obtient que $p_i = f_{i-1}\,\alpha'\,s[r_{i-1} + 1 \dots q_i]$. Il est possible de conclure maintenant que la lecture de p_i s'effectue le long de s à partir de la position $r_{i-1} - |f_{i-1}|$ jusqu'à la position e_{q_i}, en utilisant l'item 3 pour $i-1$. Donc $|p_i| = q_i - r_{i-1} + |f_{i-1}| + 1$. Cette occurrence de p_i est la première, puisque l'occurrence de $f_{i-1}\,\alpha'$ à la position $r_{i-1} - |f_{i-1}|$ est également la première, toujours par l'item 3 pour $i - 1$.

3. Le mot S_{i+1}^w est un suffixe de S_i^w par construction (ligne 16), et S_i^w est un suffixe de S_{i-1}^w par

l'item 6 pour $i-1$. Ce dernier est à son tour un suffixe de w, par le même item, donc S_i^w est suffixe du mot w. Comme w est reconnu par l'Oracle des Suffixes de s, tous ses suffixes, dont le mot S_i^w, sont également reconnus (*cf.* Lemme 5.5 – page 150). L'équation 5.1 – page 152 – avec $S_i^w[1] = \alpha$ implique l'existence d'une transition de l'état e_{q_i} vers l'état e_{r_i} étiquetée par α. Cette transition est forcément externe. Dans le cas contraire, on aurait $s[q_i+1] = \alpha$ et donc $s[q_i - p_i \dots q_i + 1] = p_i\alpha$ (*cf.* item 2 pour i, déjà prouvé). Par conséquent, $p_i\alpha$ serait un préfixe commun de S_i et de S_i^w plus long que p_i, contradiction. La transition est donc externe ; le lemme 5.7 – page 151 – assure que $f_i = min(e_{q_i})$ est un facteur remarquable et que la contraction c_i existe.

4. De nouveau par le lemme 5.7 – page 151 – on obtient $s[r_i - |f_i|..r_i] = v[r_i - |f_i| - sdec..i_0 - sdec] = f_i\,\alpha$. Il en découle que le mot S_{i+1} a pour préfixe $f_i\,\alpha$ (ligne 17). Le mot S_{i+1}^w a pour préfixe f_i (ligne 16). Or le symbole $S_i^w[|p_i|+1] = \alpha$ (ligne 14) se trouve à la position $|f_i|+1$ dans S_{i+1}^w. Donc $f_i\,\alpha$ est également préfixe de S_{i+1}^w.

5. Le mot S_i est égal à $v[r_{i-1} - |f_{i-1}| - sdec..|v|]$ (ligne 17). De plus, item 2 pour i (déjà prouvé) assure que $q_i - |p_i| = r_{i-1} - |f_{i-1}| - 1$, d'où $S_i = v[q_i - |p_i| + 1 - sdec..|v|]$. Concernant le mot S'_i, par construction $S'_i = S_i[|p_i|+1..|S_i|]$, donc $S'_i = v[q_i + 1 - sdec..|v|]$.

6. S_{i+1}^w est par définition un suffixe de S_i^w. Par item 6 pour $i-1$, S_i^w est un suffixe de S_{i-1}^w qui est suffixe de w. Par conséquent, S_i^w est suffixe de w. Pour la partie de l'item qui concerne S_{i+1}, remarquons déjà que S_i est un suffixe du mot v par l'item 6 pour $i-1$. Ensuite, $S_i = v[q_i - |p_i| + 1 - sdec..|v|]$ (item 5) et $S_{i+1} = v[r_i - |f_i| - sdec..|v|]$ (ligne 17). Comme la transition entre e_{q_i} et e_{r_i} existe (item 3), et que f_i est suffixe de p_i, les deux inégalités $r_i > q_i$ et $|f_i| \leq |p_i|$ sont vérifiées. Finalement, $r_i - |f_i| > q_i - |p_i|$, ce qui induit que S_{i+1} est un suffixe propre du mot S_i.

7. Le mot $f_i\,\alpha$ est préfixe de S_{i+1} (item 4). Le facteur f_i est reconnu à l'état e_{q_i} (ligne 12), et la transition de e_{q_i} étiquetée par α mène à l'état e_{r_i} (item 3). Le plus long chemin menant de e_{r_i} à $e_{|s|}$ est unique et correspond au chemin composé uniquement de transitions internes, de longueur $|s| - r_i$. De plus, $|S_{i+1}| - |f_i\,\alpha| = |s| - r_i$ (ligne 17), donc le chemin étiqueté par S_{i+1} n'est composé que de transitions internes à partir de l'état e_{r_i}. □

La propriété 5.8 – page 155 (item 6), l'équation 5.2 – page 153 – et la définition de S_{i+1}^w (ligne 16) permettent d'établir les égalités suivantes :

$$\left\{ \begin{array}{rcccll} v & = & u_i\,S_i & = & u_i\,p_i\,S'_i & (u_i \in \Sigma^*) \\ w & = & w'_i\,S_i^w & = & w'_i\,p'_i\,S_{i+1}^w & (w'_i \in \Sigma^*). \end{array} \right. \tag{5.3}$$

De la dernière égalité appliquée à la fois pour i et pour $i+1$ on obtient que $w = w'_i\,S_i^w = w'_{i+1}\,S_{i+1}^w$, qui implique successivement :

$$\left\{ \begin{array}{rcl} w'_{i+1} & = & w'_i\,p'_i \\ w'_{i+1}f_i & = & w'_i\,p_i \end{array} \right. \tag{5.4}$$

Ces égalités sont utilisées dans la preuve de la propriété suivante, laquelle est illustrée par les figures 5.8 et 5.10.

Propriété 5.9. *Pour toute étape $i \geq 0$ de l'algorithme* Contractor, *les propositions suivantes sont avérées :*

1. $Etat(w'_i\,p_i) = Etat(u_i\,p_i) = Etat(p_i) = e_{q_i}$;
2. *si $|p_i| < |S_i^w|$, alors c_i est une contraction de $s = u\,u_i\,S_i$ engendrée par le facteur remarquable f_i. Le résultat de cette contraction est le mot $u\,u_i\,p'_i\,S_{i+1}$;*

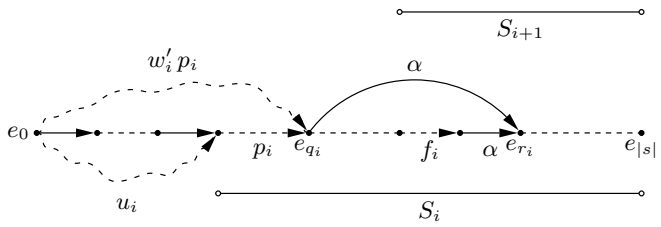


Figure 5.10 – Illustration d'une étape de l'algorithme *Contractor* ($\alpha = S_i^w[|p_i|+1]$).

3. si $|p_i| < |S_i^w|$, l'ensemble $\mathcal{C}_{i+1}$ est cohérent et $Mot(s, \mathcal{C}_{i+1}) = u\,w'_{i+1}\,S_{i+1}$.

Démonstration. 1. D'après l'item 7 de la propriété 5.8 – page 155, v est reconnu par l'Oracle par la transition issue de e_0 vers e_{sdec+1} puis par les arêtes internes menant à $e_{|s|}$. Le mot $u_i\,p_i$ est donc reconnu à l'état $e_{sdec+|u_i\,p_i|}$. La longueur de $u_i\,p_i$ est $|u_i\,p_i| = |v| - |S'_i|$, or d'après l'item 5 de la propriété 5.8 – page 155, $|v| - |S'_i| = q_i - sdec$, donc le mot $u_i\,p_i$ est reconnu à l'état $e_{q_i} = Etat(p_i)$. Il reste à montrer que $Etat(w'_i\,p_i) = e_{q_i}$. Procédons par induction sur i. Au rang $i = 0$, le mot $w'_0 = \varepsilon$ et donc $Etat(w'_0\,p_0) = e_{q_0}$ par construction (ligne 11). Supposons que $Etat(w'_i\,p_i) = e_{q_i}$ au rang i. Selon l'équation 5.4 – page précédente, $w'_{i+1}\,f_i = w'_i\,p_i$. D'après l'hypothèse d'induction, $Etat(w'_i\,p_i) = e_{q_i}$, et par construction, $Etat(f_i) = q_i$ (ligne 12), donc $Etat(w'_{i+1}\,f_i) = Etat(f_i)$. Soit $x_i \in \Sigma^*$ tel que $p_{i+1} = f_i\,\alpha\,x_i$ (*cf.* Propriété 5.8 – page 155, item 4). Il est trivial que $Etat(w'_{i+1}\,f_i\,\alpha\,x_i) = Etat(f_i\,\alpha\,x_i)$, ce qui peut s'écrire $Etat(w'_{i+1}\,p_{i+1}) = Etat(p_{i+1})$. Or par construction, $Etat(p_{i+1}) = e_{q_{i+1}}$ (ligne 10), complétant ainsi la preuve.

2. Selon l'item 3 de la propriété 5.8 – page 155, la contraction c_i par le facteur f_i existe dans s. D'autre part, l'équation 5.3 – page précédente – assure que $s = u\,u_i\,S_i$, donc la contraction donne le résultat $u\,u_i\,p'_i\,S_{i+1}$ (*cf.* Figure 5.8) selon la propriété 5.8 – page 155 (item 5) et la ligne 17 de l'algorithme *Contractor*.

3. Procédons par induction sur i. Au rang $i = 0$, $\mathcal{C}_0 = \emptyset$ (cohérent par définition), $S_0 = v$ et $w'_0 = \varepsilon$, donc $Mot(s, \mathcal{C}_0) = s = u\,w'_0\,S_0$. Supposons la propriété vraie au rang $i-1$. En écrivant $s = u\,u_i\,S_i$, d'après l'hypothèse d'induction, $Mot(u\,u_i\,S_i, \mathcal{C}_i) = u\,w'_i\,S_i$. Les contractions de l'ensemble $\mathcal{C}_i$ sont (par construction) appliquées au préfixe $u\,u_i$ de s. Cela implique que toutes les contractions $(k,l) \in \mathcal{C}_i$ sont telles que $l \leq |u\,u_i|$. En outre, il vient d'être montré (item 2) que $Mot(u\,u_i\,S_i, \{c_i\}) = u\,u_i\,p'_i\,S_{i+1}$. La contraction c_i est appliquée sur le suffixe S_i de s, donc $q_i - |f_i| + 1 > |u\,u_i|$. L'ensemble $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{c_i\}$ est donc cohérent. En appliquant les contractions de $\mathcal{C}_i$ au préfixe $u\,u_i$ et la contraction c_i au suffixe S_i, il se déduit que $Mot(u\,u_i\,S_i, \mathcal{C}_i \cup \{c_i\}) = u\,w'_i\,p'_i\,S_{i+1}$, et donc, par l'équation 5.4 – page précédente, que $Mot(s, \mathcal{C}_{i+1}) = u\,w'_{i+1}\,S_{i+1}$. □

Les propriétés 5.8 – page 155 – et 5.9 – page précédente – sont valables dans leur intégralité pour tout i tel que $|p_i| < |S_i^w|$, c'est-à-dire hormis lors de la dernière itération, qui correspond au cas d'arrêt de la récursion. Il est donc nécessaire de considérer le cas d'arrêt.

Propriété 5.10. *L'algorithme* Contractor *se termine. Il existe donc un entier $i \geq 0$ tel que $p_i = S_i^w$, et si $|S_i^w| \neq |S_i|$, alors la contraction $c_i = (q_i - |f_i| + 1, |s| - |f_i| + 1)$ existe.*

Démonstration. Si $v = w$, alors $p_0 = S_0 = S_0^w$, et donc l'algorithme s'arrête dès la première itération. Dans le cas contraire, il s'agit de montrer que la suite des entiers $k \geq 0$ tels que l'étape k est non terminale, est finie. Pour tous ces entiers k, la condition $|p_k| < |S_k^w|$ est vérifiée. Avec la notation

$U_k = |S_k^w| - |p_k|$ ($k \geq 0$), on obtient en utilisant successivement la ligne 16 de l'algorithme *Contractor* et la propriété 5.8 – page 155 (item 4) :

$$\begin{array}{rcl} U_k & = & |S_k^w| - |p_k| \quad = \quad |S_{k+1}^w| + (|p_k| - |f_k|) - |p_k| \\ & = & |S_{k+1}^w| - |f_k| \quad > \quad |S_{k+1}^w| - |p_{k+1}| \\ & = & U_{k+1} \end{array} \tag{5.5}$$

La suite d'entiers positifs U_k est donc strictement décroissante, par conséquent elle est finie. Le nombre d'étapes non-terminales l'est donc également. Il existe donc i tel que $U_i = 0$, provoquant l'arrêt de la récursion avec $p_i = S_i^w$. Dans ce cas, l'algorithme *Contractor* poursuit à la ligne 19.
Si $|S_i^w| \neq |S_i|$, comme $S_i^w = p_i$ et p_i est un préfixe de S_i, cela signifie que $|S_i^w| < |S_i|$. De plus, comme p_i est un préfixe propre de S_i il s'en suit que $Etat(p_i) \neq Etat(S_i)$, c'est à dire $Etat(p_i) \neq e_{|s|}$, selon la propriété 5.8 – page 155 (item 7). Par conséquent, $Etat(w) = Etat(w'_i\, S_i^w) = Etat(w'_i\, p_i)$ par l'équation 5.3 et puisque $S_i^w = p_i$, alors que $Etat(w'_i\, p_i) = e_{q_i} \neq e_{|s|}$, par la propriété 5.9 – page 157 (item 3) et l'inégalité $Etat(p_i) \neq e_{|s|}$ établie plus haut. Il s'en déduit que w est reconnu par l'Oracle des Suffixes de s à l'état e_{q_i}, lequel est donc final et différent de $e_{|s|}$. Il existe par conséquent un suffixe de s reconnu en e_{q_i}. Or $f_i = min(e_{q_i})$ est suffixe de tous les mots reconnus à l'état e_{q_i}, donc f_i est également suffixe de s. Ainsi, $f_i \in \mathcal{F}_s$, avérant par définition l'existence de la contraction $c_i = (Pos_s(f_i), |s| - |f_i| + 1) = (q_i - |f_i| + 1, |s| - |f_i| + 1)$. □

Lemme 5.11. *Soient un mot $s \in \Sigma^*$, son Oracle des Suffixes et un mot $w \in \Sigma^*$ reconnu par $SO(s)$. Notons par v le plus long suffixe de s commençant par le premier symbole de w et supposons que $s = u\,v$. Alors l'appel de* Contractor$(v, w, \emptyset)$ *renvoie un ensemble cohérent de contractions $\mathcal{C}$ tel que $u\,w = Mot(u\,v, \mathcal{C})$.*

Démonstration. Si $v = w$, alors $p_0 = S_0 = S_0^w$ et l'algorithme se termine avec $\mathcal{C} = \emptyset$ et l'énoncé est vérifié.
Dans le cas contraire, soit $i \geq 0$ tel que $|S_{i+1}^w| = |p_{i+1}|$ (*cf.* Propriété 5.10 – page précédente). Alors $|p_i| < |S_i^w|$ (*cf.* Figure 5.8).
Si l'égalité $|S_{i+1}^w| = |S_{i+1}|$ est vérifiée, cela implique (ligne 10) que $S_{i+1}^w = S_{i+1}$ et que $\mathcal{C}_{i+2} = \mathcal{C}_{i+1}$ (ligne 23). Il s'en déduit que $\mathcal{C}_{i+2}$ est cohérent et que $Mot(s, \mathcal{C}_{i+2}) = Mot(u\,u_{i+1}\,S_{i+1}, \mathcal{C}_{i+1}) = u\,w'_{i+1}\,S_{i+1} = u\,w'_{i+1}\,S_{i+1}^w = u\,w$ (*cf.* Propriété 5.9 – page 157, item 3 et Équation 5.3 – page 157).
Si au contraire $|S_{i+1}^w| \neq |S_{i+1}|$, comme $|p_{i+1}| = |S_{i+1}^w|$ cela signifie que $|S_{i+1}^w| < |S_{i+1}|$, et que $S_{i+1}^w = p_{i+1}$. Selon la propriété 5.10 – page ci-contre, la contraction $c_{i+1} = (q_{i+1} - |f_{i+1}| + < 1, |s| - |f_{i+1}| + 1)$ existe. Son application au mot $u\,u_{i+1}\,S_{i+1}$ produit le mot $u\,u_{i+1}\,p'_{i+1}\,f_{i+1} = u\,u_{i+1}\,p_{i+1}$. Selon l'item 3 de la propriété 5.9 – page 157, l'ensemble $\mathcal{C}_{i+1}$ est cohérent et tel que $Mot(s, \mathcal{C}_{i+1}) = Mot(u\,u_{i+1}\,S_{i+1}, \mathcal{C}_{i+1}) = u\,w'_{i+1}\,S_{i+1}$. Les contractions de l'ensemble $\mathcal{C}_{i+1}$ sont (par construction) appliquées au préfixe $u\,u_{i+1}$ de s. Cela implique que toutes les contractions $(k, l) \in \mathcal{C}_{i+1}$ sont telles que $l \leq |u\,u_{i+1}|$. La contraction c_{i+1} est appliquée sur le suffixe S_{i+1} de s, donc $q_{i+1} - |f_{i+1}| + 1 > |u\,u_{i+1}|$, assurant alors la cohérence de l'ensemble $\mathcal{C}_{i+2} = \mathcal{C}_{i+1} \cup \{c_{i+1}\}$. Il s'en déduit également que $Mot(s, \mathcal{C}_{i+2}) = Mot(u\,u_{i+1}\,S_{i+1}, \mathcal{C}_{i+1} \cup \{c_{i+1}\}) = u\,w'_{i+1}\,p_{i+1} = u\,w$.
Finalement, l'ensemble de contractions $\mathcal{C} = \mathcal{C}_{i+2}$ renvoyé par l'algorithme *Contractor* est cohérent et tel que $Mot(u\,v, \mathcal{C}) = u\,w$. □

Remarque 5.14. L'ensemble $\mathcal{C}$ renvoyé par l'algorithme *Contractor* n'est pas nécessairement minimal. En effet, dans certains cas, il se peut que deux contractions (i_1, j_1), (i_2, j_2) soient telles que $j_1 = i_2$. Toutefois, cela n'a pas d'incidence sur les résultats présentés dans la suite.

5.3.3 Langage reconnu par les Oracles

Les deux théorèmes suivants établissent la relation entre la fermeture d'un mot et l'ensemble des facteurs/suffixes reconnus par l'Oracle de ce mot.

Théorème 5.12. *L'Oracle des Suffixes de s reconnaît tous (et seulement) les suffixes des mots de $\mathcal{E}(s)$.*

Démonstration.
'$\Rightarrow$' : *Tout suffixe d'un mot de la fermeture de s est reconnu par l'Oracle des Suffixes de s.*
D'après le lemme 5.5 – page 150, si un mot w est reconnu par l'Oracle des Suffixes de s, tous les suffixes de w sont également reconnus par cet Oracle. Il suffit alors de montrer que tous les mots de la fermeture de s sont acceptés par l'Oracle des Suffixes de s pour vérifier cette assertion.
Soit $\mathcal{C} \subseteq \mathcal{C}_s^*$ un ensemble cohérent et minimal de contractions applicable au mot s et $w = Mot(s, \mathcal{C})$. Dans $\mathcal{C}$, les contractions sont supposées ordonnées par ordre croissant de leurs indices : $\mathcal{C} = \{(p_1, q_1), (p_2, q_2), \ldots, (p_{|\mathcal{C}|}, q_{|\mathcal{C}|}\}$; le facteur remarquable correspondant à la contraction (p_i, q_i) est noté f_i. Soit $\mathcal{C}_i$ l'ensemble des i premières contractions de $\mathcal{C}$ et soit $w_i = Mot(s, \mathcal{C}_i)$. Montrons que la propriété suivante est vérifiée :

(P). *Si le mot w_i ($0 \leq i < |\mathcal{C}|$) est reconnu par l'Oracle des Suffixes de s, alors w_{i+1} l'est également.*

Par définition :

$$\begin{cases} w_i & = s[1..p_1 - 1]\, s[q_1..p_2 - 1] \ldots s[q_i..|s|], \\ s[q_i..q_i + |f_i| - 1] & = f_i. \end{cases}$$

Or, selon la définition des facteurs remarquables, f_{i+1} n'apparaît pas dans le mot s avant la position p_{i+1} ($p_{i+1} > q_i$). Ainsi les mots w_i et w_{i+1} s'écrivent respectivement $x\ f_{i+1}\, y$ et $x\, f_{i+1}\, z$ (par définition), avec $x = s[1..p_1 - 1]\, s[q_1..p_2 - 1] \ldots s[q_i..p_{i+1} - 1]$ et $f_{i+1}\, y = x'\, f_{i+1}\, z$ ($x' \in \Sigma^+$, *cf.* Figure 5.11). En considérant maintenant la contraction (p_{i+1}, q_{i+1}), il est possible d'écrire $|s| - |f_{i+1}\, y| + 1 = p_{i+1}$ et $|s| - |f_{i+1}\, z| + 1 = q_{i+1}$ (ceci est dû à l'ordre d'application des contractions, *cf.* Figure 5.11). Le mot s n'est donc par conséquent pas modifié par l'application de $\mathcal{C}_i$ après la position p_{i+1}. De ce constat découlent deux propriétés. Tout d'abord, cela implique que $Mot(s[1..p_{i+1} - 1]\, f_{i+1}, \mathcal{C}_i) = x\, f_{i+1}$, et d'après le lemme 5.6 – page 150, appliqué avec le facteur remarquable f_{i+1} et l'ensemble de contractions $\mathcal{C}_i$:

$$Etat(x\ f_{i+1}) = Etat(f_{i+1}). \tag{5.6}$$

La seconde propriété est que $f_{i+1}\, y$ et $f_{i+1}\, z$ sont tous deux suffixes de s, donc ils sont nécessairement reconnus par l'Oracle des Suffixes de s. Par conséquent, les chemins étiquetés par les suffixes $f_{i+1}\, y$ et $f_{i+1}\, z$ empruntent le chemin étiqueté par f_{i+1} menant à l'état $Etat(f_{i+1})$ pour atteindre par la suite un état final de l'Oracle des Suffixes. Donc y et z sont lus par l'Oracle des Suffixes à partir de l'état $Etat(f_{i+1})$. Il s'en suit, d'après l'équation 5.6, que l'Oracle des Suffixes de s reconnaît également le mot $w_{i+1} = x\, f_{i+1}\, z$. Finalement, la propriété **(P)** est avérée pour tout i ($0 \leq i < |\mathcal{C}|$).

Dans la mesure où $w_0 = s$, w_0 est reconnu par l'Oracle des Suffixes de s, ce qui montre par induction sur i que l'Oracle des Suffixes d'un mot s reconnaît tous les mots w_i ($0 \leq i \leq |\mathcal{C}|$). Donc $w = w_{|\mathcal{C}|}$ est également reconnu.
'$\Leftarrow$' : *Tout mot reconnu par l'Oracle des Suffixes de s est suffixe d'un mot de la fermeture de s.*
Soit w un mot accepté par l'Oracle des Suffixes de s et v le plus long suffixe de $s = u\, v$ ($u, v \in \Sigma^*$) commençant par le premier symbole de w. Alors il existe un ensemble cohérent de contractions $\mathcal{C}$ tel que $u\, w = Mot(u\, v, \mathcal{C})$ (*cf.* Lemme 5.11 – page précédente). Il est facile de voir que cet ensemble peut être rendu minimal sans changer le résultat de la contraction sur s. En outre, comme le mot w est suffixe de $u\, w$ et que $u\, w \in \mathcal{E}(s)$, il en découle que w est suffixe d'un mot de la fermeture de s. □

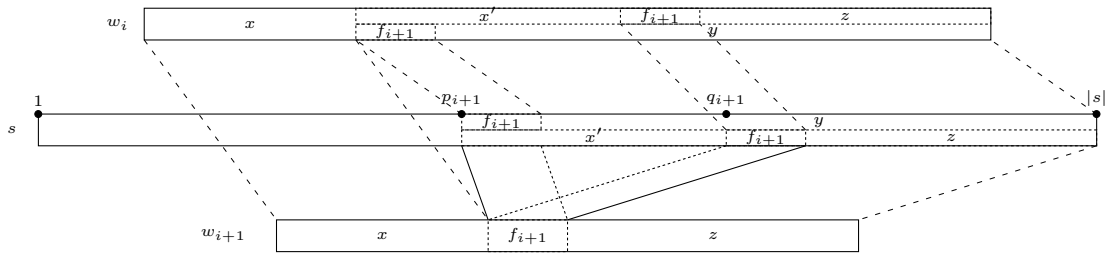


Figure 5.11 – Détail sur l'application des contractions.

Le second théorème découle assez directement du premier, et s'énonce de façon similaire.

Théorème 5.13. *L'Oracle des Facteurs de s reconnaît tous (et seulement) les facteurs des mots de $\mathcal{E}(s)$.*

Démonstration.
'$\Rightarrow$' : *Tout facteur d'un mot de la fermeture de s est reconnu par l'Oracle des Facteurs de s.*
Soient w un mot de la fermeture de s et m un facteur de w. Alors w peut s'écrire $w = u\,m\,v$ ($u, v \in \Sigma^*$) et le suffixe $m\,v$ de w est nécessairement reconnu par l'Oracle des Suffixes de s (*cf.* Théorème 5.12 – page ci-contre). Par conséquent, cela implique qu'il existe un chemin $(e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m\,v|}})$ dans l'Oracle des Suffixes de s correspondant au mot $m\,v$. Le mot m étiquette donc un chemin $(e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m|}})$ et est donc reconnu par l'Oracle des Facteurs de s.

'$\Leftarrow$' : *Tout mot reconnu par l'Oracle des Facteurs de s est facteur d'un mot de la fermeture de s.*
Soit m un mot reconnu par l'Oracle des Facteurs de s. Si ce mot est également reconnu par l'Oracle des Suffixes de s, cela signifie que m est un suffixe d'un mot de $\mathcal{E}(s)$ (*cf.* Théorème 5.12 – page précédente). Supposons maintenant que m ne soit pas reconnu par l'Oracle des Suffixes de s. Cela implique alors que m est reconnu par l'Oracle des Facteurs de s à l'état $e_{x_{|m|}}$ (qui n'est pas final dans l'Oracle des Suffixes de s) suivant un chemin $e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m|}}$. Il existe donc un chemin $(e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m|}} \rightarrow e_{x_{|m|}+1} \rightarrow \ldots \rightarrow e_{|s|})$ dans l'Oracle de s avec $e_{|s|}$ final. Ainsi, le mot m est préfixe d'un mot reconnu par l'Oracle des Suffixes de s, ce qui signifie que m est préfixe d'un suffixe d'un mot $w \in \mathcal{E}(s)$. Finalement, il s'en conclut que le mot m est un facteur d'un mot de la fermeture de s. □

5.4 Caractéristiques des Oracles

L'Oracle est minimal en nombre d'états (définition classique de la minimalité des automates, *cf.* Annexe D.2), parmi les automates reconnaissant un nombre fini de mots dont au moins tous les facteurs/-suffixes d'un mot donné (*cf.* Section 5.1.4 – page 143). Cependant, il n'est pas toujours minimal en nombre de transitions parmi les automates reconnaissant un nombre fini de mots dont au moins tous les facteurs/suffixes d'un mot donné [4]. Une des particularités de l'Oracle est sa propriété d'homogénéité, lui conférant un avantage de complexité spatiale lors de son implémentation. La question de sa minimalité en nombre de transitions parmi la classe des automates homogènes s'est légitimement posée, et dans la conclusion de leur article, CLEOPHAS *& al.* [28] montrent que même parmi les automates de cette classe, l'Oracle n'est pas toujours minimal en nombre de transitions. La section suivante montre comment cette observation peut être enrichie, en produisant des automates homogènes ayant moins de transitions et reconnaissant encore moins de faux positifs que les Oracles pour certains mots donnés. Sur la base de ce constat et des résultats de la section précédente, l'ordre de grandeur du nombre de faux positifs engendrés par les Oracle dans le pire des cas est établi.

5.4.1 Minimalité en nombre de transitions et en nombre de mots

Considérons l'ensemble des automates homogènes qui reconnaissent au moins tous les facteurs (respectivement suffixes) de s, et qui ont le même nombre d'états et au plus le même nombre de transitions que l'Oracle des Facteurs (respectivement Suffixes) de s. À l'intérieur de cet ensemble, l'Oracle n'est pas toujours minimal en nombre de mots acceptés, ni en nombre de transitions, ni même en cumulant ces deux critères. En effet, l'Oracle du mot $axttyabcdeatzattwu$ (*cf.* Figure 5.12) possède 35 transitions, l'Oracle des Facteurs accepte 247 mots et l'Oracle des Suffixes accepte 39 mots. Alors qu'il existe un autre automate homogène (*cf.* Figure 5.13), qui reconnaît au moins tous les facteurs (respectivement suffixes) de $axttyabcdeatzattwu$ et qui possède seulement 34 transitions. La version « Facteur » de cet automate reconnaît seulement 241 mots et sa version « Suffixe » accepte seulement 30 mots. Cet exemple montre qu'il existe un automate homogène qui possède à la fois moins de transitions et moins de faux positifs que l'Oracle correspondant, et cela pour les deux versions « Facteurs » et « Suffixes ».

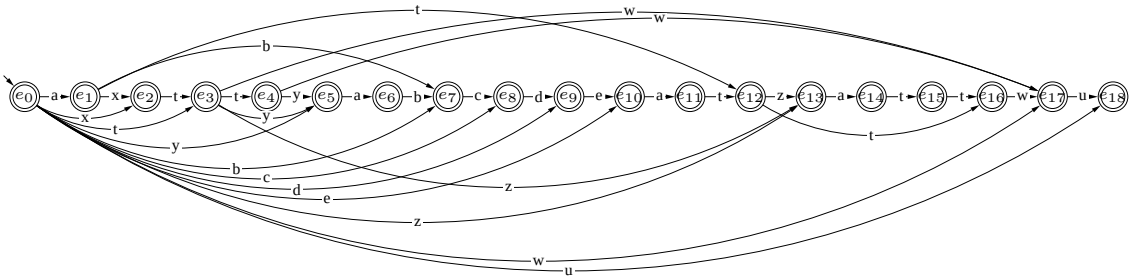


Figure 5.12 – Oracle des Facteurs du mot $axttyabcdeatzattwu$.

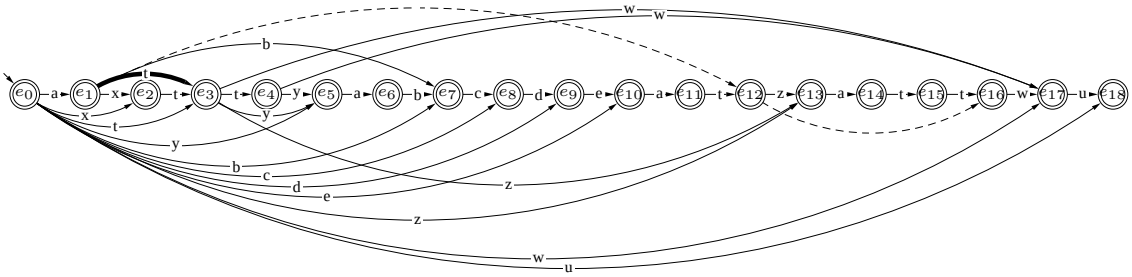


Figure 5.13 – Automate acceptant[1] tous les facteurs du mot $axttyabcdeatzattwu$.

Parmi les faux positifs des Oracles de $axttyabcdeatzattwu$, un certain nombre proviennent de la contraction $(11, 14)$ par le facteur remarquable at. Les deux transitions (e_1, t, e_{12}) et (e_{12}, t, e_{16}) sont nécessaires pour reconnaître les facteurs at, att, $attw$, $attwu$ ou le suffixe $attwu$ (selon l'Oracle) de $axttyabcdeatzattwu$. En les remplaçant par la transition (e_1, t, e_3), les facteurs at, att, $attw$, $attwu$ ou le suffixe $attwu$ sont également reconnus, mais le nombre de faux positifs engendrés par ce nouvel automate (qui demeure homogène) est moindre par rapport à l'Oracle. Il est possible de construire d'autres exemples permettant d'accentuer l'écart entre le nombre de faux positifs engendrés par les deux

[1] Seules les lignes continues appartiennent à cet automate. La transition en gras (de e_1 vers e_3) est la seule qui n'est pas présente dans l'Oracle des Facteurs de ce mot, alors que les deux transitions en pointillé (de e_1 vers e_{12} et de e_{12} vers e_{16}) sont présentes dans l'Oracle des Facteurs, mais pas dans cet automate.

automates, par exemple en insérant de nouveaux symboles après la $10^{ème}$ position. En effet, l'Oracle du mot $axttyabcdefghijatzattwu$ possède 45 transitions, et reconnaît 377 facteurs ou 49 suffixes selon la version, tandis que le même automate privé des transitions (e_1, t, e_{17}) et (e_{17}, t, e_{21}) et augmenté de la transition (e_1, t, e_3) possède 44 transitions et reconnaît 366 facteurs ou 35 suffixes (toujours selon la version). Il est également possible d'augmenter l'écart concernant le nombre de transitions, en construisant de plus grands exemples sur le même principe (*i.e.*, remplacement de n transitions par au plus $n-1$ autres transitions conservant les propriétés d'homogénéité et de reconnaissance des facteurs ou suffixes du mot d'origine).

5.4.2 Nombre de faux positifs

Plusieurs algorithmes exploitent les Oracles avec de bons résultats, comme l'algorithme de recherche de motif *Turbo*-BOM [4], la méthode d'extraction de motifs en tandem FORREPEATS [85], l'algorithme de compression de texte COMPROR [84], la méthode d'improvisation musicale en temps réel utilisée dans le logiciel *OpenMusic* [8] ou encore l'outil d'identification de protéines à partir de données de spectrométrie de masses[2] POPITAM [65]. Toutefois, toutes sont obligées de tenir compte de l'incertitude sur l'information apportée par les Oracles.

Bien que le nombre de faux positifs puisse dans certains cas être nul (*e.g.* $aaaaaa\ldots$), il peut aussi être exponentiel. De fait, il est possible de construire une famille de mots dont les Oracles associés possèdent un nombre exponentiel de faux positifs. Pour chaque mot s de cette famille, chaque sous-ensemble de $\mathcal{C}_s^*$ est cohérent et minimal. Soit, par exemple, la famille de tous les mots de type $aabbccddee\ldots$, et soit s un mot de cette famille, de longueur $|s|$ (forcément paire). L'ensemble de contractions $\mathcal{C}_s^*$ applicables à un tel mot est $\{(1,2),(3,4),\ldots,(|s|-1,|s|)\}$. Chaque sous-ensemble de contractions $\mathcal{C} \subseteq \mathcal{C}_s^*$ est cohérent et minimal, et hormis pour les sous-ensembles $\{\}$ et $\{(1,2)\}$, il est aisé de remarquer que $Mot(s,\mathcal{C}) \notin Fact(s)$. De plus, tous les mots obtenus à partir de ces sous-ensembles sont deux à deux différents.

Le nombre de ces sous-ensembles est :

$$\sum_{i=0}^{|\mathcal{C}_s^*|} \binom{|\mathcal{C}_s^*|}{i} - 2 = \sum_{i=0}^{\frac{|s|}{2}} \binom{\frac{|s|}{2}}{i} - 2 = 2^{\frac{|s|}{2}} - 2.$$

Par conséquent, dans le pire des cas, le nombre de mots acceptés par les Oracles qui ne sont pas facteurs/suffixes de s est dans $\Omega\left(\sqrt{2^{|s|}}\right)$.

5.5 Vers un nouvel Oracle

L'approche du langage reconnu à l'aide des facteurs remarquables et des contractions est très utile formellement pour arriver à une caractérisation de ce langage, mais ne permet pas a priori de répondre à une question simple telle : « Comment faire pour diminuer sérieusement le nombre de faux positifs reconnus par les Oracles ? » Pour essayer de donner une réponse à ce genre de questions, il faut se tourner vers les Oracles eux-mêmes et comprendre intuitivement la création d'un faux positif au fur et à mesure du parcours d'un chemin entre l'état initial de l'Oracle et l'un de ses états finaux. Avec

[2] Méthode permettant l'identification de molécules par l'analyse de ses différents constituants. Cette technique utilise les champs électriques et magnétiques afin de classer les molécules en fonction de leur rapport masse/charge.

une telle approche, on saisit l'importance des transitions externes qui parfois sont empruntées durant la lecture d'un mot qu'il serait souhaitable de ne pas lire. Les transitions externes sont donc en partie responsables de l'existence des faux positifs ; pour diminuer le nombre de faux positifs, il faut donc agir sur les transitions externes dans le but d'enrichir l'Oracle avec des informations supplémentaires, mais sans trop l'alourdir.

5.5.1 Oracle à transitions gardées

Soit l'Oracle d'un mot s et soit w un mot. La lecture de w par l'Oracle se fait à partir de l'état initial e_0 en suivant pas à pas les transitions. Supposons que, lorsqu'on arrive dans un état e_i après la lecture du préfixe w' de w, le caractère suivant α de w doit être lu le long d'une transition externe issue de e_i. Cette lecture ne devrait être autorisée que si $w'\,\alpha$ est un facteur de s ; mais ce test n'est pas réalisable rapidement à ce niveau. Il est néanmoins possible de trouver un autre critère qui limite sérieusement le nombre d'autorisations à ce niveau, tout en préservant la structure de l'Oracle et son caractère économique.

Il s'agit d'associer à chaque transition externe (issue, disons, de l'état e_i et ayant l'étiquette α) la longueur maximum des mots m tels que $Etat(m) = e_i$ et $m\,\alpha$ est un facteur de s. Cette condition est moins forte que le test précédemment identifié, mais elle permet en contrepartie une vérification aisée et rapide. L'entier associé à chaque transition externe sera appelée garde et la structure d'index ainsi obtenue s'appellera Oracle des Facteurs/Suffixes à transitions gardées.

Avant de détailler la méthode d'élaboration de ces gardes, il est nécessaire de revenir sur l'algorithme de construction de l'Oracle. En effet, l'algorithme 5.1 – page 145 – présenté précédemment n'est pas optimal. ALLAUZEN *& al.* ont élaboré un autre algorithme permettant de construire le même Oracle [3, page 9]. Cet algorithme est incrémental (*i.e.,* l'Oracle du mot $s = s_1 \dots s_{n+1}$ est construit à partir de l'Oracle de $s = s_1 \dots s_n$ et du symbole s_{n+1}) et sa complexité (spatiale et temporelle) est linéaire. Cet algorithme est généralement qualifié de version « on-line » dans la littérature.

Avant de le présenter, il est indispensable de définir une fonction dite de suppléance sur les états de l'automate, sur laquelle repose cette méthode, puisque cette fonction régit l'ajout des transitions externes.

Définition 5.15. Étant donnés un mot $s \in \Sigma^+$ et son Oracle, la fonction de suppléance $\mathcal{S}_s$ est définie comme étant la fonction qui identifie, pour chaque état e_i ($1 \leq i \leq |s|$) de l'Oracle de s, l'état de reconnaissance du plus long suffixe de $s[1..i]$ qui n'est pas reconnu en e_i.

Remarque 5.16. La fonction de suppléance est définie pour tout état e_i ($i > 0$), *i.e.,* pour tout état e_i ($i > 0$) il existe l'état e_j tel que $\mathcal{S}_s(e_i) = e_j$. De plus, $j < i$ par le lemme 5.5 – page 150 – et la définition de $\mathcal{S}_s$.

La définition et la remarque qui précédent sont issues de [3]. Les auteurs utilisent la fonction de suppléance pour définir une notion de « liens suffixes » correspondant à la relation entre $s[1..i]$ et $s[1..j]$ avec $e_j = \mathcal{S}_s(e_i)$ ($i > 0$). Attention toutefois à ne pas les assimiler avec leurs homonymes définis dans les arbres des suffixes. Ils n'ont pas tout à fait la même signification. Il est possible d'étendre la notion de « lien suffixe » dans les Oracles par la notion de classes de suffixes.

Définition 5.17. Étant donnés un mot $s \in \Sigma^+$ et son Oracle, l'ensemble $Suff(s)$ peut être partitionné de manière à ce que tous les suffixes d'une même partie soient reconnus dans le même état de l'Oracle de s. Chaque partie est alors une classe de suffixes de s.

En effet, soit s un mot et soit e_i l'état d'acceptation de l'un des suffixes de s. Soit $e_j = \mathcal{S}_s(e_i)$. Par définition, $s[1..j]$ est le plus grand suffixe de $s[1..i]$ qui ne soit pas reconnu à l'état e_i. Or d'après le lemme 5.5 – page 150, tous les suffixes de $s[1..i]$ sont reconnus au plus tard à l'état e_i dans l'Oracle et tous les suffixes de $s[1..j]$ sont reconnus au plus tard à l'état e_j. De surcroît, les suffixes de $s[1..i]$ de longueur supérieure à j (*i.e.*, les suffixes de $s[1..i]$ qui ne sont pas suffixes de $s[1..j]$) sont reconnus après l'état e_j (par construction), donc en e_i (sans quoi $s[1..j]$ ne serait pas le plus long suffixe de $s[1..i]$ non reconnu en e_i). Ainsi, les classes de suffixes de s s'obtiennent aisément en parcourant les liens suffixes à partir de $e_{|s|}$, et une classe de suffixes donnée (associée à un état e_i), comprend tous les suffixes de s de longueur comprise entre $j+1$ et i, avec $e_j = \mathcal{S}_s(e_i)$. Il donc possible d'ordonner les classes selon la longueur des suffixes les composant, par exemple de la classe comprenant le mot s à la classe comprenant l'unique mot ε. La numérotation des classes est directement liée au nombre de liens suffixes parcourus depuis $e_{|s|}$. Afin de simplifier les notations, $\mathcal{S}_s^n(e) = \underbrace{\mathcal{S}_s(\cdots(\mathcal{S}_s(}_{n} e)))$ désigne la composée n fois de la fonction $\mathcal{S}_s$. La classe l correspond alors à l'ensemble des suffixes de s reconnus à l'état $\mathcal{S}_s^{l-1}(e_{|s|})$ (*cf.* Figure 5.14).

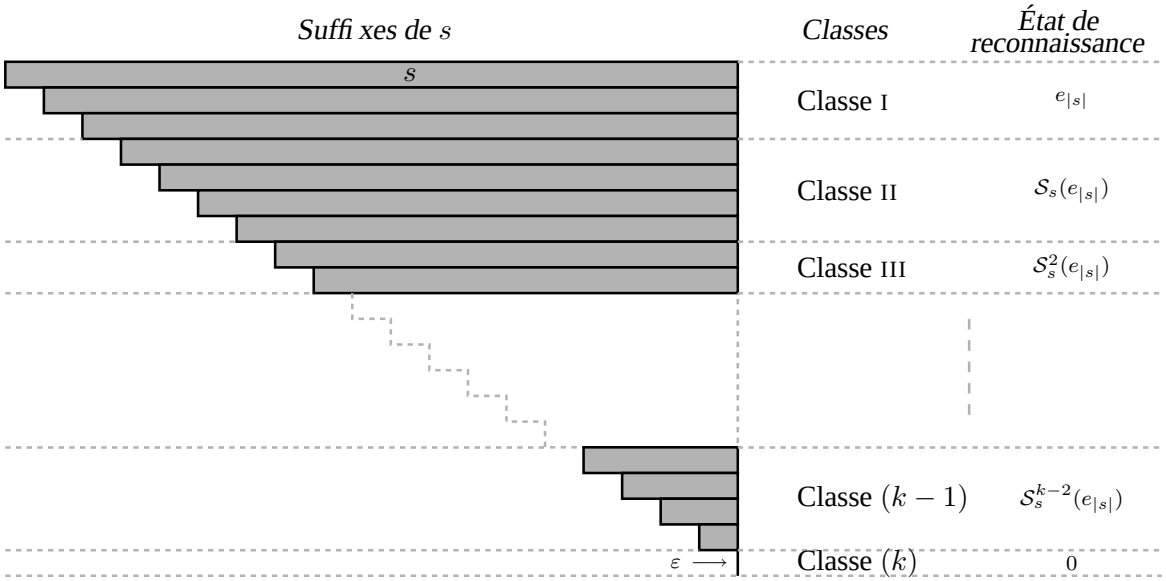


Figure 5.14 – Relation entre classes de suffixes et liens suffixes définis par les Oracles.

Dans l'algorithme « on-line » de construction de l'Oracle (*cf.* Algorithme 5.3 – page suivante), lorsque l'Oracle de $s[1..i]$ est construit, pour reconnaître $s[1..i+1]$, il suffit d'ajouter un état e_{i+1} et une transition entre $s[i]$ et $s[i+1]$ étiquetée par $s[i+1]$. Ensuite, il reste à ajouter les transitions externes arrivant en e_{i+1} permettant de reconnaître les suffixes de $s[1..i+1]$ qui ne le sont pas déjà. L'ajout de ces dernières transitions se fait précisément à l'aide de la fonction de suppléance $\mathcal{S}_s$ qui est mise à jour au fur et à mesure. Pour des raisons algorithmiques (*cf.* [3]), $\mathcal{S}_s(e_0)$ renvoie l'état virtuel e_{-1}.

Supposons que l'Oracle d'un mot s donné soit en cours de construction. L'Oracle déjà construit est celui du mot $s[1..i]$ ($1 \leq i < |s|$). L'état e_{i+1} est alors créé, et une transition de l'état e_i vers e_{i+1} étiqueté par $\alpha = s[i+1]$ est ajoutée. Supposons que $\mathcal{S}_s(e_i) = e_{k_1} \neq e_{-1}$ et qu'il n'existe pas de transition partant de e_{k_1} par α. Cet état est l'état de reconnaissance de p, le plus long suffixe de $s[1..i]$ non reconnu en e_i. Cela signifie qu'à l'état e_{k_1}, il faut ajouter une transition vers e_{i+1} afin de reconnaître les suffixes $p_1\,\alpha$ tels

```
1  Nom : Construit_Oracle_On_Line
2  Entrées : Σ % Alphabet (supposé minimal) %
3            s ∈ Σ* % Le mot à traiter %
4  Sortie : Oracle % L'Oracle de s %
5
6  Début
7    Créer l'état initial étiqueté par e_0
8    S_s(e_0) ← e_{-1}
9
10   Pour i de 1 à |s| Faire
11     Créer un état étiqueté par e_i
12     Ajouter une transition de l'état e_{i-1} vers l'état e_i étiquetée par s[i]
13     e_k ← S_s(e_{i-1})
14     Tant Que k > -1 et
15         qu'il n'existe pas de transition issue de e_k étiquetée par s[i] Faire
16       Ajouter une transition de l'état e_k vers l'état e_i étiquetée par s[i]
17       e_k ← S_s(e_k)
18     Fin Tant Que
19     Si (k = -1) Alors
20       S_s(e_i) ← e_0
21     Sinon
22       Soit e_x l'état d'arrivée de la transition issue de e_k étiquetée par s[i]
23       S_s(e_i) ← e_x
24     Fin Si
25   Fin Pour
26 Fin
```

Algorithme 5.3 – Construction « on-line » de l'Oracle d'un mot.

que les p_1 soient les suffixes de p reconnus en e_{k_1} (*i.e.*, d'une même classe de suffixe de p). Il faut ensuite faire reconnaître les suffixes $p_v\,\alpha$ ($v > 1$) de $p\,\alpha$ tels que les p_v soient les suffixes de p reconnus dans un état e_{k_v} ($k_v < k_1$, *cf.* Remarque 5.16 – page 164). C'est le rôle de la boucle `Tant Que` (lignes 15 à 18), qui permet de parcourir les différentes classes de suffixes de p. Supposons maintenant qu'il existe une transition partant de e_{k_v} ($0 \leq k_v \leq k_1$) par α vers l'état e_x ($k_v < x \leq i$) ; cela signifie alors que tous les suffixes $p_v\,\alpha$ sont reconnus par l'Oracle au plus tard à l'état e_x, et que le plus long suffixe de $s[1..i+1]$ non reconnu en e_i l'est alors en e_x (car p_v est reconnu en e_{k_v}). Cela mène à une remarque intéressante :

Remarque 5.18. Une transition externe n'est construite à partir de e_{k_v} vers e_{i+1} par α que pour permettre la reconnaissance des suffixes $p_v\,\alpha$ de $p\,\alpha$ tels que les mots p_v soient les suffixes de p reconnus en e_{k_v}. Aussi est-il possible de n'autoriser l'utilisation de ces transitions que si le mot reconnu en e_{k_v} est de longueur inférieure ou égale à $\max(|p_v|)$. S'il existait déjà une transition externe issue de e_{k_v} étiqueté par α vers un état e_x, alors il est nécessaire de mettre à jour la garde en prenant le maximum entre la garde actuelle, et $\max(|p_v|)$.

De la même manière que pour $\mathcal{S}_s$, il est possible d'associer une seconde fonction sur les états.

Définition 5.19. Étant donnés un mot $s \in \Sigma^+$ et son Oracle, la fonction succédanée $\mathcal{L}_s$ est définie comme étant la fonction qui identifie, pour chaque état e_i ($i \leq 1 \leq |s|$) de l'Oracle de s, la longueur du plus grand suffixe de $s[1..i]$ reconnu à l'état $\mathcal{S}_s(e_i)$.

Ainsi, pour construire l'Oracle à transitions gardées de $s[1..i+1]$ à partir de l'Oracle à transitions gardées de $s[1..i]$ (*cf.* Algorithme 5.4), il suffit d'ajouter un état e_{i+1}, une transition entre $s[i]$ et $s[i+1]$, et les transitions externes *gardées* arrivant en e_{i+1}. L'ajout de ces transitions se fait à l'aide de la fonction de suppléance $\mathcal{S}_s$ (*cf.* Définition 5.15 – page 164) et de la fonction succédanée $\mathcal{L}_s$ (*cf.* Définition 5.19 – page ci-contre). La mise à jour de $\mathcal{L}_s$ est faite parallèlement et sur le même principe que celle de $\mathcal{S}_s$.

```
1  Nom : Construit_Oracle_Transitions_Gardees
2  Entrées : Σ % Alphabet (supposé minimal) %
3            s ∈ Σ* % Le mot à traiter %
4  Sortie : Oracle % L'Oracle à transitions gardées de s %
5
6  Début
7    Créer l'état initial étiqueté par e_0
8    S_s(e_0) ← e_{-1}
9    L_s(e_0) ← -1
10
11   Pour i de 1 à |s| Faire
12     Créer un état étiqueté par e_i
13     Ajouter une transition de l'état e_{i-1} vers l'état e_i étiquetée par s[i]
14     l ← L_s(e_{i-1}) % Longueur du plus long suffixe répété de s[1..i-1] %
15     e_k ← S_s(e_{i-1})
16     Tant Que k > -1 et
17         qu'il n'existe pas de transition issue de e_k étiquetée par s[i] Faire
18       Ajouter une transition de l'état e_k vers l'état e_i étiquetée par s[i]
19       Poser l comme valeur de garde pour cette transition.
20       l ← L_s(e_k) % Longueur du plus long suffixe répété de s[1..k] %
21       e_k ← S_s(e_k)
22     Fin Tant Que
23     Si (k = -1) Alors
24       S_s(e_i) ← e_0
25       L_s(e_i) ← 0
26     Sinon
27       Soit e_x l'état d'arrivée de la transition issue de e_k étiquetée par s[i]
28       S_s(e_i) ← e_x
29       L_s(e_i) ← l + 1
30       Si la transition issue de e_k vers e_x par s[i+1] est externe Alors
31         l' ← valeur actuelle de la garde de cette transition.
32         Poser max(l, l') comme nouvelle valeur de garde pour cette transition.
33       Fin Si
34     Fin Si
35   Fin Pour
36 Fin
```

Algorithme 5.4 – Construction « on-line » de l'Oracle à transitions gardées d'un mot.

Il est donc possible de construire, sans modification de la complexité spatiale et temporelle, un Oracle à transitions gardées reconnaissant au moins tous les Facteurs/Suffixes d'un mot donné. Ce nouvel Oracle reconnaît au plus autant de mots que l'Oracle original défini par ALLAUZEN *& al.*. Étant donnés un mot s, son Oracle et un mot p étiquetant un chemin de l'état initial vers un état e_i, s'il existe une transition

gardée (externe) issue de e_i étiquetée par α, la transition est passante si la longueur de p est inférieure ou égale à la valeur de la garde, elle est bloquante sinon.

Exemple 5.6 (Oracle à transitions gardées).
En reprenant l'exemple de l'Oracle des Suffixes du mot $gaccattctc$ (cf. Figure 5.4), il est possible de construire son homologue à transitions gardées (cf. Figure 5.15). Celui-ci n'a plus que 8 faux positifs (gac, ac, $gaccattc$, $accattc$, $ccattc$, $cattc$, $attc$ et ttc) parmi les 32 faux positifs de l'Oracle des suffixes original. La valeur indiquée sous chaque état de la figure indique la valeur de retour de la fonction successeur qui lui est associée. Par exemple, le mot $atc(= g\,\mathbf{a}cc\mathbf{a}\mathbf{t}t\mathbf{c}\mathbf{t}\mathbf{c})$ n'est pas reconnu car la transition (e_6, c, e_8) est bloquante ($|at| \not\leq 1$).

5.5.2 Discussion sur les Oracles à transitions gardées

Cette nouvelle structure garde les principaux avantages de l'Oracle original. Cependant, elle soulève de nouvelles questions, notamment concernant la caractérisation du langage engendré par les Oracles à transitions gardées. Il est évident que ce langage est un sous-ensemble du langage reconnu par les Oracles, mais les ensembles de contractions applicables à un mot donné sont conditionnés par les valeurs des gardes. Cette question demeure ouverte. À défaut d'y répondre plus en détail, la section suivante montre le gain en qualité (eu égard aux faux positifs engendrés) de cette nouvelle structure par rapport à l'originale. Donner un ordre de grandeur du nombre de faux positifs engendrés par l'Oracle à transitions gardées serait appréciable. Toutefois, il a été possible de le déterminer pour la version originale, grâce à la connaissance du langage engendré par cette structure. Ne disposant pas de cette connaissance pour la version à transitions gardées, et n'ayant trouvé aucun mot s qui engendre un nombre exponentiel (par rapport à $|s|$) de faux positifs, il est possible (au vu des essais réalisés) d'avancer la conjecture que le nombre de faux positifs engendrés par l'Oracle à transitions gardées est polynomial (par rapport à $|s|$) dans le pire des cas. Pour exemple, les mots de la forme $aabbccddee\ldots$ qui permettent d'affirmer que le nombre de faux positifs engendrés par l'Oracle d'un mot s donnés est dans $\Omega\left(\sqrt{2^{|s|}}\right)$ dans le pire des cas n'engendrent pour la version à transition gardées qu'un nombre linéaire par rapport à $|s|$ de faux positifs dans la version Suffixes et aucun faux positif dans la version Facteurs (*cf.* Figure 5.16).

5.5.3 Tests & Résultats

Le langage reconnu par l'Oracle de Facteurs (respectivement Suffixes) à transitions gardées d'un mot s donné est un sous-ensemble du langage reconnu par l'Oracle des Facteurs (respectivement Suffixes) original de s, incluant au moins l'ensemble des facteurs (respectivement suffixes) de s. Moins le nombre de mots reconnus est important, plus l'Oracle (original ou à transitions gardées) est performant. En effet, cela traduit directement une baisse du nombre de mots non facteurs (respectivement suffixes) de s par l'automate. Il est donc intéressant d'effectuer des tests sur ces deux structures afin d'avoir une estimation de leurs performances respectives. À partir de ces estimations, il est alors possible de comparer ces deux structures afin d'estimer le gain de performance apporté par l'ajout des gardes sur les transitions externes. Afin d'évaluer ces différents automates, il est nécessaire de déterminer dans un premier temps les paramètres à faire varier, ainsi que les critères d'évaluation retenus.

Le premier paramètre à faire varier est la taille de l'alphabet Σ. Le second paramètre à faire varier est la taille des mots utilisés pour les tests, notée n. Concernant les données utilisées pour l'évaluation, les mots sont générés à partir d'une source produisant aléatoirement et uniformément les symboles d'un

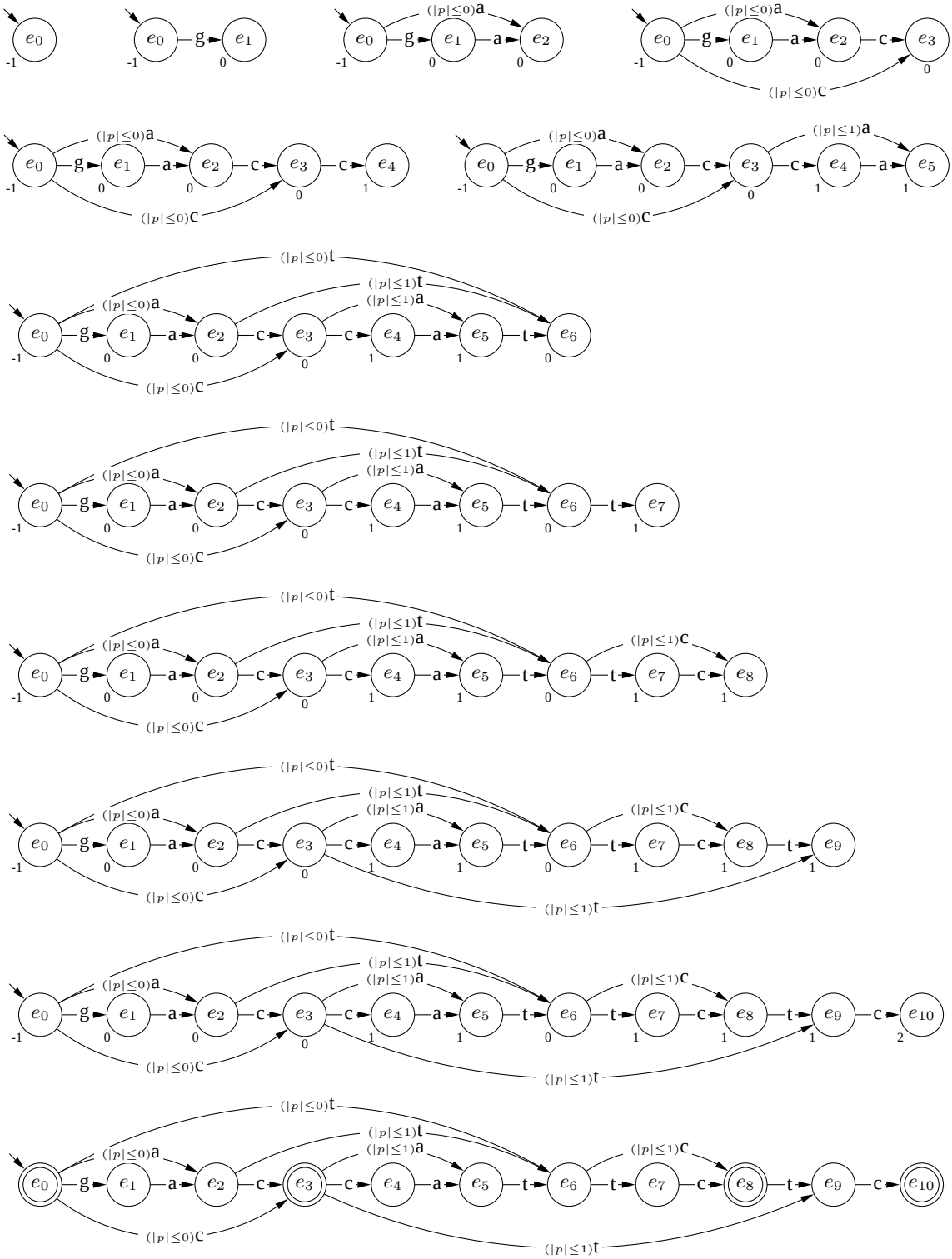


Figure 5.15 – Exemple de construction « on-line » de l'Oracle des Suffixes à transitions gardées du mot $gaccattctc$.

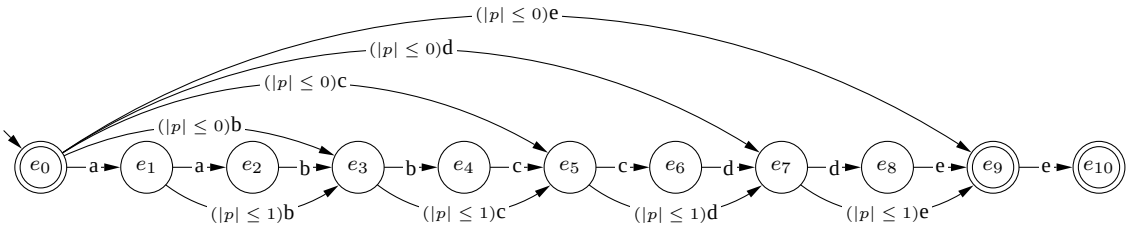


Figure 5.16 – Oracle des Suffixes à transitions gardées du mot $aabbccddee$.

alphabet de taille donnée. Pour chaque mot ont été calculés son nombre de facteurs et son nombre de suffixes, ainsi que le nombre de mots reconnus par ses différents Oracles (des Facteurs ou des Suffixes, à transitions gardées ou non). Pour les paramètres $|\Sigma|$ et n fixés, chacun des quatre Oracles a donc été testé avec les mêmes jeux d'essais, chaque jeu d'essais étant constitué de 1 000 mots.

Le protocole d'expérimentation posé, il faut définir les critères retenus pour l'évaluation. Il est possible de considérer un Oracle comme un test (au sens statistique du terme), permettant de discriminer une population (un ensemble de mots) selon que les éléments la composant présentent un critère (suffixe/facteur d'un mot donné) ou non. Dans ce cas, les résultats du test peuvent être représentés sous forme d'un tableau (*cf.* Tableau 5.1, version annotée du tableau 3.3 – page 74). Il est à noter que, pour chacun des Oracles, il n'y a pas de faux négatifs.

	Critère validé	Critère non validé
Test positif	Vrais Positifs (VP)	Faux Positifs (FP)
Test négatif	Faux Négatifs (FN)	Vrais Négatifs (VN)

⇓

	Mots à accepter	Mots à rejeter
Mots acceptés	VP	FP
Mots rejetés	FN = 0	VN

Table 5.1 – Classification des résultats des Oracles.

Parmi les indicateurs usuellement utilisés dans l'évaluation de tests statistiques, ceux les plus fréquemment employés sont la sensibilité, la spécificité, les valeurs prédictives positives (VPP) et négatives (VPN) [128]. Ils sont décrits dans la table 5.2, reproduction de la table 3.4 – page 74.

$$\text{Sensibilité} = \frac{\text{VP}}{\text{VP} + \text{FN}} \qquad \text{VPP} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

$$\text{Spécificité} = \frac{\text{VN}}{\text{VN} + \text{FP}} \qquad \text{VPN} = \frac{\text{VN}}{\text{VN} + \text{FN}}$$

Table 5.2 – Indicateurs usuels de la performance des tests statistiques.

Un autre indicateur également intéressant est le coefficient de YULE [139] (*cf.* Table 5.3 – page suivante) qui permet de mesurer l'intensité de liaison entre un test statistique et la réalité (*cf.* [126] pour

une étude comparative de 31 coefficients – dont celui-ci – applicables aux tests statistiques).

$$Q = \frac{\text{VP} \times \text{VN} - \text{FP} \times \text{FN}}{\text{VP} \times \text{VN} + \text{FP} \times \text{FN}}$$

$$\begin{cases} Q \leq 0 & \text{nulle,} \\ 0 < Q < 0{,}1 & \text{négligeable,} \\ 0{,}1 \leq Q < 0{,}3 & \text{légère,} \\ 0{,}3 \leq Q < 0{,}5 & \text{modérée,} \\ 0{,}5 \leq Q < 0{,}7 & \text{forte,} \\ 0{,}7 \leq Q \leq 1 & \text{très forte.} \end{cases}$$

Table 5.3 – Coefficient de YULE.

Plus ces indicateurs sont proches de 1, plus le test est considéré comme efficace. Une des propriétés des Oracles est que FN est nul. Par conséquent, la sensibilité, la valeur prédictive négative ainsi que le coefficient de YULE sont égaux à 1 pour tous les Oracles. De surcroît, le nombre très important de vrais négatifs fait que la spécificité est (expérimentalement) toujours très proche de 1, pour tous les Oracles. Le paramètre qui peut donc nous apporter des informations pertinentes pour la comparaison entre les Oracles et les Oracles à transitions gardées est la valeur prédictive positive.

Étude de la VPP *:*

Concernant l'Oracle original, même sur de petits mots, la VPP moyenne (*i.e.*, la proportion de facteurs ou suffixes parmi l'ensemble des mots acceptés par l'Oracle) tend vers 0, ce qui rend ces structures extrêmement peu fiables (*cf.* Figures 5.17 et 5.19). En effet, ce résultat signifie que la quasi totalité des mots acceptés par les Oracles d'un mot donné ne sont pas facteur ou suffixe (selon la version) de ce mot.

A contrario, pour l'Oracle à transitions gardées, la VPP moyenne décroît beaucoup plus lentement, surtout lorsque la taille de l'alphabet est grande (*cf.* Figures 5.18 et 5.20). En particulier, l'Oracle des Facteurs à transitions gardées a une VPP supérieure à 90% pour une taille d'alphabet supérieure à 13 ($0 \leq n \leq 1\,000$). Même si ces valeurs sont éloignées de 1, pour de petits alphabets, ou pour la version Suffixes, le gain observé par rapport à l'Oracle est d'ores et déjà très largement significatif.

Comparaison de l'Oracle original et de celui à transitions gardées :

L'Oracle est certainement une structure intéressante à plusieurs titres, ce que nous n'avons pas manqué de dire tout au long de ce chapitre. Il est toutefois à utiliser avec beaucoup de précaution lorsqu'il s'agit de gérer ses réponses fausses. Certains algorithmes, comme *Turbo*-BOM, n'utilisent que l'information garantie apportée par les Oracles : lorsque l'Oracle répond négativement, la réponse attendue est effectivement négative. D'autres applications ne permettent pas d'éviter aussi sûrement les imperfections des Oracles. L'étude que nous avons présentée montre qu'il ne faut surtout pas sous-estimer ces imperfections, et qu'il faut les gérer au cas par cas en fonction du contexte (type de séquences – avec ou sans répétitions, probabilité d'apparition de chaque caractère, qualité de la réponse souhaitée, ...)

Les résultats obtenus en ajoutant les gardes sur les transitions externes sont très satisfaisants. Ils sont d'autant plus probants que la taille de l'alphabet utilisé est élevée. Certainement, l'ajout des gardes augmente légèrement l'espace mémoire nécessaire à l'utilisation des Oracles à transitions gardées par

rapport aux Oracles, mais cet inconvénient est faible par rapport au gain en fiabilité. Un autre avantage non négligeable de cette nouvelle structure est que, de par sa similarité avec l'Oracle, il est envisageable de l'utiliser à la place de l'Oracle dans les algorithmes et applications tels ceux cités précédemment (*Turbo*-BOM, FORREPEATS, COMPROR, *OpenMusic* et POPITAM). Une étude comparative sur la qualité des résultats de ces méthodes en substituant l'Oracle à transitions gardées à l'Oracle original serait par ailleurs très intéressante. Une étude des nouvelles propriétés inhérentes à la fonction succédanée (s'il en est) devrait améliorer encore la compréhension de cette structure, et éventuellement aboutir à l'élaboration d'une structure encore plus efficace.

Conclusion :

Les Oracles ont été étudiés dans cette thèse dans le but d'une éventuelle utilisation pour le stockage et la gestion des séquences dans STARS et StatiSTARS. Le premier pas était de s'assurer que les Oracles était fiables. Le deuxième était de trouver le moyen de les intégrer efficacement dans les algorithmes des chapitres précédents. Les conclusions qui se sont imposées après étude théorique et expérimentations ont changé forcément cette démarche, en concentrant plus de forces et de temps que prévu pour la première tâche. Le résultat de cette étude plus approfondie est l'aboutissement à un Oracle légèrement modifié mais pour lequel les expérimentations donnent des résultats très prometteurs. Cette structure doit, elle aussi, être rigoureusement évaluée avant que son intégration dans STARS ou StatiSTARS ne soit envisagée.

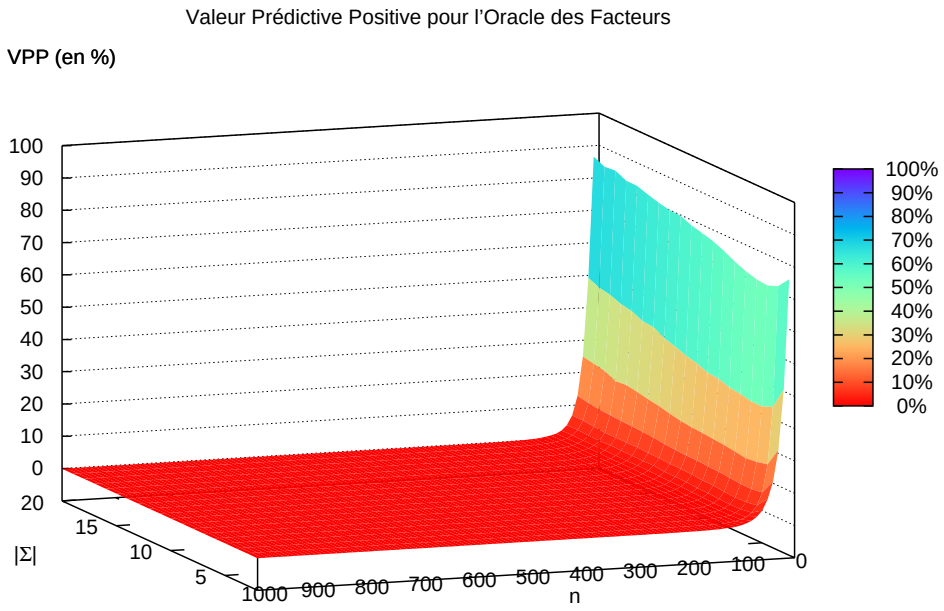


Figure 5.17 – Valeur Prédictive Positive pour l'Oracle des Facteurs.

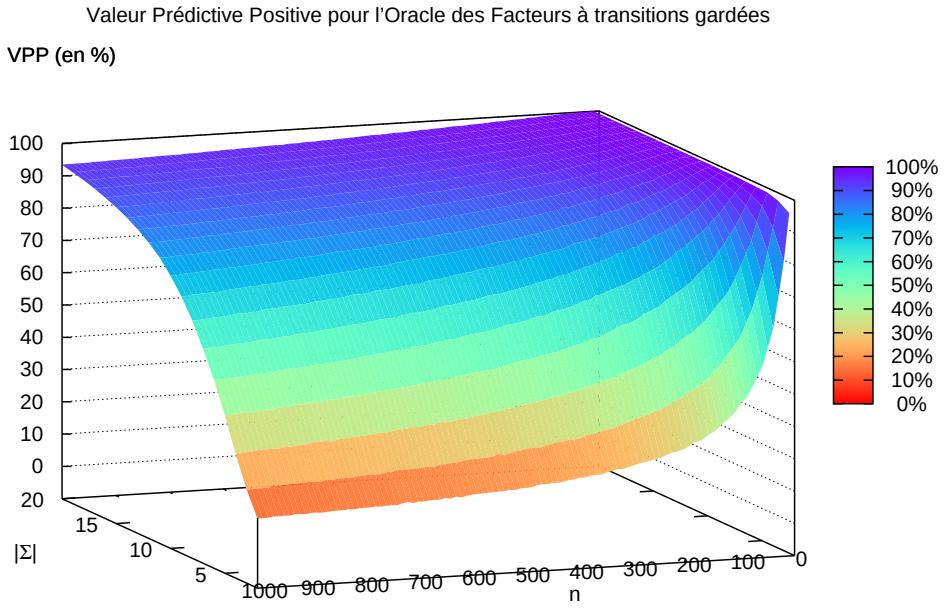


Figure 5.18 – Valeur Prédictive Positive pour l'Oracle des Facteurs à transitions gardées.

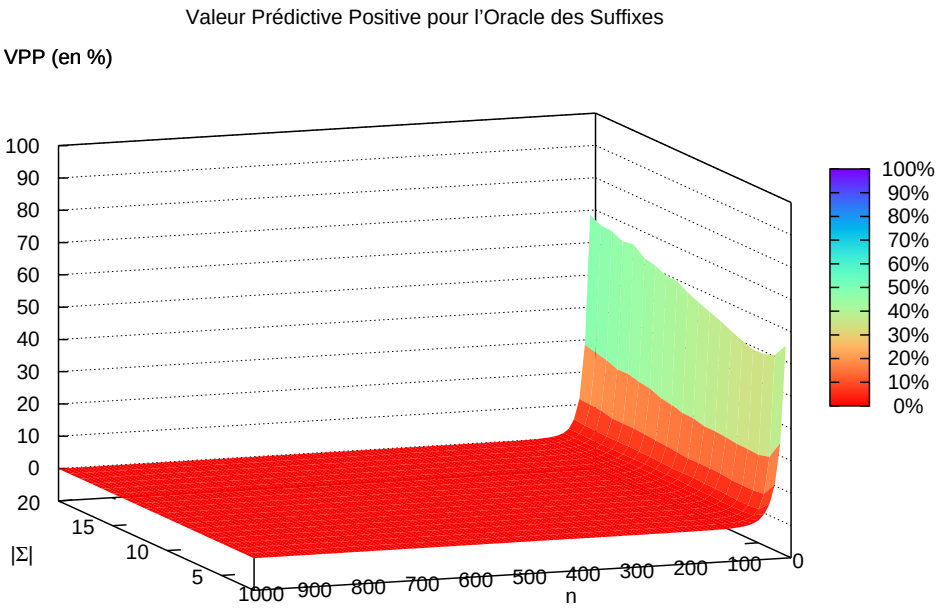


Figure 5.19 – Valeur Prédictive Positive pour l'Oracle des Suffixes.

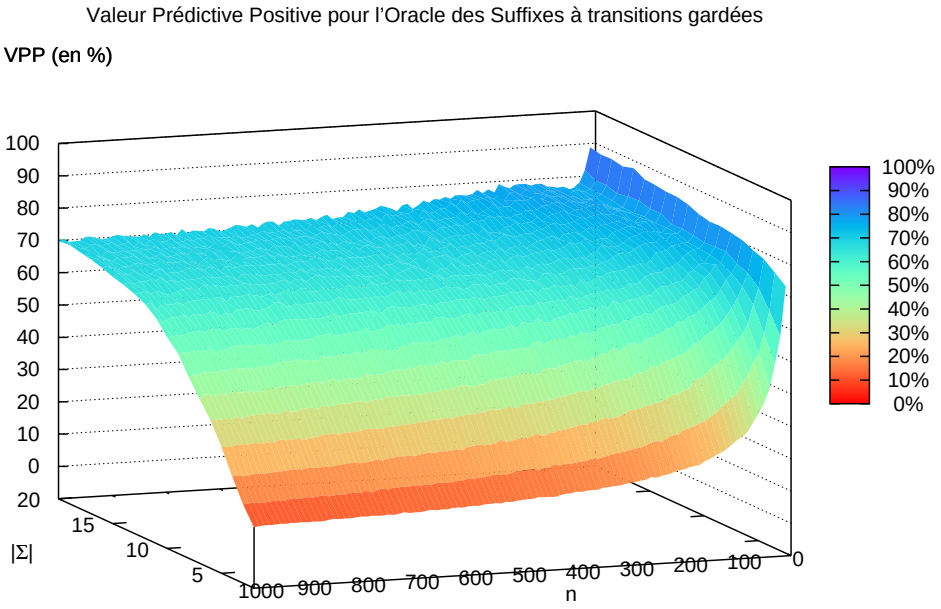


Figure 5.20 – Valeur Prédictive Positive pour l'Oracle des Suffixes à transitions gardées.

Conclusions et perspectives

Need a little patience,
Just a little patience,
Some more patience…

I've been walking these streets at night,
Just trying to get it right.
It's hard to see with so many around.
You know I don't like being stuck in a crowd
And the streets don't change but maybe the name.
I ain't got time for the game,
'Cause I need you
All this time…

— Guns N' Roses, Patience (1988).

Plusieurs résultats ont été présentés tout au long de ce manuscrit. Ils sont l'aboutissement d'un travail long de plusieurs années. Il est toutefois difficile de conclure, car bien qu'il s'agisse de la fin d'un cycle universitaire, les travaux réalisés s'inscrivent dans une continuité. Plusieurs questions demeurent ouvertes, tandis que d'autres sont apparues. Avant d'évoquer ces questions, le contexte initial de cette thèse est rappelé, afin de situer le travail réalisé par rapport au sujet. Cette synthèse permettra d'aborder les questions ouvertes, mais également d'aborder différentes pistes en vue de leur répondre et plus généralement d'envisager de nouvelles perspectives de recherche. Il apparaît évident que ces perspectives s'inscrivent dans le cadre d'un projet personnel, lequel sera l'objet du troisième et dernier sujet développé dans ce chapitre.

Synthèse

L'objectif initial de cette thèse a été l'étude et la réalisation de méthodes informatiques, d'outils permettant l'analyse *in silico* de séquences biologiques, principalement d'ADN. Les applications se sont rapidement orientées vers l'identification de sites de liaisons ADN/protéine. Du point de vue « informatique », l'intérêt de ce problème réside dans l'explosion combinatoire du nombre de solutions potentielles. La motivation « biologique » (et même au-delà de la biologie) provient essentiellement de la volonté de mieux comprendre le monde vivant, afin de mieux pouvoir interagir sur le fonctionnement métabolique des cellules.

Cette double motivation a ceci de particulier que les travaux menés pendant la durée de cette thèse se devaient d'être validés tant pour l'aspect informatique que pour l'aspect biologique.

Le premier algorithme développé, STARS, est une méthode probabiliste à base de fouille. Sa complexité est polynomiale en espace et en temps. Il permet de répondre au problème de l'extraction de motifs communs à un ensemble de séquences (*i.e.*, motifs présents dans toutes les séquences). L'évaluation statistique de ses performances donne des résultats très encourageants. Appliqué à la localisation de sites de liaisons dans le génome d'*Escherichia coli*, il a concurrencé les plus célèbres algorithmes d'extraction. Une des particularités de cet outil est sa simplicité d'utilisation, et sa modularité. En effet, il autorise l'emploi d'un grand nombre de fonctions de score afin d'évaluer la similarité locale entre motifs.

Parmi les fonctions de score utilisées dans STARS, les fonctions *Block-Based* ont donné de bons résultats, ce qui a mené à les étudier de manière plus approfondie. Cette étude a permis de mettre en évidence des formules closes pour calculer (ou approcher) la moyenne et la variance des scores obtenus à partir de séquences générées aléatoirement selon la fonction utilisée. Des travaux connexes ont par ailleurs mis en évidence le caractère Gaussien de la distribution de ces scores. Une seconde méthode a alors été développée, afin d'intégrer et de tirer profit de l'information statistique associée à chacune de ces fonctions : StatiSTARS. En effet, la connaissance de la moyenne, de la variance, et du caractère Gaussien de la distribution des scores permet de déterminer pour un score donné, la probabilité que ce score soit un fait du hasard. Cette information, appelée P-valeur, permet d'accompagner les divers choix effectués dans StatiSTARS d'une interprétation statistique, ce qui augmente encore l'intérêt de cette méthode. Cet algorithme est déterministe à base de fouille et polynomial. En outre, il répond au problème plus général de l'extraction de motifs communs sous contrainte de quorum dans un ensemble de séquences.

Une autre piste a également été explorée, en essayant de développer une méthode à base d'index et non plus à base de fouille. Parmi les structures existantes, l'Oracle des Facteurs ou des Suffixes a semblé un choix judicieux, du fait des résultats obtenus par les diverses méthodes l'utilisant. Toutefois,

plusieurs questions étaient ouvertes au sujet de cette structure, dont la caractérisation du langage reconnu par cet index. En effet, l'Oracle d'un mot donné reconnaît au moins les facteurs ou suffixes (selon la version) de ce mot. Il est question de reconnaissance faible de motifs. Cela signifie qu'un mot rejeté par l'Oracle n'est pas facteur ou suffixe du mot indexé, mais qu'un mot accepté n'est pas nécessairement facteur ou suffixe du mot indexé. Aussi avant d'intégrer cette structure dans un quelconque algorithme nous sommes nous intéressés à la caractérisation du langage reconnu par les Oracles. Une fois cette caractérisation établie, l'emploi de cette structure comme index pour un algorithme d'extraction de motif a semblé inopportun. Toutefois, un nouvel index basé sur l'Oracle a été élaboré, toujours dans le contexte de la reconnaissance faible de motifs, offrant une fiabilité de réponse au moins égale à celle des Oracles, et nettement supérieure en pratique. Il s'agit de l'Oracle à transitions gardées.

Il demeure évident qu'il reste des questions non résolues au sujet des Oracles, et que d'autres algorithmes d'extraction de motifs sont encore à élaborer. Néanmoins, l'objectif initial de la thèse a été traité, et deux nouvelles méthodes ont été développées. Il est possible d'utiliser STARS (et prochainement StatiSTARS) *via* internet, et les deux outils sont disponibles au téléchargement ; les sources sont distribuées sous licence GPL.

Perspectives

Parmi les travaux restant à effectuer sur les Oracles, une étude statistique des mots acceptés a été entamée. Cette connaissance permettrait notamment de fournir un indicateur de vraisemblance des résultats de l'Oracle. Un autre aspect a également été abordé, l'élaboration d'un automate acyclique homogène permettant une reconnaissance faible de motifs, qui soit minimal en nombre de sommets, mais également minimal en nombre de transitions. En effet, l'Oracle est la structure la plus économique en espace à ce jour, et permet par conséquence l'exploration de plus grands génomes. Le gain d'espace représente donc l'un des aspects qu'il serait souhaitable d'explorer concernant ce type de structure. En outre, la notion de minimalité en nombre de transitions n'a pas été beaucoup explorée en théorie des automates, et cela pourrait constituer une voie d'investigation intéressante. Enfin, les tests réalisés sur les Oracles et les Oracles à transition gardées tendent à montrer que l'augmentation de la taille de l'alphabet n'implique pas une détérioration des performances (au contraire, la fiabilité augmente même pour l'Oracle à transitions gardées). C'est pourquoi leur utilisation dans d'autres domaines de l'informatique (par exemple la fouille de données, la lutte antispam, ...) pourrait s'avérer utile.

Concernant la famille de fonctions de score *Block-Based*, l'étude a été menée dans le cas particulier de la mesure de la similarité locale entre deux motifs de même longueur (*i.e.*, des mots alignement sur un alphabet à deux symboles). Une étude plus complète a également été entamée dans le cas plus général où le nombre de fonctions composantes est supérieur à 2 (donc l'alphabet également). Ces fonctions pourraient également être utilisées dans d'autres domaines de l'informatique (réseau, bases de données, ...).

Enfin, l'extraction de motifs communs sous contrainte de quorum dans un ensemble de séquences demeure très spécifique, et d'autres problèmes d'exploration de séquences existent qui pourraient être abordés ; entre autres, l'extraction de mots sur-exprimés dans un ensemble de séquences et sous-exprimés dans un second ensemble de séquences (*i.e.*, les mots sont supposés présents dans toutes ou parties des séquences des deux ensembles, le critère distinctif étant le nombre d'occurrences dans chacun des deux ensembles).

Bilan

Outre les aspects scientifiques nécessairement développés durant ce cycle universitaire, la thèse est également un parcours initiatique au monde de la recherche. C'est durant cette période que paraissent à la lumière les enjeux des laboratoires, des équipes, des institutions. À cette occasion, le doctorat est une préparation au métier de chercheur et parfois d'enseignant. Outre les connaissances acquises et développées pendant ces quelques années, la formation doctorale permet d'appréhender les qualités à développer sur toute une carrière d'enseignant et de chercheur : la méthode, la rigueur et l'autonomie.

Annexes

ANNEXE A

Le code IUPAC

La beauté d'un mot ne réside pas dans l'harmonie phonétique de ses syllabes, mais dans les associations sémantiques que sa sonorité éveille.

— Milan KUNDERA, L'art du roman (1986).

A.1 Les Acides Nucléiques

Le standard de l'IUPAC spécifie que la représentation des macromolécules s'effectue de préférence en utilisant un codage à trois lettres afin de désigner chaque molécule. L'idée d'un codage à une lettre a été mentionné à partir de 1958 [50]. Son utilisation est conseillée uniquement pour une représentation de longues séquences. En effet le codage à trois lettres demeure plus intuitif (*cf.* Tableau A.1).

Molécule(s)	Système de codage		symbolique
	trois lettres	une lettre	
Adénine	Ade	A	Adénine
Cytosine	Cyt	C	Cytosine
Guanine	Gua	G	Guanine
Thymine	Thy	T	Thymine
Uracile	Ura	U	Uracile
Purine (Gua ou Ade)	Pur	R	puRine
Pyrimidine (Thy ou Cyt)	Pyr	Y	pYrimidine
Amines (Ade ou Cyt)	/	M	aMines
Kétines (Gua ou Thy/Ura)	/	K	Kétines
Gua ou Cyt	/	S	interactions fortes (Strong)
Ade, Thy ou Ura	/	W	interactions faibles (Weak)
Ade, Cyt ou Thy ou Ura	/	H	tous sauf G, qui est suivi de H
Cyt, Gua, Thy ou Ura	/	B	tous sauf A, qui est suivi de B
Cyt, Gua ou Ade	/	V	tous sauf T/U, qui est suivi de V
Ade, Gua, Thy ou Ura	/	D	tous sauf C, qui est suivi de D
Ade, Cyt, Gua, Thy ou Ura	Bas	N	Toutes les bases (aNy)

Table A.1 – Nomenclature des Acides Nucléiques [69].

A.2 De l'ADN aux protéines

Dans la table A.2 est fourni le code quasi-universel du code génétique. Les protéines sont synthétisées lors de la phase de traduction de l'ARN en acides aminés. Chaque acide aminé provient de la traduction d'un codon ARN spécifique (séquence de trois acides ribonucléiques). Ces codons sont représentés dans la table en lisant successivement la base de l'entête de ligne gauche, celle de l'entête de colonne, puis celle de l'entête de ligne droit.

<table>
<tr><th></th><th>U</th><th>C</th><th>A</th><th>G</th><th></th></tr>
<tr><td rowspan="2">U</td><td rowspan="2">Phénylalanine</td><td rowspan="3">Sérine</td><td>Tyrosine</td><td>Cystéine</td><td>U</td></tr>
<tr><td>Stop[1]</td><td>Stop[2]</td><td>C</td></tr>
<tr><td rowspan="3">Leucine</td><td>Stop[3]</td><td>Tryptophane</td><td>A</td></tr>
<tr><td>C</td><td rowspan="2">Proline</td><td>Histidine</td><td rowspan="3">Arginine</td><td>G</td></tr>
<tr><td rowspan="2">C</td><td rowspan="2">Glutamine</td><td>U</td></tr>
<tr><td rowspan="2">A</td><td rowspan="2">Isoleucine</td><td rowspan="3">Thréonine</td><td>C</td></tr>
<tr><td>Asparagine</td><td>Sérine</td><td>A</td></tr>
<tr><td>Methionine</td><td rowspan="2">Lysine</td><td rowspan="2">Arginine</td><td>G</td></tr>
<tr><td rowspan="2">G</td><td rowspan="2">Valine</td><td rowspan="2">Alanine</td><td>U</td></tr>
<tr><td>Ac. Aspartique</td><td rowspan="2">Glycine</td><td>C</td></tr>
<tr><td>Ac. Glutamique</td><td>A</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>G</td></tr>
</table>

Table A.2 – De l'ADN aux acides aminés.

Ce code génétique est le même quelles que soient les espèces, à quelques exceptions près [105] :

- le codon CUG, traduit habituellement par la leucine, correspond à la sérine chez de nombreuses espèces de champignons *Candida* ;
- les variations du codage utilisé par les mitochondries sont encore plus nombreuses. Par exemple, dans le génome mitochondrial de *Saccharomyces cerevisiæ* (levure de boulanger, *cf.* Section 1.2.2 – page 17), la thréonine est codée par 4 des 6 codons correspondant classiquement à la leucine ;
- de nombreuses espèces d'algues vertes du genre *Acetabularia* utilisent les codons stop UAG et UAA pour coder la glycine ;
- dans les trois règnes du vivant (*Archaea*, *Bacteria* et *Eucarya*), un 21ème acide aminé existe, la sélénocystéine, codé par le codon UGA ;
- dans les régnes *Archaea* et *Bacteria*, un 22ème acide aminé est parfois rencontré : la pyrrolysine, codé par UAG.

[1] Ce codon est aussi appelé *Ocre*.
[2] Ce codon est aussi appelé *Opale* ou *Azur*.
[3] Ce codon est aussi appelé *Ambre*.

A.3 Les Acides Aminés

A.3.1 Codage

Le codage à une lettre des acides aminés (hormis la Sélénocystéine et la Pyrrolysine découverts après 1983) suit la même convention que pour les acides nucléiques lorsqu'il n'y a pas d'ambiguïté. Les autres codes ont été assignés en essayant de faciliter leurs mémorisations par des moyens mnémotechnique (notamment phonétiques).

Molécule(s)	Système de codage		codon ADN
	trois lettres	une lettre	
Alanine	Ala	A	GCN
Cystéine	Cys	C	UGY
Acide Aspartique	Asp	D	GAY
Acide Glutamique	Glu	E	GAR
Phénylalanine	Phe	F	UUY
Glycine	Gly	G	GGN
Histidine	His	H	CAY
Isoleucine	Ile	I	AUH
Lysine	Lys	K	AAR
Leucine	Leu	L	YUN (CUN et UUR)
Methionine	Met	M	AUG
Asparagine	Asn	N	AAY
Proline	Pro	P	CCN
Glutamine	Gln	Q	CAR
Arginine	Arg	R	MGN (CGN et AGR)
Sérine	Ser	S	WSN (UCN et AGY)
Thréonine	Thr	T	ACN
Sélénocystéine	Sec	U	UGA
Valine	Val	V	GUN
Tryptophane	Trp	W	UGG
Tyrosine	Tyr	Y	UAY
Acide Aspartique ou Asparagine	Asx	B	RAY
Acide Glutamique ou Glutamine	Glx	Z	SAR
Inconnu ou autres	Xaa	X	NNN
Pyrrolysine[4]	/	UAG	

Table A.3 – Nomenclature des Acides Animés [70].

[4] Cet acide aminé a été découvert en 2002 [124]. L'IUPAC ne s'est pas prononcé sur son sort... Gageons que ce sera Pyr / J.

A.3.2 Propriétés

Dans certains cas, il peut être intéressant de ne pas comparer les acides aminés au niveau moléculaire, mais plutôt en fonction de leurs propriétés physico-chimique (*cf.* Table A.4).

Acide Aminé	Propriétés Physiques				Propriétés Chimiques			
	charge		aspect		hydrophobe	polaire	aromatique	aliphatique
	+	-	petit	très petit				
Ala			x	x	x			
Cys			x	x	x	x		
Asp		x	x					
Glu		x				x		
Phe					x		x	
Gly			x	x	/			
His	x				x	x	x	
Lys	x				x	x		
Ile					x			x
Leu					x			x
Met					x			
Asn			x			x		
Pro			x					
Gln						x		
Arg	x					x		
Ser			x	x		x		
Thr			x	x	x	x		
Val			x		x			x
Trp					x	x	x	
Tyr					x	x	x	

Table A.4 – Propriétés physico-chimiques des acides aminés.

ANNEXE B

Représentation des ensembles de séquences biologiques et classification des algorithmes d'extraction de motif

Et si le temps domine encor sur nos désirs,
Faisons que sur le temps la constance domine.

— Jean DE SPONDE [1557–1595], Mon Dieu, que je voudrais que ma main fût oisive.

B.1 Format FASTA

Le format FASTA a été introduit par le programme homonyme, et repris ensuite par BLAST. Une séquence est représentée par une ligne de description débutant nécessairement par le caractère '>', suivie d'une ou plusieurs lignes représentant la structure primaire de la séquence, en utilisant la nomenclature à une lettre définie par le standard IUPAC (*cf.* Annexe A). La casse des caractères n'a pas d'incidence, tout caractère minuscule est converti en majuscule. Le tiret semi-cadrin permet de modéliser un trou de longueur indéterminée. L'usage requiert que les lignes représentant la séquence I^{aire} soient formatées sur 80 colonnes.

Exemple B.1 (Séquences au format FASTA).

```
>gi|145340|gb|M24185.1|ECOARGFPRM E.coli argininosuccinate lyase (argF) gene
ggatccaatcattctcatttctgactcgacctagttgtagaattcgatccaatgtctttctgcttctgcagagaatcgga
ggcagatacgattattttcacacacggacgggtttgcctccacctttgtaagaaagaattgtgaaatggggttgcaAATG
AATAATTACACATAtaaAGTGAATTTTAATTCAATaagtggcgttcgccatgcgaggataaaatgtccgatttatacaaa
aaacactttctgaaactgctcgactttacccctgcacagttcacttctctgctgaccctt

>gi|394809|emb|X68963.1|ECARGG E.coli DNA for argG operator
aaagatgATTAAATGAAAACTCATTtatTTTGCATAAAAATTCAGTgagagcg
```

Ce format est utilisé dans de nombreux logiciels, il est simple et intuitif, et qui plus est, il existe un utilitaire permettant de manipuler et d'effectuer des conversions dans une vingtaine de formats différents (dont le format FASTA) : *readseq*.

B.2 Notation PROSITE

La notation `Prosite` a été introduite vers la fin des années 1980 par BAIROCH dans la base de donnée homonyme, et officialisée en 1994 [20]. Elle permet de représenter facilement un motif consensuel. La syntaxe d'un motif `Prosite` est assez intuitive. En effet, les acides aminés ou nucléiques sont représentés par le code à une lettre défini par la norme IUPAC (indépendamment de la casse), à l'exception de la lettre X, qui correspond toujours à un acide indéterminé. S'il existe une ambiguïté à une position précise entre plusieurs acides, deux cas sont distingués :

- l'ambiguïté est inclusive (*i.e.*, chaque occurrence du motif peut mettre en correspondance un des acides, indépendamment des autres occurrences), auquel cas l'ambiguïté est définie par l'ensemble des acides autorisés délimités par des crochets (*e.g.* `[IL]`);
- l'ambiguïté est exclusive (*i.e.*, toutes les occurrences du motif doivent mettre en correspondance le même acide parmi les acides autorisés). Dans ce cas l'ambiguïté est définie par l'ensemble des acides autorisés délimités par des accolades (*e.g.* `{IL}`).

Il est possible de dénoter un nombre borné de répétitions en définissant le couple représentant les nombres minimum et maximum de répétitions entre parenthèses (*e.g.* `[ILP](2,3)`). Lorsque le nombre de répétitions est fixe, il est possible de n'écrire alors que ce nombre (*e.g.* `X(3,3)` $\equiv$ `X(3)`). Enfin un motif est délimité par les symboles '`<`' et '`>`'. Chaque position est séparée par le tiret semi-cadrin. Si le motif est grand, il est possible de le scinder en plusieurs lignes, en utilisant le symbole '`+`' qui signifie que le motif continue sur la ligne suivante.

Exemple B.2 (Motif au format PROSITE).
La première ligne de l'exemple ci-dessous représente l'ensemble des séquences protéiques commençant par une molécule d'alanine, suivie de n'importe quel acide aminé, puis d'une glycine, d'une histidine, de trois acides aminés quelconques et d'un acide aminé parmi une glutamine, un sérine ou une thréonine, terminée soit toujours par un acide aspartique, soit toujours par une arginine. Les trois lignes du milieu correspondent à un motif. La dernière ligne représente une séquence débutant par une cystéine, suivie soit toujours par une seconde cystéine, une proline, une molécule de tryptophane, une histidine ou une phénylalanine. La séquence se continue de deux à quatre acides aminés quelconques, puis de nouveau d'une cystéine, d'une histidine et se termine toujours soit par une cystéine, une phénylalanine, une tyrosine ou une tryptophane.

```
AXGHXXX[QST]{DR}
<P-x(2)-R-G-[STAIV](2)-x-N-[APK]-x-[DE]
[STANVF]-x(2)-F-x(4)-[DNS]-[DENQTF]-Y-[HFY]-x(2)-[LIVMFY]-x(3)-+
[LIVM]-x(4)-[LIVM]-x(6,8)-Y-x(12,13)-[LIVM]-x(2)-N-[SACF]-x(2)-[FY]>
C-{CPWHF}-X(2,4)-C-H-{CFYW}
```

B.3 Hiérarchie de motifs

BRAZMA *& al.* [18] ont établi une classification structurale des types de motifs. Cette classification permet d'établir une relation d'ordre partiel en utilisant l'inclusion ensembliste. Ainsi, sur la base des classes de motifs qu'un algorithme est capable d'extraire, il devient plus aisé de le classer. Il convient de distinguer un motif « simple » (*i.e.,* un motif n'autorisant pas l'usage d'un alphabet dégénéré) d'un motif « dégénéré » (*i.e.,* qui peut s'écrire en utilisant un alphabet étendu, également appelé dégénéré, *cf.* AnnexeA).

classe	motifs	trous	illustration
A	simples	non	T-C-T-T-G-A
B	simples	longueur fixe	D-R-C-C-x(2)-H-D-x-C
C	dégénérés	non	G-G-G-T-F-[ILV]-[ST]-[ILV]
D	dégénérés	longueur fixe	V-x-P-x(2)-[RQ]-x(4)-G-x(2)-L-[LM]
E	simples	longueur bornée	G-C-x(1,3)-C-P-x(8,10)-C-C
F	dégénérés	longueur bornée	C-x(2,4)-C-x(3)-[ILVFYC]-x(8)-H-x(3,5)-H
G	simples	longueur non bornée	D-T-A-G-Q-E-*-L-V-G-N-K
H	dégénérés	longueur non bornée	D-T-A-G-[NQ]-*-L-V-G-N-[KEH]
I	dégénérés	longueur bornée ou non	D-T-A-x(2,5)-G-[NQ]-*-L-V-G-N-[KEH]

Table B.1 – Classification des algorithmes en fonction des motifs qu'ils peuvent extraire.

ANNEXE C

Matrices de similarité

The practice of normal science depends on the ability acquired from exemplars to group objects and situations into similarity sets which are primitive in the sense that the grouping is done without an answer to the question "similar with respect to what?" A central aspect of a revolution is that some of the similarity relations change.

— Thomas Samuel KUHN [1922–1996], The Structure of Scientific Revolutions (1962).

C.1 Acides nucléiques

Exemple de matrice de similarité entre acides nucléiques disponible à l'adresse :
http://www.umanitoba.ca/afs/plant_science/psgendb/dat/fasta/dna.mat
(Université de Manitoba – Canada)

	A	C	G	T	U	R	Y	M	W	S	K	D	H	V	B	N	X
A	5	-4	-4	-4	-4	2	-1	2	2	-1	-1	1	1	1	-2	-1	-1
C	-4	5	-4	-4	-4	-1	2	2	-1	2	-1	-2	1	1	1	-1	-1
G	-4	-4	5	-4	-4	2	-1	-1	-1	2	2	1	-2	1	1	-1	-1
T	-4	-4	-4	5	5	-1	2	-1	2	-1	2	1	1	-2	1	-1	-1
U	-4	-4	-4	5	5	-1	2	-1	2	-1	2	1	1	-2	1	-1	-1
R	2	-1	2	-1	-1	2	-2	-1	1	1	1	1	-1	1	-1	-1	-1
Y	-1	2	-1	2	2	-2	2	-1	1	1	1	-1	1	-1	1	-1	-1
M	2	2	-1	-1	-1	-1	-1	2	1	1	-1	-1	1	1	-1	-1	-1
W	2	-1	-1	2	2	1	1	1	2	-1	1	1	1	-1	-1	-1	-1
S	-1	2	2	-1	-1	1	1	1	-1	2	1	-1	-1	1	1	-1	-1
K	-1	-1	2	2	2	1	1	-1	1	1	2	1	-1	-1	1	-1	-1
D	1	-2	1	1	1	1	-1	-1	1	-1	1	1	-1	-1	-1	-1	-1
H	1	1	-2	1	1	-1	1	1	1	-1	-1	-1	1	-1	-1	-1	-1
V	1	1	1	-2	-2	1	-1	1	-1	1	-1	-1	-1	1	-1	-1	-1
B	-2	1	1	1	1	-1	1	-1	-1	1	1	-1	-1	-1	1	-1	-1
N	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
X	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Table C.1 – Matrice de similarité entre acides nucléiques.

C.2 Acides aminés

Pour chacun des trois exemples de matrices qui suivent, deux indicateurs sont donnés en sus, la distance moyenne entre deux acides aminés sur la matrice et l'entropie des acides aminés.

La moyenne est calculée selon l'équation

$$\mathbf{E}[\mathcal{M}] = \sum_{i=1}^{20} \sum_{j=1}^{i} p_i \, p_j \, \mathcal{M}_{i,j}, \tag{C.1}$$

où $\mathcal{M}$ désigne la matrice, $\mathcal{M}i, j$ le score de la cellule à la ligne i et à la colonne j dans $\mathcal{M}$, et p_i (respectivement p_j) désigne la fréquence d'apparition du symbole en tête de ligne i (respectivement j) lors de la construction de la matrice.

L'entropie est calculée en appliquant une formule inspirée de la formule de SHANNON [121] sur l'ensemble des couples d'acides aminés (*cf.* Équation 2.8 – page 28), comme le montre l'équation C.2.

$$H[\mathcal{M}] = -\sum_{i=1}^{20} \sum_{j=1}^{i} q_{i,j} \, \lambda \, \mathcal{M}_{i,j}, \tag{C.2}$$

où $q_{i,j}$ et λ sont tels que $q_{i,j} = p_i \, p_j \, e^{\lambda \, \mathcal{M}_{i,j}}$.

La moyenne calculée est toujours négative, tandis que l'entropie est toujours positive (*cf.* [78] pour de plus amples détails sur la construction des matrices et le calcul de ces indicateurs ; le lecteur y trouvera un descriptif d'une étonnante limpidité et vraiment très complet).

C.2.1 Matrice de type PAM

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	0	0	-8
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	-1	-8
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	1	0	-8
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	-1	-8
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-3	-8
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	-1	-8
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	3	-1	-8
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	0	-1	-8
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	-1	-8
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	-1	-8
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	-1	-8
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	-1	-8
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	-1	-8
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	-5	-2	-8
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	-1	-8
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0	-8
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0	-8
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	-4	-8
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	-2	-8
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	-1	-8
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	2	-1	-8
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	-1	-8
X	0	-1	0	-1	-3	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1	0	0	-4	-2	-1	-1	-1	-1	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

Table C.2 – Matrice PAM250, dite de DAYHOFF.

La valeur moyenne de la distance entre deux acides aminés sur cette matrice est de $-0{,}844$; l'entropie est de 0,354.

C.2.2 Matrice de type BLOSUM

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-2	-2	0	-1	-1	0	-5
R	-2	7	0	-1	-3	1	0	-2	0	-3	-2	3	-1	-2	-2	-1	-1	-2	-1	-2	-1	0	-1	-5
N	-1	0	6	2	-2	0	0	0	1	-2	-3	0	-2	-2	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-1	2	7	-3	0	2	-1	0	-4	-3	0	-3	-4	-1	0	-1	-4	-2	-3	5	1	-1	-5
C	-1	-3	-2	-3	12	-3	-3	-3	-3	-3	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-2	-3	-2	-5
Q	-1	1	0	0	-3	6	2	-2	1	-2	-2	1	0	-4	-1	0	-1	-2	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-2	0	-3	-2	1	-2	-3	0	0	-1	-3	-2	-3	1	4	-1	-5
G	0	-2	0	-1	-3	-2	-2	7	-2	-4	-3	-2	-2	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-5
H	-2	0	1	0	-3	1	0	-2	10	-3	-2	-1	0	-2	-2	-1	-2	-3	2	-3	0	0	-1	-5
I	-1	-3	-2	-4	-3	-2	-3	-4	-3	5	2	-3	2	0	-2	-2	-1	-2	0	3	-3	-3	-1	-5
L	-1	-2	-3	-3	-2	-2	-2	-3	-2	2	5	-3	2	1	-3	-3	-1	-2	0	1	-3	-2	-1	-5
K	-1	3	0	0	-3	1	1	-2	-1	-3	-3	5	-1	-3	-1	-1	-1	-2	-1	-2	0	1	-1	-5
M	-1	-1	-2	-3	-2	0	-2	-2	0	2	2	-1	6	0	-2	-2	-1	-2	0	1	-2	-1	-1	-5
F	-2	-2	-2	-4	-2	-4	-3	-3	-2	0	1	-3	0	8	-3	-2	-1	1	3	0	-3	-3	-1	-5
P	-1	-2	-2	-1	-4	-1	0	-2	-2	-2	-3	-1	-2	-3	9	-1	-1	-3	-3	-3	-2	-1	-1	-5
S	1	-1	1	0	-1	0	0	0	-1	-2	-3	-1	-2	-2	-1	4	2	-4	-2	-1	0	0	0	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-1	-1	2	5	-3	-1	0	0	-1	0	-5
W	-2	-2	-4	-4	-5	-2	-3	-2	-3	-2	-2	-2	-2	1	-3	-4	-3	15	3	-3	-4	-2	-2	-5
Y	-2	-1	-2	-2	-3	-1	-2	-3	2	0	0	-1	0	3	-3	-2	-1	3	8	-1	-2	-2	-1	-5
V	0	-2	-3	-3	-1	-3	-3	-3	-3	3	1	-2	1	0	-3	-1	0	-3	-1	5	-3	-3	-1	-5
B	-1	-1	4	5	-2	0	1	-1	0	-3	-3	0	-2	-3	-2	0	0	-4	-2	-3	4	2	-1	-5
Z	-1	0	0	1	-3	4	4	-2	0	-3	-2	1	-1	-3	-1	0	-1	-2	-2	-3	2	4	-1	-5
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-2	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Table C.3 – Matrice BLOSUM45.

La valeur moyenne de la distance entre deux acides aminés sur cette matrice est de $-0{,}278\,9$; l'entropie est de $0{,}379\,5$.

C.2.3 Matrice de type GONNET

Matrice dérivée de la matrice PAM250, telle que définie par GONNET *& al.* [53] en 1992.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	X	*
C	12	0	0	-3	0	-2	-2	-3	-3	-2	-1	-2	-3	-1	-1	-2	0	-1	0	-1	-3	-8
S	0	2	2	0	1	0	1	0	0	0	0	0	0	-1	-2	-2	-1	-3	-2	-3	0	-8
T	0	2	2	0	1	-1	0	0	0	0	0	0	0	-1	-1	-1	0	-2	-2	-4	0	-8
P	-3	0	0	8	0	-2	-1	-1	0	0	-1	-1	-1	-2	-3	-2	-2	-4	-3	-5	-1	-8
A	0	1	1	0	2	0	0	0	0	0	-1	-1	0	-1	-1	-1	0	-2	-2	-4	0	-8
G	-2	0	-1	-2	0	7	0	0	-1	-1	-1	-1	-1	-4	-4	-4	-3	-5	-4	-4	-1	-8
N	-2	1	0	-1	0	0	4	2	1	1	1	0	1	-2	-3	-3	-2	-3	-1	-4	0	-8
D	-3	0	0	-1	0	0	2	5	3	1	0	0	0	-3	-4	-4	-3	-4	-3	-5	-1	-8
E	-3	0	0	0	0	-1	1	3	4	2	0	0	1	-2	-3	-3	-2	-4	-3	-4	-1	-8
Q	-2	0	0	0	0	-1	1	1	2	3	1	2	2	-1	-2	-2	-2	-3	-2	-3	-1	-8
H	-1	0	0	-1	-1	-1	1	0	0	1	6	1	1	-1	-2	-2	-2	0	2	-1	-1	-8
R	-2	0	0	-1	-1	-1	0	0	0	2	1	5	3	-2	-2	-2	-2	-3	-2	-2	-1	-8
K	-3	0	0	-1	0	-1	1	0	1	2	1	3	3	-1	-2	-2	-2	-3	-2	-4	-1	-8
M	-1	-1	-1	-2	-1	-4	-2	-3	-2	-1	-1	-2	-1	4	2	3	2	2	0	-1	-1	-8
I	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-2	-2	-2	2	4	3	3	1	-1	-2	-1	-8
L	-2	-2	-1	-2	-1	-4	-3	-4	-3	-2	-2	-2	-2	3	3	4	2	2	0	-1	-1	-8
V	0	-1	0	-2	0	-3	-2	-3	-2	-2	-2	-2	-2	2	3	2	3	0	-1	-3	-1	-8
F	-1	-3	-2	-4	-2	-5	-3	-4	-4	-3	0	-3	-3	2	1	2	0	7	5	4	-2	-8
Y	0	-2	-2	-3	-2	-4	-1	-3	-3	-2	2	-2	-2	0	-1	0	-1	5	8	4	-2	-8
W	-1	-3	-4	-5	-4	-4	-4	-5	-4	-3	-1	-2	-4	-1	-2	-1	-3	4	4	14	-4	-8
X	-3	0	0	-1	0	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	-2	-4	-1	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

Table C.4 – Matrice de GONNET *& al.*.

La valeur moyenne de la distance entre deux acides aminés sur cette matrice est de $-0{,}615\,2$; l'entropie est de $1{,}684\,5$.

ANNEXE D

Théorie des Graphes et des Automates

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" (I found it!) but rather, "hmm. . . . . that's funny. . . "

— Isaac ASIMOV.

Les graphes et automates permettent de manipuler plus facilement des objets et leurs relations avec une représentation graphique simple et naturelle. L'ensemble des techniques et outils mathématiques mis au point en Théorie des Graphes, comme en Théorie des Automates permettent d'établir des propriétés sur ces objets ou ces relations, d'en déduire des méthodes de résolution, des algorithmes, ... La différence essentielles entre graphes et automates réside dans le fait que les graphes sont généralement utilisés pour la modélisation des problèmes, tandis que les automates sont plutôt utilisés pour leur résolution.

D.1 Éléments de théorie des graphes

Après avoir présentés les définitions basiques relatives aux graphes, sont détaillés les structures d'arbres, et notamment les arbres binaires.

Les définitions et concepts présentés dans cette section sont essentiellement issus de [17, 110].

D.1.1 Définitions de base

Définition D.1. Un graphe G est un couple (V, E) où :

- V est un ensemble fini d'objets. Les éléments de V sont appelés les sommets du graphe ;
- E est sous-ensemble de $V \times V$. Les éléments de E sont appelés les arêtes du graphe.

Une arête e du graphe est une paire $e = (v_1, v_2)$ de sommets. Les sommets v_1 et v_2 sont les extrémités de l'arête. Si la paire (v_1, v_2) est orientée (*i.e.*, $(v_1, v_2) \neq (v_2, v_1)$), alors il s'agit d'un arc de v_1 vers v_2. Un graphe dont les arêtes sont orientées est dit orienté. Dans le cas contraire, il est dit non orienté.

Définition D.2. Étant donné un graphe $G = (V, E)$,

- deux sommets v_1 et v_2 sont adjacents si $(v_1, v_2) \in E$. Les sommets v_1 et v_2 sont alors dits voisins. Si G est orienté, v_1 est un prédécesseur de v_2, et v_2 est un successeur de v_1 ;
- une arête est incidente à un sommet v si v est l'une de ses extrémités. Si G est orienté, l'arête est dite sortante par rapport à v si v est le premier sommet de la paire, elle est dite entrante par rapport à v si v est le second sommet de la paire ;
- le degré d'un sommet v de G est le nombre d'arêtes incidentes à v, il est noté $\#(v)$. Dans le cas des graphes orientés, le degré d'un sommet est la somme de son degré entrant et de son degré sortant, notés respectivement $\#_{in}(v)$ et $\#_{out}(v)$.

Définition D.3. Étant donnés un graphe non orienté (respectivement orienté) $G = (V, E)$ et une suite de sommets $(v_1, \ldots, v_k)$ telle qu'il existe une arête (respectivement un arc) entre chaque paire de sommets successifs, on dit que $(v_1, \ldots, v_k)$ est un chemin de G. La longueur du chemin est le nombre d'arêtes qui le composent : $k - 1$. Si $(v_k, v_1) \in E$, alors $(v_1, \ldots, v_k)$ est un cycle de G. Un graphe tel qu'il existe au moins un chemin entre toutes les paires de sommets est dit connexe.

D.1.2 Arbres

Les arbres sont une catégorie de graphes aux propriétés particulières. Parmi ceux-ci, la sous-catégorie des arbres binaires est largement utilisée en informatique, principalement pour sa facilité d'implémentation et d'exploitation (la nature binaire de l'arbre étant en adéquation avec le mode de représentation des données en informatique).

D.1.2.1 Généralités

Définition D.4. Étant donné un graphe non orienté $G = (V, E)$, G est un arbre si et seulement s'il est connexe et sans cycle.

Définition D.5. La racine d'un arbre est un sommet particulier de l'arbre qui lui confère une orientation (*i.e.*,, tous les chemins entre la racine et un nœud v de l'arbre sont orientés de la racine vers v). La racine est alors le seul sommet sans prédécesseur.

Définition D.6. Une feuille d'un arbre est un sommet de l'arbre de degré 1.

Définition D.7. Un nœud de l'arbre est un sommet de l'arbre. Si le nœud n'est ni la racine, ni une feuille, il est dit interne.

Définition D.8. La profondeur d'un nœud donné dans l'arbre est la longueur du chemin unique de la racine vers ce nœud. Le niveau d'un nœud est sa profondeur plus 1.

Définition D.9. La hauteur d'un arbre est la longueur du plus long chemin de la racine à une feuille.

Définition D.10. Étant donné un arbre $T = (V, E)$, et un nœud $v \in V$:
- le père de v est son unique prédécesseur (s'il existe). Il est noté $Pere(v)$;
- les fils de v sont ses successeurs (s'ils existent). Ils sont notés $Fils(v)$.

D.1.3 Arbres binaires

Définition D.11. Un arbre binaire est un arbre dont chaque nœud possède au plus deux fils, dénotés fils gauche et fils droit.

Définition D.12. Un arbre ordonné en tas est tel que pour chaque nœud de l'arbre, les clés (étiquettes) de ses fils (s'ils existent) sont plus petites ou égales à la clé du nœud.

Définition D.13. Un arbre binaire est dit équilibré – appelé également AVL du nom de leur deux inventeurs ADELSON-VELSKII et LANDIS en 1962 – lorsque la valeur absolue de la différence entre les hauteurs des fils gauche et droit de tout nœud n'excède pas 1.

Définition D.14. Un arbre binaire est dit complet si toutes ses feuilles sont à la même profondeur et si tous les nœuds non feuilles ont exactement deux fils.

Définition D.15. Un arbre binaire est dit quasi-complet si toutes ses feuilles sont à la profondeur n ou $n-1$ et si tous les nœuds non feuilles ont exactement deux fils, sauf éventuellement le plus à droite au niveau $n-1$ qui peut n'en avoir qu'un.

Définition D.16. Un tas est un arbre binaire quasi-complet et ordonné en tas.

Remarque D.17. Il est possible de représenter un tas à n feuilles par un vecteur V de taille $2\,n - [\![n \mod 2 = 0]\!]$ (*cf.* Figure D.1), tel que la $i^{ème}$ feuille du tas est donnée par $V[n+i]$. Une telle représentation permet de surcroît d'accéder au père, au fils gauche et au fils droit du nœud $V[i]$ en temps constant. En effet, ces nœuds (s'ils existent) sont respectivement donnés par $V[i/2]$, $V[i \times 2]$ et $V[2 \times i + 1]$.

Définition D.18. Un arbre est dit de recherche si chaque nœud a une clé supérieure ou égale à chaque clé des nœuds de son sous-arbre gauche et une clé inférieure ou égale à chaque clé de son sous-arbre droit.

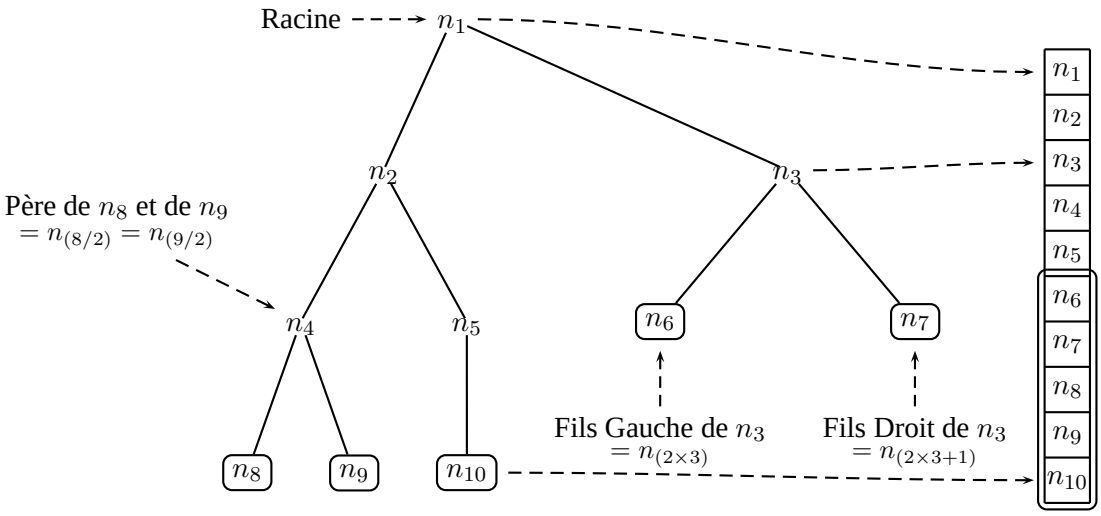


Figure D.1 – Représentation d'un tas à n feuilles par un vecteur de taille $2\,n - [\![n \mod 2 = 0]\!]$.

D.2 Éléments de théorie des Automates

Un automate est une structure de données proche de celle de graphe, c'est pourquoi les automates sont présentés dans cette même annexe. Les définitions données dans cette section sont très largement inspirées de celles données par WATSON [130, 131].

Définition D.19. Un automate à états finis (AF) est un quintuplet $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$, où Q est un ensemble fini d'états, $\mathcal{I} \subseteq Q$ est l'ensemble des états initiaux, $\mathcal{F} \subseteq Q$ est l'ensemble des états finaux, Σ est un alphabet et $\delta \in \mathcal{P}(Q \times \Sigma \times Q)$ est un relation de transition.

Définition D.20. Un automate à états finis $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$ est déterministe si et seulement si :

- il existe un unique état initial ;
- il n'existe pas de transitions étiquetée par ε dans l'ensemble δ ;
- pour chaque état q et pour chaque symbole α, il existe au plus une transition issue de q étiquetée par α.

Définition D.21. Étant donnés un automate à états finis $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$, ainsi qu'une séquence de transitions (ou chemin) $(q_0, \alpha_1, q_1), \cdots, (q_{n-1}, \alpha_n, q_n) \in \delta^n$, le mot $\alpha_1 \cdots \alpha_n$ est l'étiquette du chemin issu de l'état q_0 et arrivant à l'état q_n.

Propriété D.1. *Étant donnés un automate à états finis déterministe, et un état q de cet automate, chaque chemin issu de q a une étiquette unique.*

Définition D.22. Un mot m est accepté ou reconnu par un automate à états finis $\mathcal{A}$ s'il existe un chemin issu d'un état initial et arrivant dans un état final, dont l'étiquette est m. Le langage reconnu par un automate est l'ensemble des mots que l'automate peut reconnaître ; il est noté $\mathcal{L}(\mathcal{A})$.

Définition D.23. Étant donnés un automate à états finis déterministe, ainsi qu'un chemin étiqueté par m issu de l'état initial vers un état q, la fonction *Etat* est définie par $Etat(m) = q$.

Définition D.24. Un automate à états finis $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$ est minimal si et seulement s'il n'existe pas d'automate à états finis $(Q', \mathcal{I}', \mathcal{F}', \Sigma, \delta')$ ayant moins d'états ($|Q'| < |Q|$) et reconnaissant le même langage.

Remarque D.25. Lorsqu'un automate à états finis est dit minimal déterministe, il est déterministe et minimal par rapport à l'ensemble des automates déterministes.

Définition D.26. Un automate à états finis $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$ est homogène si et seulement si pour tout état $q \in Q$, toutes les transitions arrivant à l'état q sont étiquetées par le même symbole $\alpha \in \Sigma$.

Définition D.27. Un automate à états finis $(Q, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$ est acyclique si et seulement s'il n'existe pas de chemin menant d'un état initial vers un état $q \in Q$ tel que ce chemin passe deux fois par le même état.

Définition D.28. Étant donné un automate à états finis, une garde est un prédicat associé aux transitions de l'automate. Une transition est dite passante si la garde est vérifiée et bloquante sinon. L'automate résultant est alors appelé automate à transitions gardées.

ANNEXE E

Lois de probabilités

Calculer la probabilité d'un événement n'a aucun sens une fois que l'on sait qu'il s'est produit. L'apparition de la « vie », celle des dinosaures, celles des Hommes, a résulté d'un grand nombre de bifurcations dans le cours des processus se déroulant sur notre planète ; chacune de ces bifurcations s'est produite alors que de nombreuses autres étaient possibles ; chacune avait une probabilité faible, mais il fallait bien qu'une de ces possibilités se produise.

— Albert JACQUARD, La science à l'usage des non-scientifiques (2003).

Une loi de probabilité, appelée également distribution de probabilité, est une fonction associée à une ou plusieurs variables aléatoires (*cf.* Définition 4.1 – page 84). Cette fonction, notée $x \mapsto \text{Prob}[X = x]$ définit la probabilité que la (ou les) variable(s) aléatoire(s) soi(en)t égale(s) à un résultat – ou comprise(s) dans un intervalle – donné. La loi de probabilité d'une variable aléatoire décrit la répartition des valeurs que cette dernière peut prendre.

Les lois de probabilité usuelles sont souvent classées par familles dépendant d'un ou plusieurs paramètres.

E.1 Loi Normale

La distribution normale, également appelée loi de LAPLACE-GAUSS ou plus simplement gaussienne, est une distribution continue dépendant de deux paramètres : la moyenne et la variance de la variable aléatoire, notées μ et σ. Cette distribution de probabilité (*cf.* Figure E.1) est notée $\mathcal{N}(\mu, \sigma^2)$ et est définie par

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

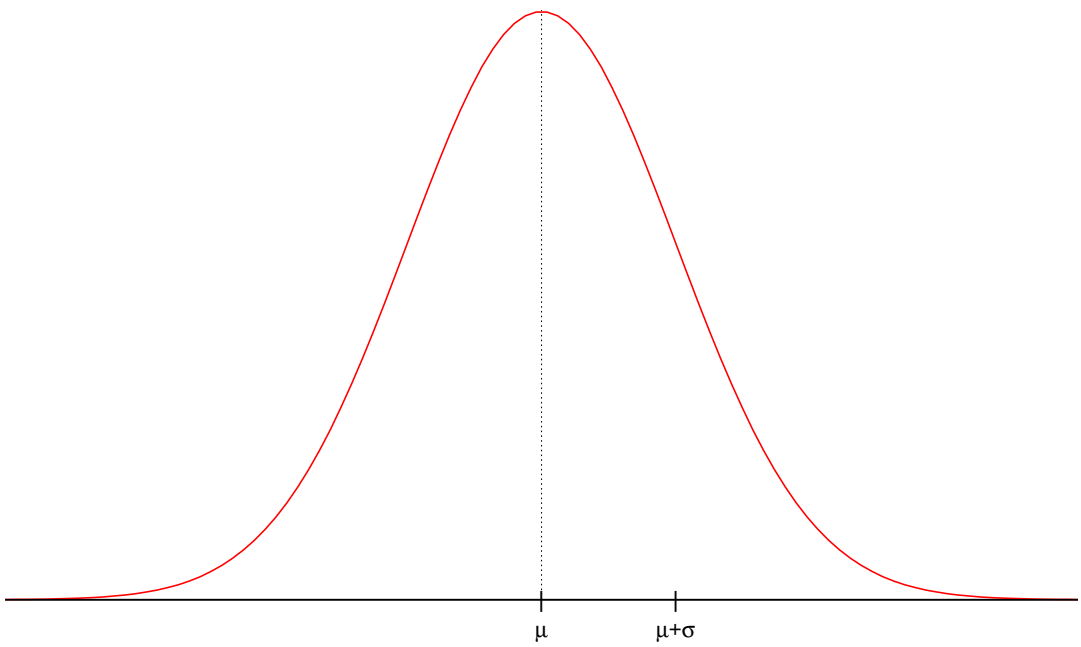


Figure E.1 – Loi de distribution Normale $\mathcal{N}(\mu, \sigma^2)$.

Propriété E.1. *Étant donnée une loi $\mathcal{N}(\mu, \sigma^2)$ associée à une variable aléatoire X, pour tout $x \in \mathbb{R}$, Prob$[X = \mu + x]$ = Prob$[X = \mu - x]$.*

La loi $\mathcal{N}(0,1)$ est également appelée la loi normale centrée réduite (*cf.* Figure E.2). Sa distribution de probabilité est alors telle que ;

$$\text{Prob}[X \leq x] = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}}\, e^{-\frac{t^2}{2}}\, dt \qquad \text{et} \qquad \text{Prob}[X \leq -x] = 1 - \text{Prob}[X \leq x].$$

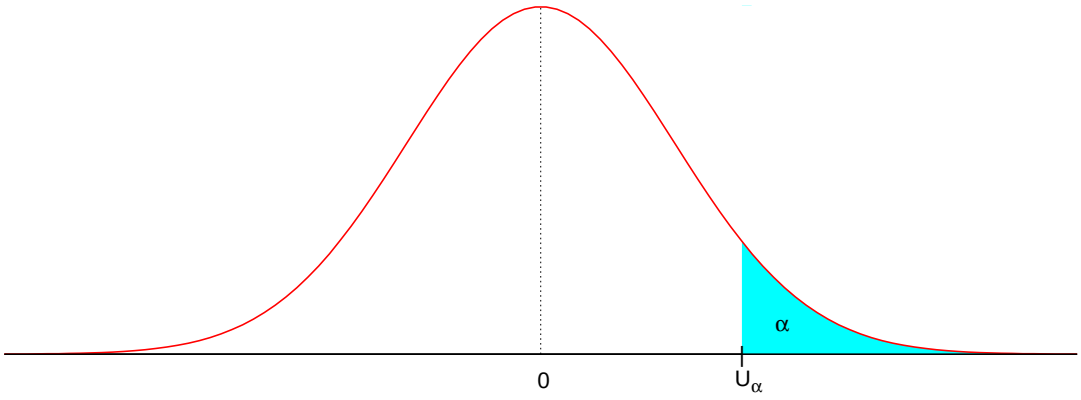


Figure E.2 – Loi de distribution Normale centrée réduite $\mathcal{N}(0,1)$.

Propriété E.2. *Étant donnée une loi $\mathcal{N}(\mu,\sigma^2)$ associée à une variable aléatoire X, il est possible de définir une nouvelle variable aléatoire $Y = \dfrac{X-\mu}{\sigma}$. Cette variable aléatoire suit la loi normale centrée réduite. Ainsi, il suffit de connaître la table de $\mathcal{N}(0,1)$ pour manipuler n'importe quelle variable aléatoire normale.*

E.1.1 Tables statistiques

Étant données une variable aléatoire de loi $\mathcal{N}(0,1)$ et une erreur de première espèce $\alpha \in [0;1]$ (*cf.* Figure E.2), il est possible de déterminer la valeur U_α telle que :

$$\text{Prob}[X \leq U_\alpha] = 1 - \alpha.$$

Pour cela, il suffit de regarder dans la case du tableau E.1 telle que l'abscisse additionnée à son ordonnée soit le plus proche possible de α.

α	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0,00	∞	2,327	2,054	1,881	1,751	1,645	1,555	1,476	1,405	1,341
0,10	1,282	1,227	1,175	1,126	1,080	1,036	0,994	0,954	0,915	0,878
0,20	0,841	0,806	0,772	0,739	0,706	0,674	0,643	0,612	0,582	0,553
0,30	0,524	0,495	0,467	0,439	0,412	0,385	0,358	0,331	0,305	0,279
0,40	0,253	0,227	0,202	0,176	0,151	0,125	0,100	0,075	0,050	0,025

Table E.1 – Détermination du seuil centré réduit U_α.

Exemple E.1 (Détermination du seuil U_α).
La valeur $\alpha = 0{,}12$ s'obtient en additionnant l'étiquette 0,10 de la deuxième ligne avec l'étiquette 0,02 de la $3^{ème}$ colonne. Ainsi la valeur seuil $U_{0,12} = 1{,}175$, est telle que Prob$[X \leq 1{,}175] = 1 - 0{,}12 = 0{,}88$.

De la même manière, à partir d'une valeur x donnée pour une variable aléatoire X de loi $\mathcal{N}(0,1)$, il est possible d'approcher Prob$[X \leq x]$ à l'aide du tableau E.2 – page suivante, en considérant la cellule de la table dont la somme de l'entête de ligne et de l'entête de colonne approche au plus x. En outre, en se basant sur la remarque E.1 – page 211, avec une table des valeurs de Prob$[X \leq x]$ pour $x \geq 0$, il est possible de déterminer Prob$[X \leq -x]$. En effet, Prob$[X \leq -x] =$ Prob$[X \geq x] \simeq 1 -$ Prob$[X \leq x]$.

Exemple E.2 (Détermination de la probabilité Prob$[X \leq x]$).
En se basant sur la table E.2 – page suivante – et sur l'exemple précédent, il est facile de vérifier que pour $x = 1{,}175$:

$$\begin{array}{ccccc} \textit{Prob}[X \leq 1{,}17] & \leq & \textit{Prob}[X \leq 1{,}175] & \leq & \textit{Prob}[X \leq 1{,}18], \\ 0{,}879 & \leq & \textit{Prob}[X \leq 1{,}175] & \leq & 0{,}881. \end{array}$$

x	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0,0	0,500	0,504	0,508	0,512	0,516	0,520	0,524	0,528	0,532	0,536
0,1	0,540	0,544	0,548	0,552	0,556	0,560	0,564	0,567	0,571	0,575
0,2	0,579	0,583	0,587	0,591	0,595	0,599	0,603	0,606	0,610	0,614
0,3	0,618	0,622	0,626	0,629	0,633	0,637	0,641	0,644	0,648	0,652
0,4	0,655	0,659	0,663	0,666	0,670	0,674	0,677	0,681	0,684	0,688
0,5	0,691	0,695	0,698	0,702	0,705	0,709	0,712	0,716	0,719	0,722
0,6	0,726	0,729	0,732	0,736	0,739	0,742	0,745	0,749	0,752	0,755
0,7	0,758	0,761	0,764	0,767	0,770	0,773	0,776	0,779	0,782	0,785
0,8	0,788	0,791	0,794	0,797	0,800	0,802	0,805	0,808	0,811	0,813
0,9	0,816	0,819	0,821	0,824	0,826	0,829	0,831	0,834	0,836	0,839
1,0	0,841	0,844	0,846	0,848	0,851	0,853	0,855	0,858	0,860	0,862
1,1	0,864	0,867	0,869	0,871	0,873	0,875	0,877	0,879	0,881	0,883
1,2	0,885	0,887	0,889	0,891	0,893	0,894	0,896	0,898	0,900	0,901
1,3	0,903	0,905	0,907	0,908	0,910	0,911	0,913	0,915	0,916	0,918
1,4	0,919	0,921	0,922	0,924	0,925	0,926	0,928	0,929	0,931	0,932
1,5	0,933	0,934	0,936	0,937	0,938	0,939	0,941	0,942	0,943	0,944
1,6	0,945	0,946	0,947	0,948	0,949	0,951	0,952	0,953	0,954	0,954
1,7	0,955	0,956	0,957	0,958	0,959	0,960	0,961	0,962	0,962	0,963
1,8	0,964	0,965	0,966	0,966	0,967	0,968	0,969	0,969	0,970	0,971
1,9	0,971	0,972	0,973	0,973	0,974	0,974	0,975	0,976	0,976	0,977
2,0	0,977	0,978	0,978	0,979	0,979	0,980	0,980	0,981	0,981	0,982
2,1	0,982	0,983	0,983	0,983	0,984	0,984	0,985	0,985	0,985	0,986
2,2	0,986	0,986	0,987	0,987	0,987	0,988	0,988	0,988	0,989	0,989
2,3	0,989	0,990	0,990	0,990	0,990	0,991	0,991	0,991	0,991	0,992
2,4	0,992	0,992	0,992	0,992	0,993	0,993	0,993	0,993	0,993	0,994
2,5	0,994	0,994	0,994	0,994	0,994	0,995	0,995	0,995	0,995	0,995
2,6	0,995	0,995	0,996	0,996	0,996	0,996	0,996	0,996	0,996	0,996
2,7	0,997	0,997	0,997	0,997	0,997	0,997	0,997	0,997	0,997	0,997
2,8	0,997	0,998	0,998	0,998	0,998	0,998	0,998	0,998	0,998	0,998
2,9	0,998	0,998	0,998	0,998	0,998	0,998	0,998	0,999	0,999	0,999
3,0	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999
3,1	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999
3,2	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999
3,3	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,5	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,7	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
3,9	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000

Table E.2 – Distribution de la loi normale centrée réduite $\mathcal{N}(0,1)$.

E.2 Loi Gamma

Une variable aléatoire X suit la loi Gamma (*cf.* Figure E.3) de paramètres $n > 0$ et $\lambda > 0$, notée $X \hookrightarrow \Gamma(n, \lambda)$ si elle est absolument continue, et admet pour densité :

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ \frac{\lambda^n}{\Gamma(n)}\, x^{n-1}\, e^{-\lambda\, x} & \text{sinon} \end{cases} \qquad \text{avec } \Gamma(x) = \int_0^{+\infty} e^{-x}\, x^{n-1}\, dx.$$

La fonction $\Gamma(x)$ est la fonction *Gamma* d'EULER.

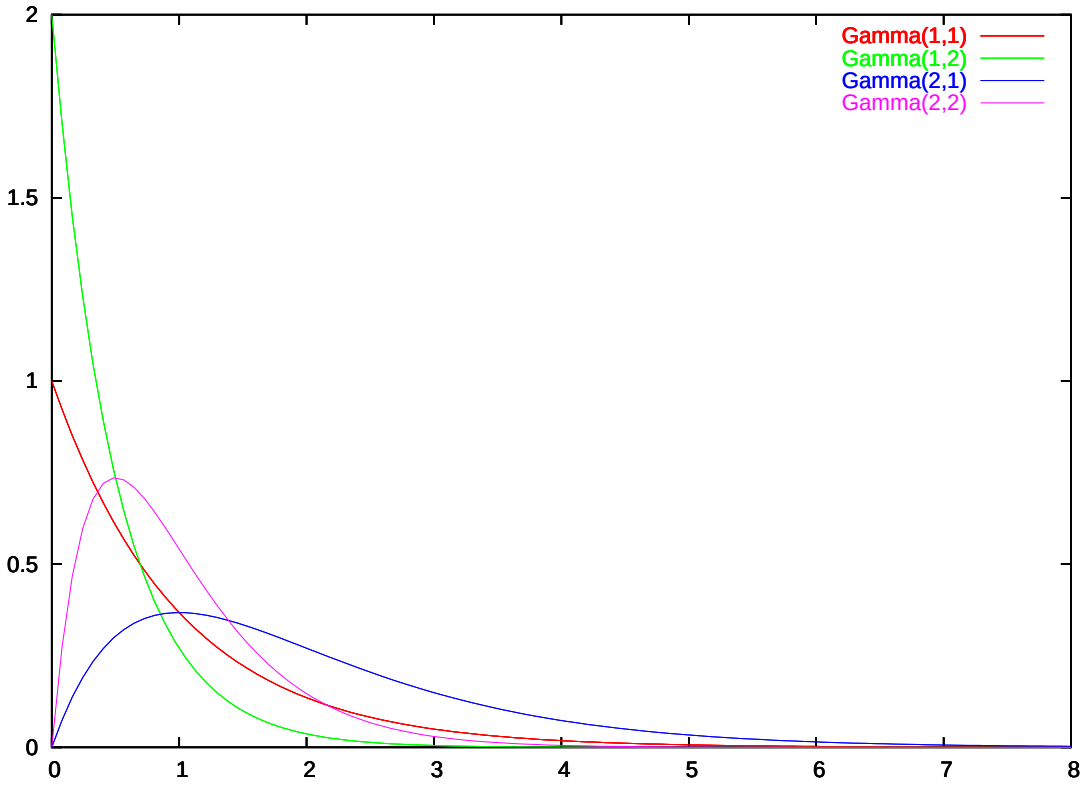


Figure E.3 – Loi de distribution *Gamma* $G(n, \lambda)$.

Lorsque $n > 1$, cette loi peut (par exemple) traduire la durée de vie d'un appareil ou d'un organisme quand son vieillissement intervient.

Lorsque le paramètre $n = 1$, cette loi est également appelée loi exponentielle, notée $X \hookrightarrow \mathcal{E}(\lambda)$. Cette loi modélise par exemple la durée de vie d'une ampoule électrique. Une des propriétés importantes de cette loi est sa propriété de non vieillissement (*i.e.*,, $\text{Prob}[X > s + t | X > t] = \text{Prob}[X > s]$).

Lorsque le paramètre $\lambda = \frac{1}{2}$, cette loi est connue sous le nom de loi du χ^2 à $2\,n$ degrés de liberté, notée $X \hookrightarrow \chi^2_{2\,n}$ qui permet par exemple de tester l'indépendance deux à deux de $2\,n$ variables aléatoires distribuées normalement centrées et réduites.

E.3 Loi de GUMBEL

Une variable aléatoire X suit une loi doublement exponentielle de paramètres μ et λ ($\mu > 0$, $\lambda > 0$) si sa loi de distribution (*cf.* Figure E.4) est définie par

$$f(x) = \frac{1}{\lambda}\, e^{-\frac{x-\mu}{\lambda} - e^{-\frac{x-\mu}{\lambda}}}.$$

Cette loi est également appelée loi des valeurs extrêmes de type I, ou encore loi de GUMBEL [59]. Soit X, une variable aléatoire suivant une loi de GUMBEL de paramètres μ et λ, noté $X \hookrightarrow Gu(\mu, \lambda)$, est telle que $\mathbf{E}[X] = \mu - \gamma\,\lambda$, où $\gamma = 0{,}577\,2$ est la constante d'EULER-MASCHERONI, et $\mathbf{Var}[X] = \frac{\pi^2}{6}\,\lambda^2$. Cette loi est la forme limite de la distribution de la valeur maximale d'un échantillon de n valeurs, et est utilisée entre autres en hydrologie ; en effet, le débit des cours d'eaux maximum annuel est souvent décrit par cette loi [111].

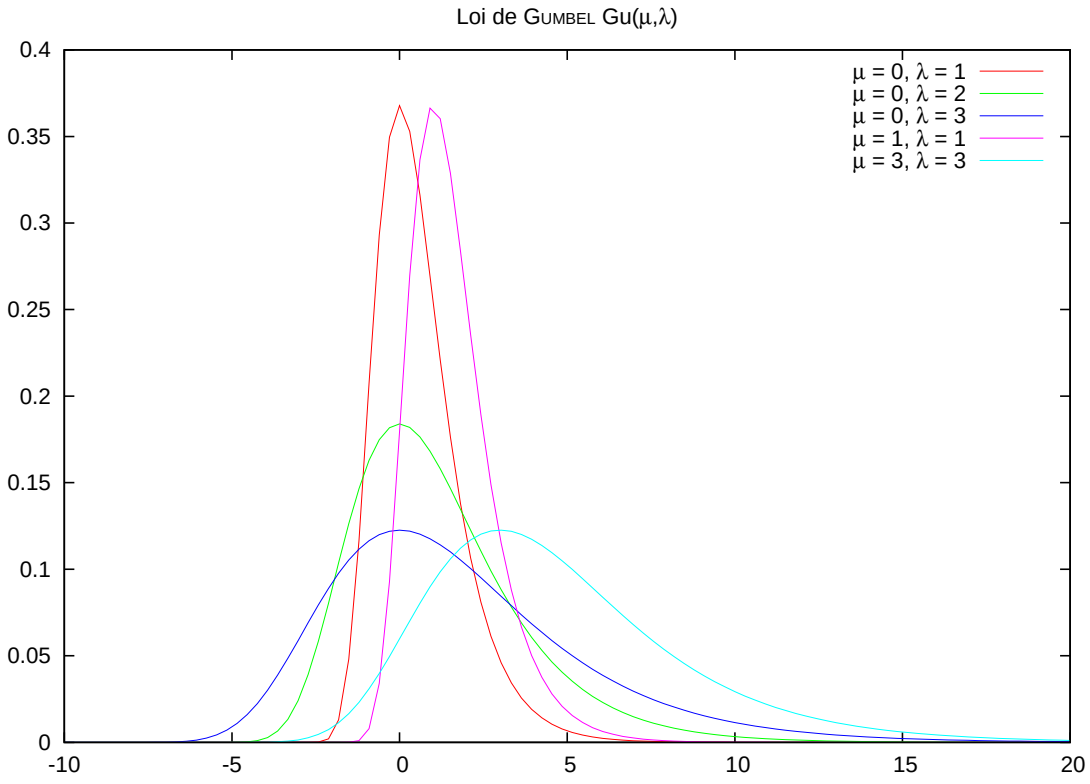


Figure E.4 – Loi de distribution de GUMBEL.

Annexe F

Illustration de STARS

Qui pourrait être plus arrogant qu'une étoile ? Et pourtant, regarde longtemps les étoiles et tu verras comme elles font honnêtement leur devoir. Aucune ne gêne l'autre, toutes s'aiment, chacune a sa place auprès de son père, un soleil, et elles ne se jettent pas toutes au même endroit pour profiter, pour réussir.

— Albert COHEN [1895–1981], Solal (1930).

L'objectif de cette annexe est d'illustrer le fonctionnement de STARS sur un exemple simple. Les différents algorithmes mis en jeu sont retranscrits afin de faciliter la lecture. Les étapes successives sont illustrées et commentées, plusieurs sorties produites par le programme sont également reproduites.

F.1 Présentation du programme

Avant de détailler une exécution de l'algorithme sur un jeu de séquences, une première présentation du programme semble nécessaire.

F.1.1 Résumé des options de la ligne de commande

Les différentes options du programme sont documentées dans la documentation fournie avec STARS aux formats `man`, `info`, `ps`, `pdf` et `html`. L'invocation du programme sans option ni argument ou avec l'option `--help` (ou `-h`) produit la sortie de l'exemple F.1.

Exemple F.1 (Résultat de l'invocation de STARS avec l'option `--help`).

```
bash$ stars --help

*****************************************************************************
*                                                                           *
* stars version 1.0.13, Copyright © 2002-2005 -- University of Nantes       *
* Author: Alban MANCHERON <alban.mancheron@univ-nantes.fr>                  *
*                                                                           *
* Stars comes with ABSOLUTELY NO WARRANTY.                                  *
* This is free software, and you are welcome to redistribute it under certain *
* conditions.                                                               *
* See the GNU General Public License for more details.                      *
*                                                                           *
*                                                                           *
* Usage:                                                                    *
*  stars [Options...] <file>                                                *
*  file:     Fasta formated file containing sequences to be processed.      *
*                                                                           *
* Options:                                                                  *
*  -p P                   Number of nodes to be developed (default 3).      *
*  -e E                   Number of consecutive mismatches (default 5).     *
*  -s score function      Scoring function to use (see --fct option).       *
*  -m similarity matrix   NCBI formated file describing (see --fct option). *
*  -q Q                   Quorum in % (default 100%, see --fct option).     *
*  -o output              Output text file (default stdout).                *
*  -O output              Output text file (even if exists).                *
*  -rna|-dna|-aa|-auto    Kind of biological data to process (default auto). *
*  -simulate              Don't process the file, just do a simulation.     *
*                                                                           *
* To obtain some function description: stars --fct                          *
* To obtain version number: stars --version (or stars -v)                   *
* To obtain this help: stars --help (or stars -h)                           *
*                                                                           *
*****************************************************************************
```

F.1.2 Fonctions disponibles

De même que les explications détaillées des options sont fournies dans la documentation livrée avec STARS, le détail des fonctions et de leur utilisation s'y trouve également. Il est possible de connaître les fonctions disponibles en invoquant le programme avec l'option `--fct` (*cf.* Exemple F.2).

Exemple F.2 (Résultat de l'invocation de STARS avec l'option `--fct`).

```
bash$ stars --fct

*******************************************************************************
*                                                                             *
* stars version 1.0.13, Copyright © 2002-2005 -- University of Nantes         *
* Author: Alban MANCHERON <alban.mancheron@univ-nantes.fr>                    *
*                                                                             *
* Stars comes with ABSOLUTELY NO WARRANTY.                                    *
* This is free software, and you are welcome to redistribute it under certain *
* conditions.                                                                 *
* See the GNU General Public License for more details.                        *
*                                                                             *
*******************************************************************************

Function n°  1: Score: Matrix-based Function (eg. BLOSUM, PAM, Gonnet, ...)
Function n°  2: Score: Pratt Function
Function n°  3: Score: Pratt-based Function
             ⋮
Function n°  6: Score: Q independent(max:id:id)
Function n°  7: Score: Q independent(max:id:nulle)
Function n°  8: Score: Q independent(max:id:carre)
             ⋮
Function n° 64: Score: Q dependent(max_Q:n_racine_n:suite)
Function n° 65: Score: Q dependent(max_Q:n_racine_n:n_racine_n)
```

F.2 Entrées/Sorties de l'algorithme

Les trois séquences (fictives) d'ADN ci-dessous sont fournies en entrée du programme dans un fichier (*e.g.* `fich.fasta`) au format FASTA.

```
     1     5     9
s1 - ACAATSTAC
s2 - AACGTCCAC
s3 - TTANCTACA
```

Le programme est invoqué (*cf.* Exemple F.3) avec les paramètres $p = 3$, $e = 2$ et la sixième fonction de score (*i.e.*, Fonction *Block-Based* Q-indépendante avec $f^{=}(k) = 2\,k$ et $f^{\neq}(k) = k$).

La sortie se décompose en plusieurs parties. Tout d'abord un entête comportant les informations de versions et les mentions légales (qui peut être obtenu avec l'option `--version` ou `-v`). Ensuite une partie récapitulative du paramétrage utilisé et d'informations complémentaires sur les séquences fournies en entrées. La troisième partie concerne les résultats de l'extraction des motifs. Ils sont synthétisés dans une table, indiquant la séquence de référence et les positions de début et de fin sur cette séquence

de chaque motif, ainsi que le score de la collection et le score pondéré (*cf.* Section 3.3.6 – page 72). Les séquences sont retranscrites en utilisant un système de balises de type XML pour repérer les occurrences de chaque motif sur les séquences. Enfin, une synthèse des informations relatives à l'exécution du programme (nombre de cycles effectués, temps d'utilisation, accès à la mémoire, ...) est proposée.

Exemple F.3 (Invocation de STAMS sur le fichier `fich.fasta` avec $p = 3$, $e = 2$ et $s = 6$).

```
bash$ stars -p 3 -e 2 -s 6 -auto fich.fasta

*****************************************************************************
*                                                                           *
* stars version 1.0.13, Copyright © 2002-2005 -- University of Nantes       *
* Author: Alban MANCHERON <alban.mancheron@univ-nantes.fr>                  *
*                                                                           *
* Stars comes with ABSOLUTELY NO WARRANTY.                                  *
* This is free software, and you are welcome to redistribute it under certain *
* conditions.                                                               *
* See the GNU General Public License for more details.                      *
*                                                                           *
*****************************************************************************

********************************** Data ************************************

P = 3
E = 2
Score: Q independent(max:id:id)
Q = 100%

Frequencies of bases from file fich.fasta:
(We suppose that sequences are Nucleic Acids. If they are not, you have to speci
fy it.)
 A: 0.407407
 C: 0.37037
 G: 0.111111
 T: 0.259259

Length of sequences:
 Average:         9 characters.
 Minimum:         9 characters.
 Maximum:         9 characters.

******************************** STARS Result ******************************

  List of found Patterns:

-------------------------------------------------------------------------------
  Model    Seqref   LBnd   UBnd   Score   DScore   Pattern
-------------------------------------------------------------------------------
     1        3       3      8        8        8   ANxTxC
     2        3       7      9        6        2   xCA
     3        3       1      5        4      1.6   TxxNC
-------------------------------------------------------------------------------

  Patterns in sequences:

> sequence n°  n° 1
<2>ACA</2><1>A<3>TSTAC</3></1>
```

```
> sequence n°  n° 2
A<1>ACG<3>T<2>CC</1>A</2>C</3>

> sequence n°  n° 3
<3>TT<1>ANC</3>T<2>AC</1>A</2>

********************************** Infos ************************************

  Last Modifying cycle:7
  Total Number of cycles: 17
  User time used:0,20000 sec.
  System time used:0,0 sec.
  Maximum resident set size:0
  Integral shared memory size:0
  Integral unshared data size:0
  Integral unshared stack size:0
  Page reclaims:39
  Page faults:293
  Swaps:0
  Block input operations:0
  Block output operations:0
  Messages sent:0
  Messages received:0
  Signals received:0
  Voluntary context switches:0
  Involuntary context switches:0

*****************************************************************************

That's All, Folks!!!
```

*Il est possible de remarquer que le total des fréquences des bases est supérieur à 1. Ceci est dû au fait que les séquences comportent des symboles de l'alphabet dégénéré de l'*ADN *(cf. Annexe A.1).*

F.3 Détail sur l'exécution de l'algorithme

L'algorithme F.1 (copie de l'algorithme 3.6 – page 69) commence par une phase de pré-traitement des séquences fournies par l'utilisateur (lignes 17 à 28). Ensuite, plusieurs cycles sont effectués (boucle de la ligne 33 à la ligne 46) correspondant chacun à une ordre particulier de traitement des séquences. Puis enfin, les résultats sont affichés (ligne 47). Plutôt que de reprendre l'intégralité de l'exécution, seule un cycle est présenté, celui correspondant aux résultats de l'exemple F.3 (il n'est pas systématique – ce serait même plutôt rare – que les résultats proviennent du même cycle, bien que cela soit le cas dans cet exemple).

F.3.1 Pré-traitement des séquences

L'algorithme de création des tableau de stockage des positions des séquences présenté dans cette annexe intègre la modification de l'algorithme 3.3 – page 64 – proposée dans la section 3.3.3 – page 62. Les tables crées par l'algorithme F.2 sont reproduites dans le tableau de la page 224.

```
Nom : STABS
Entrées : S = {s_1, ..., s_t} % Ensemble des séquences à traiter. %
  Σ, C(Σ) % Alphabet et couverture de l'alphabet utilisé. %
  p % Nombre de résultats à renvoyer. %
  f : Σ* × Σ* → ℝ % Fonction de score. %
Sortie : Sol % Ensemble des résultats. %
Variables : OldSol % Ensemble des résultats avant le dernier cycle. %
  ℓ % Numéro de la séquence de référence %
  k % indice de la séquence en cours de traitement %
  π % Permutation sur S (s'_i = s_{π_i}). %
  nb_cycles % Compteur de cycles. %
  TStock % Vecteur de taille t pour le pré-traitement des séquences. %
  TShared % Vecteur de taille 2n-1 pour les décalages entre 2 séquences. %
  TCand % Matrice Carrée de taille t pour le traitement de TShared. %
Début
  % Pré-traitement des séquences. %
  Pour ℓ de 1 à t Faire
    TStock[ℓ] ← Créer_Tableau_Stockage(s_ℓ). % Algorithme F.2, p.224 %
  Fin Pour

  Pour ℓ de 1 à t Faire
    Pour k de 1 à t Faire
      Si ℓ ≠ k Alors
        TShared ← Créer_Tableau_Correspondances(s_ℓ, TStock[k]). % Algorithme F.3, p.225 %
        TCand[ℓ, k] ← Créer_Tableau_Candidats(TShared). % Algorithme F.4, p.226 %
      Fin Si
    Fin Pour
  Fin Pour

  % Processus d'extraction. %
  Sol ← ∅, OldSol ← ∅.
  ℓ ← 0, nb_cycles ← 0.
  Répéter
    % Choix de la séquence de référence, puis de la permutation. %
    ℓ ← (ℓ mod t) + 1.
    Choisir une permutation π de S telle que s'_1 = s_ℓ.
    % Calcul de la solution pour cette permutation. %
    Sol ← Sol ∪ STABS-χ(S, π, p, f). % Algorithme F.5, p.229 %
    Trier Sol par scores ↘ et ne conserver que les p meilleurs résultats.
    % Mise à jour du nombre de cycles exempts de modification. %
    Si Sol ≠ OldSol Alors
      nb_cycles ← 0.
    Sinon
      nb_cycles ← nb_cycles + 1.
    Fin Si
  Tant Que (nb_cycles < t^2)
  Afficher et Retourner Sol.
Fin
```

Algorithme F.1 – Algorithme d'extraction de motifs STABS.

```
1  Nom : Créer_Tableau_Stockage_modifié
2  Entrée : s % Séquence à traiter. %
3  Sortie : Stock_s % Tableau de Stockage de taille |Σ|. %
4  Variable : k % Position courante dans la séquence. %
5  Début
6    Pour Chaque α ∈ Σ Faire
7      Stock_s[α] ← ∅
8    Fin Pour
9    Pour k de 1 à |s| Faire
10     Pour Tout α ∈ Σ tel que α =_{C(Σ)} s[k] Faire
11       Ajouter k à la fin de la liste Stock_s[α].
12       % k est nécessairement le plus grand élément de Stock_s[α]. %
13     Fin Pour
14   Fin Pour
15   Retourner Stock_s
16 Fin
```

Algorithme F.2 – Pré-traitement des séquences (version modifiée).

	Σ			
	A	C	G	T
$Stock_{s_1}$	1, 3, 4, 8	2, 6, 9	6	5, 7
$Stock_{s_2}$	1, 2, 8	3, 6, 7, 9	4	5
$Stock_{s_3}$	3, 4, 7, 9	4, 5, 8	4	1, 2, 4, 6

Table F.1 – Tables de stockage des positions dans les séquences.

À partir de ces tables de stockage sont créées les tables de positions en « correspondances » entre une séquence considérée comme séquence de référence et les autres séquences (*cf.* Algorithme F.3, reproduction de l'algorithme 3.4 – page 65). Ensuite, ces tables de positions sont utilisées par l'algorithme F.4 pour créer les tables de candidats. Ce dernier algorithme intègre la modification de l'algorithme 3.5 – page 66 – proposée à la section 3.3.3 – page 62.

Plutôt que de représenter l'ensemble des tables, seules celles utilisées dans la section suivante sont proposées dans la table F.2 – page 227. Elles correspondent au cas où la séquence de référence est la séquence s_3.

```
1  Nom : Créer_Tableau_Correspondances
2  Entrées : s % séquence de référence. %
3    Stock_s' % Tableau de stockage associé à une séquence s'. %
4  Sortie : Shared_{s,s'} % Tableau des positions de s en correspondances dans s' %
5                  % pour chaque alignement donné par un décalage δ. %
6  Variables : δ % Décalages entre s et s'. %
7    k,l % indices de positions des séquences s et s'. %
8  Début
9    Pour δ de 1 − |s_i| à |s_j| − 1 Faire
10     Shared_{s,s'}[δ] ← ∅
11   Fin Pour
12   Pour k de 1 à |s| Faire
13     Pour Chaque α ∈ Σ Faire
14       Si s[k] =_{C(Σ)} α Alors
15         Pour Chaque l dans Stock_s'[α] Faire
16           % Les listes Stock_s'[α] sont ordonnées par construction. %
17           δ ← l − k
18           % En raison de la dégénérescence de l'alphabet, %
19           Si le dernier élément de Shared_{s,s'}[δ] n'est pas k Alors
20             % il faut prévenir l'ajout potentiel de doublons. %
21             Ajouter k à la fin de la liste Shared_{s,s'}[δ]
22           Fin Si
23           % Les listes Shared_{s,s'}[δ] sont donc également ordonnées. %
24         Fin Pour
25       Fin Si
26     Fin Pour
27   Fin Pour
28   Retourner Shared_{s,s'}
29 Fin
```

Algorithme F.3 – Extraction des positions en « correspondances » entre deux séquences.

```
1  Nom : Créer_Tableau_Candidats
2  Entrée : Shared_{s,s'} % Tableau des positions de s en correspondances dans s' %
3                          % pour chaque alignement donné par le décalage δ. %
4  Sortie : Cand_{s,s'} % Tableau des candidats de s ayant une occurrence dans s' %
5                       % pour chaque alignement donné par le décalage δ. %
6  Variables : δ % Décalages entre s et s'. %
7    k,l % indices de positions des séquences s et s'. %
8  Début
9    Pour δ de 1−|s| à |s'|−1 Faire
10     deb ← 0 % Pour chaque nouveau décalage, il n'y a pas encore de motif. %
11     Pour Chaque k de la liste Shared_{s,s'}[δ] Faire
12       % La liste est ordonnée par construction. %
13       Si deb = 0 Alors % Pas encore de motif en cours d'extraction. %
14         deb ← k % Un nouveau motif débute à la position courante dans s, %
15         fin ← deb % et se termine au plus tôt à cette même position. %
16       Sinon % Motif en cours d'extraction. %
17         Si (k − fin − 1 ≤ e) Alors % Non correspondances consécutives. %
18           fin ← k % Le motif s'achève au plus tôt à la position courante. %
19         Sinon % Le motif s[deb..fin] est un nouveau candidat. %
20           Si deb ≠ fin Alors
21             Ajouter [deb; fin] à la fin de la liste Cand_{s,s'}[δ].
22           Fin Si
23           deb ← k % Un nouveau motif débute à la position courante dans s, %
24           fin ← deb % et se termine au plus tôt à cette même position . %
25         Fin Si
26       Fin Si
27     Fin Pour
28     Si deb ≠ 0 et deb ≠ fin Alors % Il reste un motif en à ajouter. %
29       Ajouter [deb; fin] à la fin de la liste Cand_{s,s'}[δ].
30     Fin Si
31    % La liste Cand_{s,s'}[δ] est également ordonnée. %
32   Fin Pour
33   Retourner Cand_{s,s'}.
34 Fin
```

Algorithme F.4 – Extraction des motifs e-similaires entre deux séquences.

		$Shared_{s_3,s_1}$	$Shared_{s_3,s_2}$	$Cand_{s_3,s_1}$	$Cand_{s_3,s_2}$
δ	-8	9	9		
	-7		9		
	-6	$7,8,9$	7	$[7;9]$	
	-5	9	$7,8$		$[7;8]$
	-4	7			
	-3	$4,5,7$	4	$[4;7]$	
	-2	$3,4,8$	$3,4,5,8$	$[3;4]$	$[3;8]$
	-1	$4,6,9$	$3,4,6,8,9$	$[4;9]$	$[3;9]$
	0	$3,4$	4	$[3;4]$	
	1	$3,4,5,6,7,8$	$4,5,7,8$	$[3;8]$	$[4;8]$
	2	4	$4,5$		$[4;5]$
	3	$2,4$	$2,4$	$[2;4]$	$[2;4]$
	4	$1,4,5$	$1,4,5$	$[1;5]$	$[1;5]$
	5	$2,3,4$	$3,4$	$[2;4]$	$[3;4]$
	6	1			
	7				
	8				

Table F.2 – Tables de positions en « correspondances » et tables des candidats avec s_3 comme séquence de référence.

F.3.2 Exécution d'un cycle de l'algorithme

Un cycle de l'algorithme correspond essentiellement au choix d'une permutation (ligne 36), et à l'exécution de l'algorithme STARS-χ (ligne 39 ; *cf.* Algorithme F.5, reproduction de l'algorithme F.5 – page ci-contre). Dans cet exemple, les séquences sont traitées dans l'ordre s_3, s_2, s_1.

La figure F.1 correspond à la création de la racine λ de l'arbre des solutions (lignes 16 et 17).

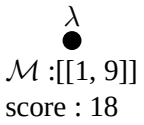


Figure F.1 – Construction de l'arbre des solutions de STARS (1/6).

Les figures F.2 et F.3 correspondent à la première itération de la boucle principale (lignes 18 à 30). Les variables sont donc $Niv = 2$, $Feuilles = \{\lambda\}$. Il n'y a par conséquent qu'un seul passage dans la boucle de la ligne 20 à la ligne 27. La figure F.2 correspond à la création des fils de la racine, tandis que la figure F.3 représente l'arbre après élagage (ligne 28).

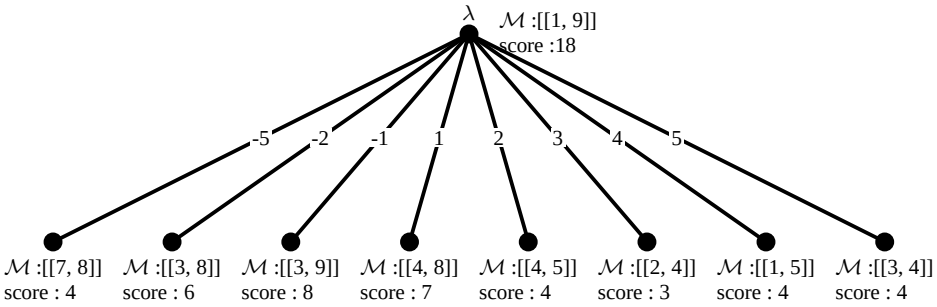


Figure F.2 – Construction de l'arbre des solutions de STARS (2/6).

Il est possible de remarquer que seuls deux nœuds sont conservés après l'élagage alors que le paramètre p est fixé à 3. Ceci est dû au fait que les listes de candidats associés aux autres nœuds sont entièrement incluses dans l'une des listes $\mathcal{L}(n_1)$ et $\mathcal{L}(n_2)$ (*cf.* Section 3.3.6 – page 72).

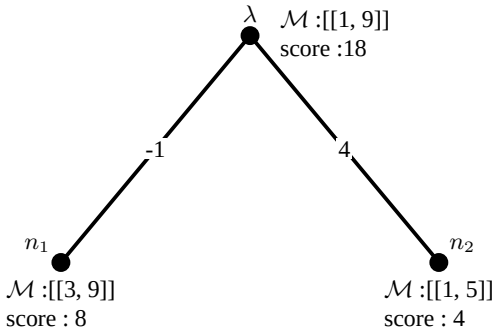


Figure F.3 – Construction de l'arbre des solutions de STARS (3/6).

Les figures F.4 à F.6 – pages 230 à 231 – correspondent, quant à elles, au second passage dans la

Nom : STABS-χ
Entrées : $\mathcal{S} = \{s_1, \ldots, s_t\}$ % *Ensemble des séquences à traiter.* %
  π % *Permutation sur* $\mathcal{S}$ ($s'_i = s_{\pi_i}$). %
  p % *Nombre de résultats à renvoyer.* %
  $f : \Sigma^* \times \Sigma^* \to \mathbb{R}$ % *Fonction de score.* %
Sortie : *res* % *Ensemble de solutions pour ce cycle.* %
Variables : λ, u, v, w % *Nœuds.* %
  $\mathcal{A}$. % *Arbre des solutions.* %
  δ % *décalage entre deux séquences.* %
  Feuilles % *Liste ordonnée (par scores* $\searrow$*) des* p *meilleurs nœuds.* %
  Liste % *Liste temporaire de nœuds.* %
  Niv % *Profondeur de l'arbre en construction.* %
  chemin % *pile.* %
Début
  % *Création de l'arbre des solutions.* %
  Créer la racine λ de l'arbre $\mathcal{A}$.
  $\mathcal{L}(\lambda) \leftarrow \{[1..|s'_1|]\}$, $score(\lambda) \leftarrow +\infty$, $Liste \leftarrow \{\lambda\}$.
  Pour *Niv* de 2 à t **Faire**
    $Feuilles \leftarrow \emptyset$.
    Pour Chaque nœud $u \in Liste$ **Faire**
      Pour Chaque décalage δ entre s'_1 et s'_{Niv} **Faire**
        Créer un nouveau nœud v, ainsi qu'un arc de u vers v étiqueté par δ.
        Calculer $\mathcal{L}(v)$ à partir de $\mathcal{L}(u)$ pour l'alignement donné par δ.
        $score(v) \leftarrow \min(score(u), \max(\{f(s'_1[a..b], s'_{Niv}[a+\delta..b+\delta]) \mid [a..b] \in \mathcal{L}(v)\}))$.
        Ajouter v dans *Feuilles*.
      Fin Pour
    Fin Pour
    Trier *Feuilles* par scores $\searrow$ et ne conserver que les p premiers éléments.
    $Liste \leftarrow Feuilles$.
  Fin Pour

  % *Parcours de l'arbre des solutions.* %
  $res \leftarrow \emptyset$.
  Pour Chaque feuille v de l'arbre $\mathcal{A}$ **Faire**
    $chemin \leftarrow \$$. % *Pile vide.* %
    $u \leftarrow v$.
    Tant Que $u \neq \lambda$ **Faire**
      empiler l'étiquette de l'arc $(Pere(u), u)$ sur *chemin*.
      $u \leftarrow Pere(u)$.
    Fin Tant Que
    $res \leftarrow res \cup (\mathcal{L}(v), score(v), chemin, \pi)$.
  Fin Pour Chaque
  Retourner *res*
Fin

Algorithme F.5 – Construction de l'arbre des solutions.

boucle de la ligne 20 à la ligne 27. Les variables sont telles que $Niv = 3$ et $Feuilles = \{n_1, n_2\}$. Les figures F.4 et F.5 correspondent respectivement au premier et au second (et dernier) passage dans la boucle de la ligne 21 à la ligne 26 (*i.e.*, le développement respectifs des nœuds n_1 et n_2).

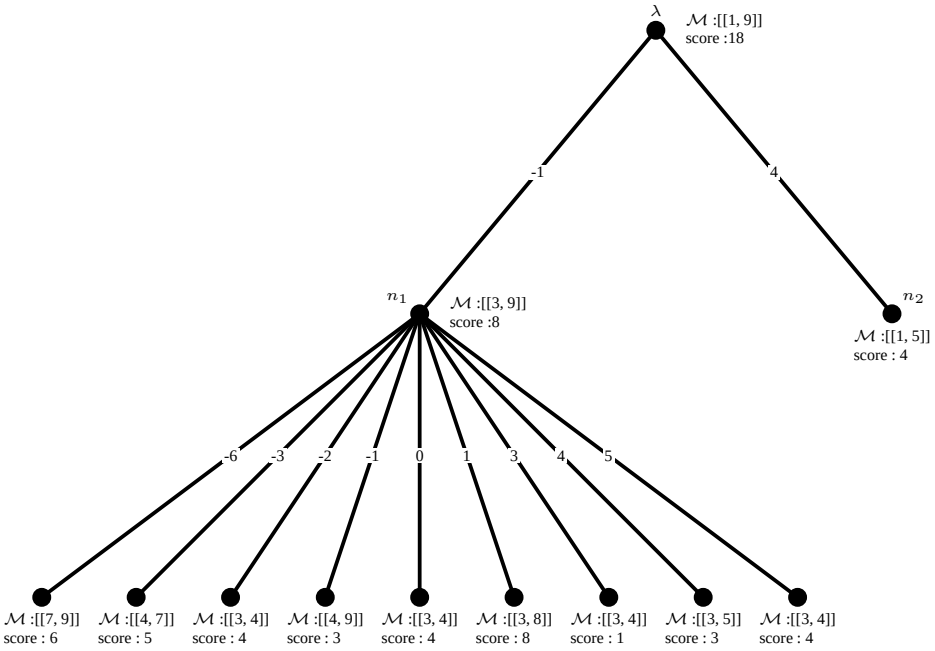


Figure F.4 – Construction de l'arbre des solutions de STARS (4/6).

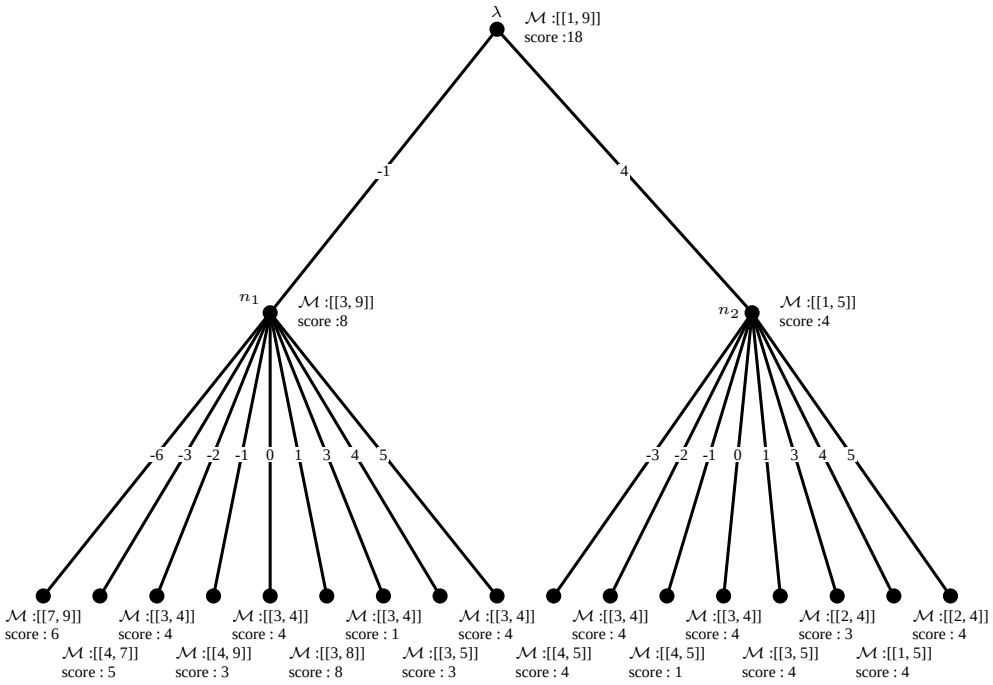


Figure F.5 – Construction de l'arbre des solutions de STARS (5/6).

La figure F.6 correspond à l'élagage de l'arbre (ligne 28).

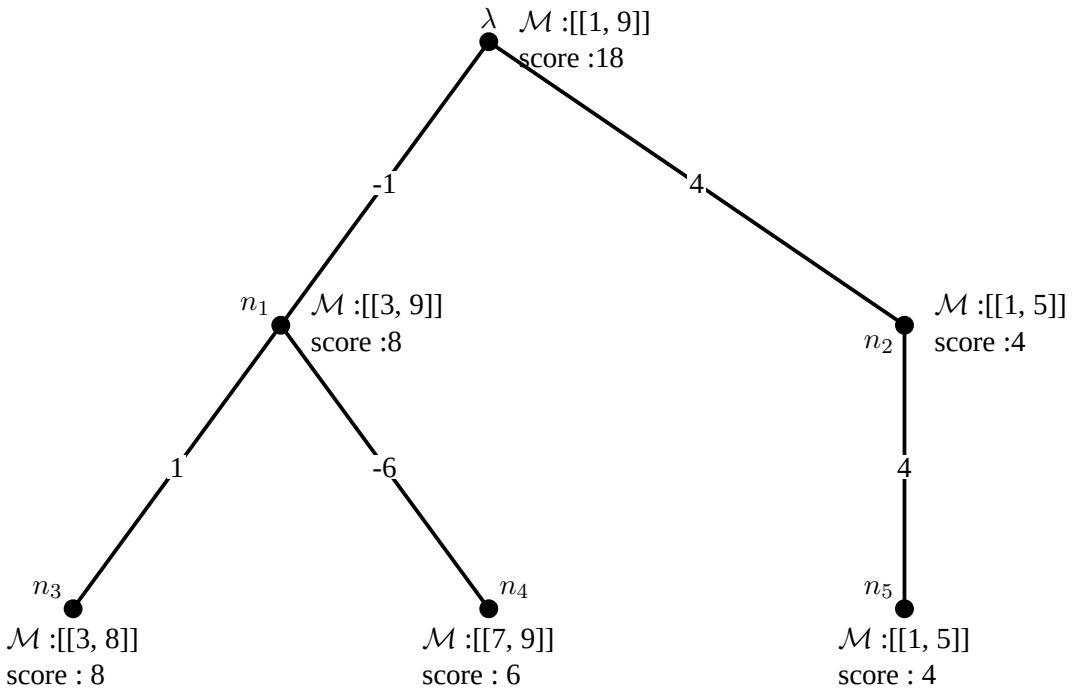


Figure F.6 – Construction de l'arbre des solutions de STABS (6/6).

À l'issue de la boucle principale (lignes 18 à 30), les feuilles de l'arbre sont les nœuds n_3, n_4 et n_5. Le traitement des lignes 34 à 42 permet de récupérer les différentes informations nécessaires à l'affichage des résultats (*cf.* Exemple F.4).

Exemple F.4 (Résultats de STABS sur le fichier `fich.fasta` avec $p = 3$, $e = 2$ et $s = 6$).

```
-------------------------------------------------------------------------------
  Model    Seqref   LBnd   UBnd   Score   DScore   Pattern
-------------------------------------------------------------------------------
     1        3       3      8        8        8   ANxTxC
     2        3       7      9        6        2   xCA
     3        3       1      5        4      1.6   TxxNC
-------------------------------------------------------------------------------

  Patterns in sequences:

> sequence n°  n° 1
<2>ACA</2><1>A<3>TSTAC</3></1>

> sequence n°  n° 2
A<1>ACG<3>T<2>CC</1>A</2>C</3>

> sequence n°  n° 3
<3>TT<1>ANC</3>T<2>AC</1>A</2>
```

Bibliographie

[1] Alfred V. AHO et Margaret J. CORASICK.
Efficient string matching: an aid to bibliographic search.
Communication ACM, 18(6):333–340, 1975.
Cette référence a été citée à la page 35.

[2] Cyril ALLAUZEN, Maxime CROCHEMORE et Mathieu RAFFINOT.
Factor Oracle: A New Structure for Pattern Matching.
Dans *Conference on Current Trends in Theory and Practice of Informatics*, pages 295–310, 1999.
Cette référence a été citée à la page 233.

[3] Cyril ALLAUZEN, Maxime CROCHEMORE et Mathieu RAFFINOT.
Oracle des facteurs, Oracle des Suffixes.
Rapport technique 99–08, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999.
Voir également [2] et [4].
Cette référence a été citée aux pages 143, 146, 150, 164, 165.

[4] Cyril ALLAUZEN, Maxime CROCHEMORE et Mathieu RAFFINOT.
Efficient Experimental String Matching by Weak Factor Recognition.
Dans *Proceedings of the 12th Conference on Combinatorial Pattern Matching (*CPM*)*, volume 2089 de *Lecture Notes in Computer Science (*LNCS*)*, pages 51–72. Springer-Verlag, 2001.
Version étendue (communication personnelle).
Cette référence a été citée aux pages 161, 163, 233.

[5] Richard ALTMANN.
Ueber Nucleinsäure.
Archiv für Physiologie, pages 524–536, 1889.
Cette référence a été citée à la page 3.

[6] Stephen F. ALTSCHUL, Ralf BUNDSCHUH, Rolf OLSEN et Terence HWA.
The estimation of statistical parameters for local alignment score distributions.
Nucleic Acids Research, 29(2):351–361, 2001.
Cette référence a été citée à la page 27.

[7] Stephen F. ALTSCHUL, Warren R. GISH, Webb MILLER, Eugène W. MYERS et David J. LIPMAN.
A Basic Local Alignment Search Tool.
Journal of Molecular Biology, 215:403–410, 1990.
Cette référence a été citée à la page 83.

[8] Gérard ASSAYAG et Shlomo DUBNOV.
Using Factor Oracles for machine Improvisation.
Soft Computing, 8(9):604–610, 2004.
Cette référence a été citée à la page 163.

[9] Timothy BAILEY et Charles ELKAN.
Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers.
Dans *Proceedings of the 2nd International Conference on Intelligent System of Molecular Biology (*ISMB*)*, volume 80, pages 28–36. AAAI Press, 1994.
Cette référence a été citée aux pages 39, 40.

[10] Timothy BAILEY et Charles ELKAN.
Unsupervised Learning of Multiple Motifs in Biopolymers Using Expectation Maximization.
Machine Learning, 21(1-2):51–80, 1995.
Cette référence a été citée à la page 77.

[11] Timothy BAILEY et Michael GRIBSKOV.
Combining evidence using p-values: Application to sequence homology searches.
Bioinformatics, 14(1):48–54, 1998.
Disponible à l'adresse
http://www.sdsc.edu/MEME.
Cette référence a été citée à la page 77.

[12] Vladimir BATAGELJ et Matevž BREN.
Comparing Resemblance Measures.
Journal of Classification, 12(1):73–90, 1995.
Cette référence a été citée à la page 30.

[13] Dennis A. BENSON, Mark S. BOGUSKI, David J. LIPMAN, James OSTELL et B. F. Francis OUELLETTE.
GenBank.
Nucleic Acids Research, 26(1):1–7, 1998.
Cette référence a été citée à la page 76.

[14] Jérémie BOURDON.
Analyse dynamique d'algorithmes : exemples en arithmétique et en théorie de l'information.
Thèse de Doctorat, Université de Caen, France, 2002.
Cette référence a été citée aux pages 85, 87.

[15] Jérémie BOURDON et Alban MANCHERON.
Statistical Properties of Similarity Score Functions.
Dans *Proceedings of the 4th Colloquium on Mathematics and Computer Science. Algorithms, Trees, Combinatorics and Probabilities*, Discrete Mathematics and Theoretical Computer Science, DMTCS, pages 129–140, 2006.
Disponible à l'adresse
http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/.
Cette référence a été citée aux pages 83, 105, 109, 110, 111.

[16] Robert Stephen BOYER et J. Strother MOORE.
A fast string searching algorithm.
Communication ACM, 20(10):762–772, 1977.
Cette référence a été citée à la page 35.

[17] Andreas BRANDSTÄDT, Van Bang LE et Jeremy P. SPINRAD.
Graph Classes: A Survey.
SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, USA, 1999.
Cette référence a été citée à la page 205.

[18] Alvis BRAZMA, Inge JONASSEN, Ingvar EIDHAMMER et David GILBERT.
Approaches to the Automatic Discovery of Patterns in Biosequences.
Journal of Computational Biology, 5(2):277–304, 1998.
Cette référence a été citée aux pages 33, 38, 39, 193.

[19] Broňa BREJOVÁ, Chrysanne DIMARCO, Tomáš VINAŘ, Sandra Romero HIDALGO, Gina HOGUIN et Cheryl PATTEN.
Finding Patterns in Biological Sequences.
Rapport technique CS-2000-22, University of Waterloo, 2000.
Disponible à l'adresse
http://genetics.uwaterloo.ca/~tvinar/cs798g/motif.
Cette référence a été citée aux pages 5, 37.

[20] Philipp BUCHER et Amos BAIROCH.
A Generalized Profile Syntax for Biomolecular Sequence Motifs and its Function in Automatic Sequence Interpretation.
Dans *Proceedings of the 2nd International Conference on Intelligent System of Molecular Biology (*ISMB*)*, pages 53–61. AAAI Press, 1994.
Cette référence a été citée à la page 192.

[21] Jeremy BUHLER et Martin TOMPA.
Finding Motifs Using Random Projections.
Dans *Proceedings of the 5th Annual International Conference on Computational Molecular Biology (*RECOMB*)*, pages 69–76, New York, USA, 2001. ACM Press.
Cette référence a été citée aux pages 39, 41.

[22] Andrea CALIFANO.
SPLASH: Structural Pattern Localization Analysis by Sequential Histograms.
Bioinformatics, 16(4):341–357, 2000.
Cette référence a été citée aux pages 5, 39, 41.

[23] William I. CHANG et Thomas G. MARR.
Approximate String Matching and Local Similarity.
Dans *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (*CPM*)*, pages 259–273. Springer-Verlag, 1994.
Cette référence a été citée à la page 36.

[24] Inaam CHARAF.
Dispositifs d'organisation de la connaissance, étude comparée d'encylopédies, de classifications et de Thesaurus en anglais, français et arabe.
Thèse de Doctorat, Université Claude Bernard, Lyon I, France, 2005.
volume 1.
Cette référence a été citée à la page 16.

[25] Christian CHARRAS et Thierry LECROQ.
Handbook of Exact String Matching Algorithms.
King's College London Publications, 2004.
Cette référence a été citée à la page 35.

[26] Rajagopal CHATTOPADHYAYA, Kausik GHOSH et V. M. Haridasan NAMBOODIRI.
Model of a LexA repressor dimer bound to recA operator.
Journal of Biomolecular Structure and Dynamics, 18(2):181–198, 2000.
Cette référence a été citée à la page 15.

[27] Jean-Michel CLAVERIE, Stéphane AUDIC et Chantal ABERGEL.
La BioInformatique : une discipline stratégique pour l'analyse et la valorisation des génomes, 2000.
Disponible à l'adresse

http://igs-server.cnrs-mrs.fr/.
Cette référence a été citée à la page 4.

[28] Loek G.W.A. CLEOPHAS, Gerard ZWAAN et Bruce W. WATSON.
Constructing Factor Oracles.
Dans *Proceedings of the 3rd Prague Stringology Conference (*PSC*)*, pages 37–50, 2003.
Version corrigée (communication personnelle).
Cette référence a été citée à la page 161.

[29] Jack S. COHEN et Franklin H. PORTUGAL.
The Search for the Chemical Structure of DNA.
Connecticut Medecine, 38(10):551–557, 1974.
Cette référence a été citée à la page 3.

[30] Jacques COHEN.
Bioinformatics – an introduction for computer scientists.
ACM *Computing Surveys*, 36(2):122–158, 2004.
Cette référence a été citée à la page 5.

[31] Graham CORMODE.
Sequence Distance Embeddings.
Thèse de Doctorat, University of Warwick, UK, 2003.
Cette référence a été citée à la page 30.

[32] Francis Harry Compton CRICK, Leslie BARNETT, Sydney BRENNER et Richard J. WATTS-TOBIN.
General Nature of the Genetic Code for Proteins.
Nature, 192(4809):1227–1232, 1961.
Cette référence a été citée à la page 4.

[33] Maxime CROCHEMORE et Wojciech RYTTER.
Text algorithms.
Oxford University Press, Inc., 1994.
Cette référence a été citée aux pages 35, 38, 142.

[34] Ralf DAHM.
Friedrich MIESCHER and the discovery of DNA.
Developmental Biology, 278(2):274–288, 2005.
Cette référence a été citée à la page 3.

[35] Margaret Oakley DAYHOFF, Robert M. SCHWARTZ et Bruce C. ORCUTT.
A model of evolutionary change in proteins.
Dans Margaret Oakley DAYHOFF, réd., *Atlas of Protein Sequence and Structure*, volume 5, chapitre 22, pages 345–352. National Biomedical Research Foundation, 1978.
Cette référence a été citée à la page 32.

[36] Jean-Pierre DELMAS.
Introduction aux Probabilités.
Collection Pédagogique des Télécommunications. Ellipses, Paris, France, deuxième édition, 2000.
Cette référence a été citée aux pages 84, 110.

[37] William Edwards DEMING.
Statistical Adjustment of Data.
Dover Publications, 1943.

Cette référence a été citée à la page 111.

[38] Alain DENISE, Mireille RÉGNIER et Mathias VANDENBOGAERT.
Assessing the Statistical Significance of Overrepresented Oligonucleotides.
Dans Olivier GASCUEL et Bernard M. E. MORET, réds., *Algorithms in Bioinformatics. Proceedings of the 1st International Workshop on Algorithms in BioInformatics (*WABI*)*, volume 2149 de *Lecture Notes in Computer Science (*LNCS*)*, pages 85–97, London, UK, 2001. Springer-Verlag.
Cette référence a été citée aux pages 40, 83.

[39] Thomas Glen DIETTERICH et Ryszard MICHALSKI.
Discovering Patterns in Sequences of Events.
Artificial Intelligence, 25(2):187–232, 1985.
Cette référence a été citée à la page 37.

[40] Ingvar EIDHAMMER, Inge JONASSEN et William R. TAYLOR.
Structure Comparison and Structure Patterns.
Rapport technique 174, Department of Informatics, University of Bergen, Norway, 1999.
Cette référence a été citée à la page 83.

[41] Robert Rossenbeck FEULGEN.
Mikroskopisch-chemischer Nachweis einer Nucleinsäure von Typus der Thymonucleinsäure und die darauf beruhende elektive Färbung von Zellkernen in mikroskopischer Präparaten.
Hoppe-Seyler's Zeitschrift für Physiologische Chemie, 135:203–248, 1924.
Cette référence a été citée à la page 3.

[42] Philippe FLAJOLET et Andrew ODLYZKO.
Singularity Analysis of Generating Functions.
SIAM *Journal on Discrete Mathematics*, 3(2):216–240, 1990.
Cette référence a été citée à la page 87.

[43] Philippe FLAJOLET et Robert SEDGEWICK.
Analytic Combinatorics.
À paraître.
Disponible à l'adresse
http://algo.inria.fr/flajolet/Publications/books.html.
Cette référence a été citée aux pages 85, 87, 93.

[44] Rosalind Elsie FRANKLIN et Raymond GOSLING.
Evidence for 2-Chain Helix in Crystalline Structure of Sodium Deoxyribonucleate.
Nature, 172:156–157, 1953.
Cette référence a été citée aux pages 3, 12.

[45] Rosalind Elsie FRANKLIN et Raymond GOSLING.
Molecular Configuration in Sodium Thymonucleate.
Nature, 171:740–741, 1953.
Cette référence a été citée à la page 3.

[46] Edward FREDKIN.
Trie Memory.
Communication ACM, 3(9):490–499, 1960.
Cette référence a été citée à la page 141.

[47] Eileen A. FRIEDMAN et Hamilton O. SMITH.

An Adenosine Triphosphate-dependent Deoxyribonuclease from *Hemophilus influenzae* Rd. I. Purification and properties of the enzyme.
Journal of Biological Chemistry, 247(9):2846–2853, 1972.
Cette référence a été citée à la page 4.

[48] Dominique FRÉMY, Michèle FRÉMY et Fabrice FRÉMY.
Quid 2000.
Robert Laffont, 1999.
Cette référence a été citée à la page 3.

[49] FULCANELLI.
Les demeures philosophales et le symbolisme hermétique dans ses rapports avec l'art sacré et l'ésotérisme du Grand Œuvre, volume 2.
Pauvert, 1977.
Cette référence a été citée à la page IV.

[50] George GAMOW et Martynas YCAS.
The cryptographic Approach to the Problem of Protein Synthesis.
Dans New York PERGAMON PRESS, réd., *Symposium on Information Theory in Biology*, pages 63–69. Hubert P. Yockey, Robert Platzman and Henry Quastler, 1958.
Cette référence a été citée à la page 185.

[51] Robert GIEGERICH, Stefan KURTZ et Jens STOYE.
Efficient Implementation of Lazy Suffix Trees.
Software: Practice and Experience, 33(11):1035–1049, 2003.
Cette référence a été citée à la page 145.

[52] Robert L. GOLDSTONE.
Similarity.
Dans R. A. WILSON et F. C. KEIL, réds., MIT *encyclopedia of the cognitive sciences*, pages 757–759. MIT Press, Cambridge, MA., 1999.
Cette référence a été citée à la page 25.

[53] Gaston H. GONNET, Mark A. COHEN et Steven A. BENNER.
Exhaustive matching of the entire protein sequence database.
Science, 256:1443–1444, 1992.
Cette référence a été citée aux pages 32, 201.

[54] Ronald Lee GRAHAM, Donald Ervin KNUTH, Oren PATASHNIK et Alain DENISE.
Mathématiques concrètes : Fondations pour l'informatique.
Vuibert, deuxième édition, 2003.
Cette référence a été citée aux pages 84, 86, 88, 91, 93, 97, 265.

[55] Michael GRIBSKOV, Andrew MCLACHLAN et David EISENBERG.
Profile analysis: detection of distantly related proteins.
*Proceedings of National Academy of Science (*PNAS*)*, 84:4355–4358, 1987.
Cette référence a été citée à la page 37.

[56] Anthony J.F. GRIFFITHS, William M. GELBART, Jeffrey H. MILLER et Richard C. LEWONTIN.
Modern Genetic Analysis, chapitre 7. Gene Mutations.
W. H. FREEMAN & Co, 1999.
Cette référence a été citée aux pages 16, 37.

[57] William Noble GRUNDY, Charles Elkan TIMOTHY BAILEY et Michael BAKER.
Meta-MEME: Motif-based Hidden Markov Models of Biological Sequences.

*Computer Applications in the Biosciences (*CABIOS*)*, 13(4):397–406, 1997.
Cette référence a été citée aux pages 39, 41.

[58] Emil Julius GUMBEL.
Statisitics of Extremes.
Columbia University Press, New York, 1958.
Cette référence a été citée à la page 27.

[59] Emil Julius GUMBEL.
Distributions des valeurs extrêmes en plusieurs dimensions.
Publications de l'Institut de Statistiques, Paris, 9:171–173, 1960.
Cette référence a été citée aux pages 27, 216, 239.

[60] Emil Julius GUMBEL.
Multivariate Extremal Distributions.
Bulletin de l'Institut International de Statistiques, 37(2):461–475, 1960.
Version anglaise de [59].
Cette référence a été citée à la page 27.

[61] Dan GUSFIELD.
Algorithms on Strings, Trees, and Sequences: computer science and computational biology.
Cambridge University Press, 1997.
Cette référence a été citée à la page 35.

[62] Richard Wesley HAMMING.
Error detecting and error correcting codes.
The Bell System Technical Journal, 29(2):147–160, 1950.
Cette référence a été citée à la page 27.

[63] Franklin M. HAROLD.
To shape a cell: an inquiry into the causes of morphogenesis of microorganisms.
Microbilogy and Molecular Biology Reviews, 54(4):381–431, 1990.
Cette référence a été citée à la page 253.

[64] Steven HENIKOFF et Jorja G HENIKOFF.
Amino Acid Substitution Matrices from Protein Blocks.
*Proceedings of National Academy of Science (*PNAS*)*, 89(22):10915–10919, 1992.
Cette référence a été citée à la page 32.

[65] Patricia HERNANDEZ, Robin GRAS, Julien FREY et Ron D. APPEL.
POPITAM: Towards new heuristic strategies to improve protein identification from tandem mass spectrometry data.
Proteomics, 3(6):870–878, 2003.
Cette référence a été citée à la page 163.

[66] Gerard Z. HERTZ et Gary D. STORMO.
A Large-Deviation Statistical Basis for Penalizing Gaps.
Dans *Proceedings of the 3rd International Conference on Bioinformatics and Genome Research*, Identification of Consensus Patterns in Unaligned DNA and Protein Sequences, pages 201–216. World Scientific Publishing Co., Ltd., Singapore, 1995.
Cette référence a été citée aux pages 39, 41.

[67] Gerard Z. HERTZ et Gary D. STORMO.
Identifying DNA and Protein Patterns with Statistically Significant Alignments of Multiples Sequences.

Bioinformatics, 15(7–8):563–577, 1999.
Cette référence a été citée aux pages 77, 83.

[68] International Union of Pure and Applied Chemistry and International Union of Biochemistry and Molecular Biology.
IUPAC IUBMB Joint Commission on Biochemical Nomenclature.
Disponible à l'adresse
http://www.chem.qmul.ac.uk/iubmb/.
Cette référence a été citée aux pages 18, 240.

[69] IUPAC IUBMB Joint Commission on Biochemical Nomenclature (JCBN).
Abbreviations and Symbols for Nucleic Acids, Polynucleotides and their Constituants, 1970.
Recommandations. Voir [68].
Cette référence a été citée aux pages 18, 185, 249.

[70] IUPAC IUBMB Joint Commission on Biochemical Nomenclature (JCBN).
Abbreviations and Symbols for the Description of Conformations of Polynucleotide chains, 1983.
Recommandations. Voir [68].
Cette référence a été citée aux pages 18, 187, 249.

[71] Inge JONASSEN.
Efficient discovery of conserved patterns using a pattern graph.
*Computer Applications in the Biosciences (*CABIOS*)*, 13:509–522, 1997.
Cette référence a été citée aux pages 33, 39, 40, 52, 83.

[72] Inge JONASSEN, Carsten HELGESEN et Desmond HIGGINS.
Scoring function for pattern discovery programs taking into account sequence diversity.
Rapport technique 116, Department of Informatics, University of Bergen, Norway, 1996.
Cette référence a été citée aux pages 31, 83.

[73] David T. JONES, William R. TAYLOR et Janet M. THORNTON.
The rapid generation of mutation data matrices from protein sequences.
*Computer Applications in the Biosciences (*CABIOS*)*, 8(3):275–282, 1992.
Cette référence a été citée à la page 32.

[74] Walter JONES.
NUCLEIC ACIDS, *Their Chemical Properties and Physiological Conduct*.
Monographs in Biochemistry. Longmans, Green & co, New York, deuxième édition, 1920.
Cette référence a été citée aux pages 3, 11.

[75] Samuel KARLIN et Stephen F. ALTSCHUL.
Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes.
Dans *Proceedings of National Academy of Science (*PNAS*)*, volume 87 de *6*, pages 2264–2268, 1990.
Cette référence a été citée à la page 83.

[76] Richard M. KARP, Raymond E. MILLER et Arnold L. ROSENBERG.
Rapid identification of repeated patterns in strings, trees and arrays.
Dans ACM, réd., *Proceedings of the 4th Annual* ACM *Symposium on Theory of Computing*, pages 125–136, New York, USA, 1972. ACM Press.
Cette référence a été citée à la page 36.

[77] Donald Ervin KNUTH.
Seminumerical Algorithms, volume 2 de *The Art of Computer Programming*.

Addison-Wesley Longman Publishing Co., Inc., Boston, USA, troisième édition, 1997.
Cette référence a été citée à la page 71.

[78] Ian KORF, Mark YANDELL et Joseph BEDELL.
BLAST.
O'Reilly & Associates, Inc., Sebastopol, USA, 2003.
Cette référence a été citée à la page 198.

[79] Timo KOSKI.
Hidden Markov Models for Bioinformatics, volume 2 de *Computational Biology*.
Springer-Verlag, première édition, 2002.
Cette référence a été citée à la page 41.

[80] Albrecht KOSSEL.
Ueber die basischen Stoffe des Zellkerns.
Hoppe-Seyler's Zeitschrift für Physiologische Chemie, 22:176–187, 1896.
Cette référence a été citée aux pages 3, 11.

[81] Charles E. LAWRENCE, Stephen F. ALTSCHUL, Mark S. BOGUSKI, Jun S. LIU, Androw F. NEUWALD et John C. WOOTTON.
Detecting subtle sequence signal: a Gibbs sampling strategy for multiple alignment.
Science, 262(5631):208–214, 1993.
Cette référence a été citée aux pages 39, 41, 52, 77.

[82] Thierry LECROQ.
A Variation on the Boyer-Moore Algorithm.
*Theoretical Computer Science (*TCS*)*, 92(1):119–144, 1992.
Cette référence a été citée à la page 35.

[83] Lillian LEE.
Measures of Distributional Similarity.
Dans *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32. Association for Computational Linguistics, 1999.
Cette référence a été citée à la page 30.

[84] Arnaud LEFEBVRE et Thierry LECROQ.
Compror: on-line lossless data compression with a factor oracle.
Information Processing Letters, 83(1):1–6, 2002.
Cette référence a été citée à la page 163.

[85] Arnaud LEFEBVRE, Thierry LECROQ, Hélène DAUCHEL et Joël ALEXANDRE.
FORRepeats: detects repeats on entire chromosomes and between genomes.
Bioinformatics, 19(3):319–326, 2003.
Cette référence a été citée à la page 163.

[86] Pierre LEGENDRE et Louis LEGENDRE.
Numerical Ecology.
Elsevier Sciences BV, Amsterdam, The Netherlands, deuxième édition, 1998.
Cette référence a été citée à la page 25.

[87] Vladimir I. LEVENSHTEIN.
Binary codes capable of correcting deletions, insertions, and reversals.
Doklady 4, Akademii Nauk SSSR, 1965.
Cette référence a été citée à la page 242.

[88] Vladimir I. LEVENSHTEIN.

Binary codes capable of correcting deletions, insertions, and reversals.
Cybernetics and Control Theory, 10(8):707–710, 1966.
Version anglaise de [87].
Cette référence a été citée à la page 27.

[89] Carl von LINNÉ (Carolinus LINNÆUS).
Systema Naturæ, sive, Regna tria Naturæ systematice proposita per classes, ordines, genera & species.
Lugduni Batavorum, première édition, 1735.
cf. [90].
Cette référence a été citée à la page 16.

[90] Carl von LINNÉ (Carolinus LINNÆUS).
Systema naturæ per Regna tria Naturæ, secundum classes, ordines, genera, species, cum characteribus, differentiis, synonymis, locis.
Lugduni Batavorum, douzième édition, 1766.
Cette référence a été citée à la page 242.

[91] David J. LIPMAN et William R. PEARSON.
Rapid and Sensitive Protein Similarity Search.
Science, 227(4693):1435–1441, 1985.
Cette référence a été citée aux pages 62, 67, 83.

[92] Brenda MADDOX.
Rosalind FRANKLIN*: The Dark Lady of* DNA.
HarperCollins, 2002.
Cette référence a été citée à la page 3.

[93] Udi MANBER et Gene MYERS.
Suffix Arrays: a new method for on-line string searches.
SIAM *Journal on Computing*, 22(5):935–948, 1993.
Cette référence a été citée à la page 38.

[94] Alban MANCHERON, Jérémie BOURDON et Irena RUSU.
Pattern Extraction Algorithm using a Gaussian Scoring Scheme.
En cours de rédaction, 2006.
Cette référence a été citée à la page 83.

[95] Alban MANCHERON et Christophe MOAN.
Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles.
Dans *Proceedings of the 4th Prague Stringology Conference (*PSC*)*, pages 139–154, 2004.
Disponible à l'adresse
http://www.stringology.org/event/2004/p12.html.
Cette référence a été citée à la page 242.

[96] Alban MANCHERON et Christophe MOAN.
Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles.
*International Journal on Foundations of Computer Science (*IJFCS*)*, 16(6):1179–1191, 2005.
Version journal de [95].
Cette référence a été citée à la page 141.

[97] Alban MANCHERON et Irena RUSU.
Pattern discovery allowing gaps, substitution matrices and multiple score functions.

Dans Gary BENSON et Roderic PAGE, réds., *Algorithms in Bioinformatics. Proceedings of the 3rd International Workshop on Algorithms in BioInformatics (*WABI*)*, volume 2812 de *Lecture Notes in Bioinformatics (*LNBI*)*, pages 129–145. Springer-Verlag, 2003.
Cette référence a été citée aux pages 45, 57, 109.

[98] Laurent MARSAN et Marie-France SAGOT.
Extracting structured motifs using a suffix tree – algorithms and application to promoter consensus identification.
Dans *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (*RECOMB*)*, pages 210–219, New York, USA, 2000. ACM Press.
Cette référence a été citée à la page 77.

[99] Laurent MARSAN et Marie-France SAGOT.
Extracting Structured Motifs Using a Suffix Tree – Algorithms and Application to Promoter Consensus Identification.
Journal of Computational Biology, 7:345–360, 2001.
Cette référence a été citée aux pages 39, 40.

[100] Johann Friedrich MIESCHER.
Ueber die chemische Zusammensetzung der Eiterzellen.
Hoppe-Seyler's medicinisch-chemische Untersuchungen, 4:441–460, 1871.
Cette référence a été citée à la page 3.

[101] Donald R. MORRISON.
PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric.
Journal of the ACM, 15(4):514–534, 1968.
Cette référence a été citée à la page 141.

[102] Gonzalo NAVARRO.
A Guided Tour to Approximate String Matching.
ACM *Computing Surveys*, 33(1):31–88, 2001.
Cette référence a été citée à la page 35.

[103] Gonzalo NAVARRO et Kimmo FREDRIKSSON.
Average complexity of exact and approximate multiple string matching.
*Theoretical Computer Science (*TCS*)*, 321:283–290, 2004.
Cette référence a été citée aux pages 35, 36.

[104] Saul B. NEEDLEMAN et Christian D. WUNSCH.
A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins.
Journal of Molecular Biology, 48:443–453, 1970.
Cette référence a été citée à la page 27.

[105] Syozo OSAWA, Thomas JUKES, Kunio WATNABE et Akira MUTO.
Recent evidence for evolution of the genetic code.
Microbiology Reviews, 56(1):229–264, 1992.
Cette référence a été citée à la page 186.

[106] William R. PEARSON et David J. LIPMAN.
Improved tools for biological sequences comparison.
Dans *Proceedings of National Academy of Science (*PNAS*)*, volume 85, pages 2444–2448, 1988.
Cette référence a été citée aux pages 67, 83.

[107] Anders Gorm PEDERSEN, Pierre BALDI, Yves CHAUVIN et Søren BRUNAK.

The Biology of Eukaryotic Promoter Prediction – a Review.
Computers & Chemistry, 23(3–4):191–207, 1999.
Cette référence a été citée à la page 5.

[108] Pavel A. PEVZNER et Sing-Hoi SZE.
Combinatorial Approaches to Finding Subtle Signals in DNA Sequences.
Dans *Proceedings of the 8th International Conference on Intelligent System of Molecular Biology (ISMB)*, pages 269–278. AAAI Press, 2000.
Cette référence a été citée aux pages 39, 41, 47.

[109] Mathieu RAFFINOT.
On the Multi Backward DAWG Matching algorithm (MultiBDM).
Dans Roberto BAEZA-YATES, réd., *Proceedings of the 4th South American Workshop on String Processing*, pages 149–165, Valparaíso, Chile, 1997. Carleton University Press.
Cette référence a été citée à la page 36.

[110] Christophe RAPINE.
Théorie des Graphes.
Disponible à l'adresse
http://gilco.inpg.fr/rapine/, et également à l'adresse
http://gilco.inpg.fr/rapine/Graphe/default.html.
Cette référence a été citée à la page 205.

[111] Benjamin RENARD.
Probabilités et statistiques appliquées à l'hydrologie, 2004.
Disponible à l'adresse
http://cvbr.ifrance.com/CoursMasterLyon.pdf.
Cette référence a été citée à la page 216.

[112] Isidore RIGOUTSOS et Aris FLORATOS.
Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm.
Bioinformatics, 14(1):55–67, 1998.
Cette référence a été citée aux pages 39, 41, 77.

[113] Isidore RIGOUTSOS, Aris FLORATOS, Laxmi PARIDA, Yuan GAO et Daniel PLATT.
The Emergence of Pattern Discovery Techniques in Computational Biology.
Metabolic Engineering, 2(3):159–177, 2000.
Cette référence a été citée à la page 37.

[114] Keith ROBISON, Abigail Manson MCGUIRE et George M. CHURCH.
A Comprehensive Library of DNA-binding Site Matrices for 55 Proteins Applied to the Complete *Escherichia coli* K-12 Genome.
Journal of Molecular Biology, 284(2):241–254, 1998.
Cette référence a été citée aux pages 5, 76, 135.

[115] Eduardo Pimentel Cachapuz ROCHA.
Analyse exploratoire des génomes bactériens.
Thèse de Doctorat, Université de Versailles Saint-Quentin-en-Yvelines, France, 2000.
Cette référence a été citée à la page 31.

[116] Robert J. ROTHBAUM, John C. PARTIN, K. SAALFIELD et A. James MCADAMS.
An ultrastructural study of enteropathogenic Escherichia coli infection in human infants.
Ultrastructural Pathology, 4(4):291–304, 1983.
Cette référence a été citée à la page 17.

[117] Mikhail A. ROYTBERG.
A search for common patterns in many sequences.
*Computer Applications in the Biosciences (*CABIOS*)*, 8(1):57–64, 1992.
Cette référence a été citée aux pages 37, 58.

[118] Thomas D. SCHNEIDER et R. Michael STEPHENS.
Sequence Logos: A New Way to Display Consensus Sequences.
Nucleic Acids Research, 18:6097–6100, 1990.
Cette référence a été citée à la page 29.

[119] Thomas D. SCHNEIDER, Gary D. STORMO, Larry GOLD et Andrzej EHRENFEUCHT.
The Information Content of Binding Sites on Nucleotide Sequences.
Journal of Molecular Biology, 188:415–431, 1986.
Cette référence a été citée à la page 83.

[120] Mark C. SHANER, Ian M. BLAIR et Thomas D. SCHNEIDER.
Sequence Logos: A Powerful, Yet Simple, Tool.
Dans T. N. MUDGE, V. MILUTINOVIC et L. HUNTER, réds., *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, volume 1 de *Architecture and Biotechnology Computing*, pages 813–821. IEEE Computer Society Press, 1993.
Disponible à l'adresse
http://www.lecb.ncifcrf.gov/~toms/paper/hawaii/.
Cette référence a été citée à la page 29.

[121] Claude E. SHANNON.
A Mathematical Theory of Communication.
The Bell System Technical Journal, 27:379–423, 623–656, 1948.
Cette référence a été citée aux pages 27, 28, 33, 83, 198.

[122] Hamilton SMITH, Thomas ANNAU et Srinivasan CHANDRASEGARAN.
Finding Sequence Motifs in Groups of Functionally Related Proteins.
Dans *Proceedings of National Academy of Science (*PNAS*)*, volume 87, pages 826–830, 1990.
Cette référence a été citée à la page 39.

[123] Temple F. SMITH et Michael S. WATERMAN.
Identification of Common Molecular Subsequences.
Journal of Molecular Biology, 147:195–197, 1981.
Cette référence a été citée à la page 36.

[124] Gayathri SRINIVASAN, Carey M. JAMES et Joseph A. KRZYCKI.
Pyrrolysine encoded by UAG in Archaea: charging of a UAG-decoding specialized tRNA.
Science, 296(5572):1459–1462, 2002.
Cette référence a été citée à la page 187.

[125] Wojciech SZPANKOWSKI.
Average Case Analysis of Algorithms on Sequences.
John Wiley & Sons, Inc., New York, USA, 2001.
Cette référence a été citée à la page 109.

[126] Pang-Ning TAN, Vipin KUMAR et Jaideep SRIVASTAVA.
Selecting the Right Objective Measure for Association Analysis.
Information Systems, 29(4):293–313, 2004.
Cette référence a été citée à la page 170.

[127] Esko UKKONEN.

On-line construction of suffix trees.
Algorithmica, 14(3):249–260, 1995.
Cette référence a été citée aux pages 40, 142.

[128] Alain-Jacques VALLERON.
Introduction à la biostatistique.
Collection Evaluation et Statistique. Masson, Paris, 1998.
Cette référence a été citée aux pages 74, 133, 170.

[129] Jason Tsong-Li WANG, Thomas G. MARR, Dennis SHASHA, Bruce A. SHAPIRO et Gung-Wei CHIRN.
Discovering Active Motifs in Sets of Related Protein Sequences and Using Them for Classification.
Nucleic Acids Research, 22(14):2769–2775, 1994.
Cette référence a été citée aux pages 39, 40.

[130] Bruce W. WATSON.
A Taxonomy of Finite Automata Construction Algorithms.
Rapport technique Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993.
Cette référence a été citée à la page 207.

[131] Bruce W. WATSON.
A Taxonomy of Finite Automata Minimization Algorithms.
Rapport technique Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993.
Cette référence a été citée à la page 207.

[132] James Dewey WATSON et Francis Harry Compton CRICK.
Genetical implications of the structure of deoxyribonucleic acid.
Nature, 171(4361):964–967, 1953.
Cette référence a été citée à la page 3.

[133] James Dewey WATSON et Francis Harry Compton CRICK.
Molecular structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid.
Nature, 171(4356):737–738, 1953.
Cette référence a été citée aux pages 3, 4, 12.

[134] La petite histoire du café ☕.
Disponible à l'adresse
http://www.southseatrade.com/, et également à l'adresse
http://www.southseatrade.com/histoire.html.
Cette référence a été citée à la page VIII.

[135] Wikipedia.
GNU Free Documentation License.
Disponible à l'adresse
http://fr.wikipedia.org/wiki/Accueil.
Cette référence a été citée aux pages 3, 84, 95.

[136] Peter WEINER.
Linear Pattern Matching Algorithms.
Dans *Proceedings of the 14th Annual Sympoium on Foundations of Computer Science (*FOCS*)*, pages 1–11, 1973.

Cette référence a été citée à la page 142.

[137] W. John WILBUR et David J. LIPMAN.
Rapid similarity searches of nucleic acid and protein data banks.
Dans *Proceedings of National Academy of Science (*PNAS*)*, volume 80, pages 726–730, 1983.
Cette référence a été citée aux pages 67, 83.

[138] Maurice Hugh Frederick WILKINS, Alec STOKES et Herbert WILSON.
Molecular Structure of Desoxypentose Nucleic Acids.
Nature, 171(4356):738–740, 1953.
Cette référence a été citée à la page 3.

[139] George Udny YULE.
On the Association of Attributes in Statistics: with Illustrations from the Material of the Childhood Society.
Philosophical Transactions of the Royal Society of London, 194(A):257–319, 1900.
Cette référence a été citée à la page 170.

Liste des tableaux

— *Corps du document* —

— *Annexes* —

Liste des figures

— *Pages liminaires* —

— *Corps du document* —

— *Annexes* —

Crédit photo

La reproduction de la *Tabula Smaragdina* (Table d'émeraude – page III) fait partie du domaine public. Elle est disponible à l'adresse :
http://www.la-rose-bleue.org/Etudes/Table_Emeraude_introduction.html.

Les photos de Rosalind Elsie FRANKLIN, Maurice Hugh Frederick WILKINS, James Dewey WATSON et Francis Harry Compton CRICK – pages 3 à 4 – sont la propriété du *Genetic Science Learning Center*, Université de l'Utah. Les images sont disponibles à l'adresse :
http://learn.genetics.utah.edu/features/dnaday/.

La figure 1.2 – page 12 – est inspirée du fichier
http://www3.sympatico.ca/philipe.lampron/nucleus1.gif.
L'auteur de cette figure (Philipe LAMPRON) m'ayant aimablement autorisé à la modifier.

La figure 1.3 – page 13 – est inspirée du fichier
http://arnica.csustan.edu/bioltd/images/protein_synthesis.jpg.
Cette image fait partie du domaine public.

La photo de Carl VON LINNÉ – page 16 – fait partie du domaine public. Elle est disponible à l'adresse :
http://www.efn.uncor.edu/dep/divbioeco/DivAni1/docencia/postgrado.htm.

La photo d'*Escherichia coli* – page 17 – fait partie du domaine public. Elle est disponible à l'adresse :
http://www.lbl.gov/Publications/Currents/Archive/Mar-05-2004.html.

La photo de *Saccharomyces cerevisiæ* – page 17 – est la propriété de *American Society for Microbiology*, qui m'a accordé l'autorisation de la reproduire dans ce manuscrit. Elle a été publiée dans [63] et est disponible à l'adresse :
http://www.microbeworld.org/htm/aboutmicro/gallery/gallery_06_sacc.htm.

Liste des algorithmes

— Corps du document —

— Annexes —

Liste des exemples

— *Corps du document* —

— *Annexes* —

Table des matières

— Corps du document —

Nomenclature & Biographies

- Notations -

$\alpha =_E \beta$ **:** cette égalité est vérifiée si et seulement s'il existe un ensemble X dans E tel que $\alpha, \beta \in X$ (si E n'est pas un ensemble d'ensembles, alors chaque élément de E est considéré comme un singleton).

$\alpha \neq_E \beta$ **:** cette inégalité est vérifiée si et seulement s'il n'existe pas d'ensemble X dans E tel que $\alpha, \beta \in X$ (si E n'est pas un ensemble d'ensembles, alors chaque élément de E est considéré comme un singleton).

$|\mathcal{E}|$, $|s|$ **:** ces notations représentent la cardinalité d'un ensemble ($|\mathcal{E}|$) ou la longueur d'une chaîne ($|s|$).

$f = O(g)$ **:** la fonction $f(n)$ est dans $O(g(n))$ (noté $f(n) = O(g(n))$) si et seulement s'il existe une constante $c > 0$ et $N \in \mathbb{R}$ tel que $\forall n \geq N, f(n) \leq c\, g(n)$.

$f = \Omega(g)$ **:** la fonction $f(n)$ est dans $\Omega(g(n))$ (noté $f(n) = \Omega(g(n))$) si et seulement s'il existe une constante $c > 0$ et $N \in \mathbb{R}$ tel que $\forall n \geq N, f(n) \geq c\, g(n)$.

$f = \Theta(g)$ **:** la fonction $f(n)$ est dans $\Theta(g(n))$ (noté $f(n) = \Theta(g(n))$) si et seulement si $f(n) = O(g(n))$ et $g(n) = O(f(n))$. Il est à noter que $f(n)$ est dans $\Theta(g(n))$ si et seulement si $g(n)$ est dans $\Theta(f(n))$. En outre, $f(n)$ est dans $\Theta(g(n))$ si et seulement si $f(n) = \Omega(g(n))$ et si $f(n) = O(g(n))$.

$f = o(g)$ **:** la fonction $f(n)$ est dans $o(g(n))$ (noté $f(n) = o(g(n))$) si et seulement si $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.

$\boldsymbol{v}$ **:** notation vectorielle.

f **:** ce symbole est principalement utilisé pour identifier un facteur particulier d'un mot. Il est également parfois utilisé dans un tout autre contexte pour dénoter une fonction mathématique.

m, s, w **:** ces symboles sont essentiellement utilisés pour représenter respectivement un motif, une séquence et un mot (*cf.* Définition 2.2 – page 23).

u, v **:** ces symboles sont majoritairement utilisés lors de la décomposition d'un mot en facteurs.

i, j, k **:** ces symboles sont utilisées pour représenter des entiers.

ℓ, n, t **:** ces symboles sont utilisés pour représenter des entiers. Les symboles ℓ et n traduisent essentiellement une notion de longueur, tandis que t désigne usuellement une cardinalité.

$[\![P]\!]$ **:** la notation d'IVERSON vaut 1 si la propriété P est vérifiée, 0 sinon.

$\left\{ {n \atop k} \right\}$ **:** cette notation est celle reprise dans [54, pages 273 et suivantes]. Elle désigne les « nombres de STIRLING de deuxième espèce », *i.e.*, le nombre de partitions d'un ensemble de n éléments en k

sous-ensembles non vides. Par convention, $\left\{ {n \atop 0} \right\} = [\![n = 0]\!]$.

$\#(v)$ **:** notation du degré d'un sommet v d'un graphe donné (*cf.* Définition D.2 – page 205). Le degré d'un sommet v est tel que $\#(v) = \#_{in}(v) + \#_{out}(v)$.

$\#_{in}(v)$ **:** notation du degré entrant d'un sommet v d'un graphe donné. (*cf.* Définition D.2 – page 205).

$\#_{out}(v)$ **:** notation du degré sortant d'un sommet v d'un graphe donné. (*cf.* Définition D.2 – page 205).

- $\mathcal{A}$ -

ADN : Acide DésoxyriboNucléique.

ALTMANN (Richard) : né en 1852, sa thèse fut encadrée par Friedrich MIESCHER. Il est à l'origine du terme « acide nucléique ».

ARN : Acide RiboNucléique.

- $\mathcal{B}$ -

bit : il s'agit de la plus petite unité de stockage en informatique. Un bit peut coder deux états (l'état vrai ou l'état faux), d'où son nom qui est en fait l'acronyme de l'anglais BInary digIT.

- $\mathcal{C}$ -

CRICK (Francis Harry Compton) : né en 1916, prix NOBEL de médecine en 1962 pour ses travaux sur la structure et le rôle de l'ADN.

- $\mathcal{E}$ -

EM : désigne le problème de l'extraction de motifs commun à un ensemble de séquences (*cf.* Section 2.5.1 – page 37).

EM$_q$ **:** désigne le problème de l'extraction sous contrainte de quorum de motifs communs dans un ensemble de séquences (*cf.* Section 2.5.1 – page 37).

entropie : nom féminin du XIX$^{\text{ème}}$ siècle, emprunt de l'allemand *Entropie*, dérivé savant du grec *entropê*. L'entropie désigne l'« action de se retourner », d'où l'« action de se transformer », le « changement ». En physique, l'entropie est un grandeur, qui est une fonction d'état, caractérisant la tendance d'un système à évoluer spontanément vers un état d'équilibre, différent de l'état initial dans lequel il se trouvait. L'entropie s'exprime en joules par kelvin.

- $\mathcal{F}$ -

FEULGEN (Robert) : né en 1884, il est l'inventeur d'un procédé chimique permettant de mettre en évidence la présence ou l'absence d'ADN dans un produit. Ce test est appelé « réaction de FEULGEN ». En présence d'ADN, le test prend une coloration rouge fuschia.

FRANKLIN (Rosalind Elsie) : née en 1920, elle est à l'origine de la découverte de la structure en double hélice de l'ADN.

- $\mathcal{G}$ -

génome : ensemble des gènes d'un organisme.

- $\mathcal{I}$ -

incrémental : néologisme informatique issu du verbe incrémenter. Cet adjectif caractérise une opération sur un ensemble de données qui ne concerne que les données qui ont été ajoutées depuis la dernière manipulation (« Le Jargon Français », http://www.linux-france.org/prj/jargonf/).

IUB : International Union of Biochemistry and Molecular Biology.

IUPAC : International Union of Pure and Applied Chemistry.

IVERSON (Kenneth Eugene) : né en 1920, récompensé par le prix TURING en 1979 pour ses contributions aux notations mathématiques (voir $[\![P]\!]$) et sa théorie des langages de programmation. Il est notamment l'inventeur du langage de programmation APL.

- $\mathcal{J}$ -

JONES (Walter) : né en 1865, il fut l'élève d'Albrecht KOSSEL, et fut l'un des premier à synthétiser les connaissances relatives aux acides nucléiques.

- $\mathcal{K}$ -

KOSSEL (Ludwig Karl Martin Leonhard Albrecht) : né en 1853, il fut le premier à établir le rôle de l'ADN. Il découvrit également la structure chimique des constituants de l'ADN, à savoir l'adénine, la guanine, la cytosine et la thymine. Ses travaux lui valurent de recevoir le prix NOBEL de médecine en 1910.

- $\mathcal{L}$ -

liaison covalente : liaison chimique par mise en commun d'électrons.

Limite centrale (Théorème) : soit (X_n) une suite de variables aléatoires indépendantes et de même loi (moyenne μ, variance σ^2, fonction de répartition (deux fois dérivable) $P_X(x)$), alors la suite de variables aléatoires $Z_n := \dfrac{\sum X_i - \mu\, n}{\sigma\, \sqrt{n}}$ suit asymptotiquement une loi gaussienne centrée et réduite.

- $\mathcal{M}$ -

macromolécule : association covalente d'un très grand nombre de molécules d'une même famille.

MIESCHER (Johan Friedrich) : né en 1844, c'est dans le cadre de sa thèse qu'il découvrit les acides nucléiques en 1869 ; il appelle cette nouvelle substance nucléine. Ses travaux ne furent publiés qu'en 1871, le temps que son responsable, Felix HOPPE-SEYLER, soit convaincu du bien fondé de sa découverte.

- $\mathcal{N}$ -

NOBEL (Alfred) : né en 1833, il demande par voie testamentaire que soit crée une institution chargée de récompenser chaque année les personnes qui ont rendu de grands services à l'humanité, en précisant que la nationalité des savants primés ne doit pas jouer de rôle dans l'attribution du prix. La fondation NOBEL voit le jour en juin 1900.

NOBEL (prix) : prix décerné chaque année depuis 1901 par la fondation homonyme pour récompenser les grandes avancées dans les domaines de la physique, de la chimie, de la médecine, de la littérature, de l'économie et de la paix.

- $\mathcal{P}$ -

protéome : ensemble des protéines d'un organisme.

- $\mathcal{R}$ -

***Royal Statistical Society* :** fondée en 1834, cet institut a pour objectif de faire avancer les statistiques, d'organiser des rencontres et conférences et de promouvoir cette discipline auprès du grand public.

- $\mathcal{S}$ -

STARS : **S**TARS is a **T**ool for **A**nalysis & **R**esearch in **S**equences.

StatiSTARS : **Stati**STARS uses **S**tatistical **T**echniques for **A**nalysis & **R**esearch in **S**equences.

STIRLING (James) : né en mai 1692, il publia ses principaux travaux « *Methodus Differentialis* » en 1730. Ce livre traite notamment des séries infinies, de l'addition, de la somme, des puissances carrées, ... Un des principaux objectifs de cet ouvrage fût l'étude de méthodes permettant d'accélérer la convergence des séries. Voir également les nombres de STIRLING (notés $\left\{ {n \atop k} \right\}$).

succédané : adjectif utilisé en Médecine signifiant « que l'on peut substituer à d'autres », en parlant de médicaments qui ont les mêmes propriétés. Il s'emploie aussi comme nom masculin : « un bon succédané » ; « les succédanés ». Ce mot désigne plus généralement tout produit qui est dérivé d'un autre et qui peut au besoin en tenir lieu (« Dictionnaire de l'Académie française », huitième édition).

- $\mathcal{T}$ -

TURING (Alan Mathison) : né en 1912, il est considéré comme l'un des pères fondateurs de l'informatique moderne. Il est à l'origine de la formalisation du concept d'algorithme et de calculabilité qui ont profondément marqué cette discipline, avec la machine de TURING.

TURING (prix) : le prix TURING ou TURING *Award*, en hommage à Alan Mathison TURING (1912–1954), est attribué tous les ans depuis 1966 par l'*Association for Computing Machinery* (ACM) à une personne sélectionnée pour sa contribution de nature technique faite à la communauté informatique. Elle est parfois considérée comme étant l'équivalent du prix NOBEL de l'informatique.

- $\mathcal{W}$ -

WATSON (James Dewey) : né en 1928, prix prix NOBEL de médecine en 1962 pour ses travaux sur la structure et le rôle de l'ADN.

WebStars : Interface *Web* pour STARS, écrite en PERL distribuée sous licence GPL.
`http://www.sciences.univ-nantes.fr/lina/bioserv/WebStars/`.

WILKINS (Maurice Hugh Frederick) : né en 1916, prix NOBEL de médecine en 1962 pour ses travaux sur la structure et le rôle de l'ADN.

- $\mathcal{Y}$ -

YULE (George Udny) : né en 1871, il fut l'élève de Karl PEARSON. Ils ont ensuite travaillé ensemble à l'étude des statistiques. Il fut président de la *Royal Statistical Society*.

Index alphabétique

- $\mathcal{D}$ -

- $\mathcal{E}$ -

- $\mathcal{F}$ -

- $\mathcal{G}$ -

- $\mathcal{H}$ -

- $\mathcal{I}$ -

- $\mathcal{J}$ -

- $\mathcal{K}$ -

- $\mathcal{L}$ -

- $\mathcal{M}$ -

- $\mathcal{N}$ -

- $\mathcal{O}$ -

- $\mathcal{P}$ -

- $\mathcal{Q}$ -

- $\mathcal{R}$ -

- $\mathcal{S}$ -

- $\mathcal{T}$ -

- $\mathcal{U}$ -

- $\mathcal{V}$ -

- $\mathcal{W}$ -

- $\mathcal{Y}$ -

Extraction de Motifs Communs dans un Ensemble de Séquences.

Application à l'identification de sites de liaison aux protéines dans les séquences primaires d'ADN.

Alban MANCHERON

Résumé

L'extraction de motifs ayant une signification biologique, et notamment l'identification de *sites de régulation* de la synthèse protéique dans les séquences primaires d'ADN, est un des enjeux de la recherche en bioinformatique. Une anomalie dans cette régulation peut avoir de graves conséquences sur la santé d'un organisme. Aussi, l'extraction de ces sites permet de mieux comprendre le fonctionnement cellulaire et de soigner certaines pathologies.

Les difficultés posées par ce problème sont le manque d'informations sur les motifs à extraire, ainsi que le volume important des données à traiter. Deux algorithmes polynomiaux – l'un déterministe et l'autre probabiliste – permettant de le traiter ont été conçus. Dans ce contexte, nous avons introduit une nouvelle famille de fonctions de score et étudié leurs propriétés statistiques. Nous avons également caractérisé le langage reconnu par la structure d'index appelée « Oracle », et proposé une amélioration la rendant plus efficace.

Mots-clés : bioinformatique, algorithmique, statistiques, automates, extraction de motifs, structures d'index, application à l'ADN.

Pattern Extraction from a Set of Sequences.

An application to proteins binding sites identification in DNA primary sequences.

Abstract

The extraction of significant biological patterns, and in particular the identification of regulation sites of proteinic synthesis in DNA primary sequences, is one of the major issues today in bioinformatics. Indeed any anomaly in proteinic synthesis regulation has detrimental damages on the well-being of certain organisms. Extracting these sites enables to better understand cellular operation or even to remove or cure pathology.

What is problematic is the lack of information on patterns to be extracted, as well as the large volume of data to mine. In this dissertation, we introduce two polynomial algorithms – the first one is deterministic and the other one is probabilist – to address the issue of pattern extraction. We introduce a new family of score functions and we study their statistical properties. We characterize the language which is recognized by the index structure named "Oracle", and we modify this structure in order to make it more efficient.

Keywords: bioinformatics, algorithmics, statistics, automata, pattern extraction, index structures, DNA application.

Classification ACM

Catégories et descripteurs de sujets : E.1 [**Data**]: Data Structures; F.1 [**Theory of Computation**]: Computation by Abstract Devices; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; G.3 [**Mathematics of Computing**]: Probability and Statistics; H.1.1 [**Information Systems**]: Models and Principles—*Systems and Information Theory*; H.3.1 [**Information Systems**]: Information Storage and Retrieval—*Content Analysis and Indexing*; H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*; H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*; I.5 [**Computing Methodologies**]: Pattern Recognition.

Termes généraux : Algorithms, Measurement, Performance, Theory.

Discipline : INFORMATIQUE
Spécialité : INFORMATIQUE

Laboratoire : **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.