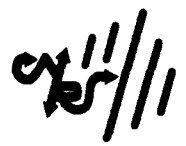
Laboratoire d'Informatique Fondamentale de Lille



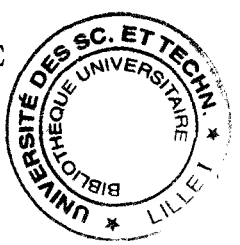
Numéro d'ordre: 1663

THÈSE

Nouveau Régime

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE



Pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Éric RIVALS

ALGORITHMES DE COMPRESSION ET APPLICATIONS À L'ANALYSE DE SÉQUENCES GÉNÉTIQUES

Thèse soutenue le 10 janvier 1996, devant la Commission d'Examen:

Président	:	Alain HÉNAUT	Université de Versailles-Saint-Quentin
Rapporteurs	:	Maxime CROCHEMORE	Université de Marne la Vallée
		Alain HÉNAUT	Université de Versailles-Saint-Quentin
Examinateurs	:	Didier ARQUÈS	Université de Marne la Vallée
		Max DAUCHET	Université de Lille I
		Jean-Paul DELAHAYE	Université de Lille I
		Christian GAUTIER	Université de Lyon I
		Mireille RÉGNIER	INRIA Rocquencourt

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Merci à ...

Max Dauchet et Jean-Paul Delahaye qui m'ont encadré depuis le DEA. Leur confiance, leur exigence et leurs idées sont autant de stimuli positifs. Ce travail leur doit beaucoup.

Maxime Crochemore et Alain Hénaut qui ont accepté de rapporter cette thèse, ainsi que pour leur œuvre au sein du GDR «Informatique et génomes» et du GREG.

Mireille Régnier, Didier Arquès et Christian Gautier qui ont aimablement accepté de composer mon jury.

Olivier Delgrange pour sa gentillesse et l'impulsion nouvelle qu'il a insufflée à notre équipe.

Marie-Odile Delorme, Emmanuelle Ollivier et Alain Hénaut pour leur patience dans nos discussions biologiques et la collaboration en cours.

Yves «tout sourire» André, Francis «big chief» Bossut, Cyrille «Walt Disney» d'Halluin et Marc «vive Emacs» Tommasi pour la géniale ambiance du bureau 316.

Anne Parrain, Denis Robilliard, David Simplot et Alain Preux pour leur soutien, leur dépannage en TeX, leurs discussions abondantes, ...

Tous les collègues du LIFL qui participent à une vie de laboratoire sympathique.

Mes parents et ma famille pour leur soutien de toujours.

Merci à Anne-Catherine.

Et pour qu'il y ait évolution, l'ensemble des hommes devra prendre conscience du fait que, si la consommation a pu constituer jusqu'ici le phénomène caractéristique de l'*homo faber*, la survie alimentaire de l'homme de demain étant assurée de façon stable par une thermodynamique économique fortement aidée par les machines, il restera alors la plus belle étape de l'évolution à parcourir, celle où l'homme n'agira plus pour consommer mais pour connaître.

Notre devoir d'homme est simple au fond, il consiste à oublier notre force pour utiliser notre imagination.

«Biologie et structure»
Henri Laborit

Table des matières

Chapitre 1

Introduction

Lancés il y a à peine dix ans, les programmes de séquençage de génomes complets progressent à grand pas. Pour la première fois, le séquençage de génomes complets d'organismes modèles tels que Escherichia coli ou Bacillus subtilis est en passe d'être achevé (E. coli et B. subtilis sont deux bactéries dont le génome est de l'ordre de 10^7 paires de bases). La quantité de paires de bases d'ADN séquencées qui s'amoncèle dans les bases de données, croît au rythme du doublement tous les 2 ans. Cependant, le séquençage n'est pas une fin en soi, seule l'analyse des séquences apporte de véritables connaissances biologiques. Si la localisation des gènes et la détermination de leur fonction sont les deux questions majeures pour les biologistes, bien d'autres rentrent dans le champ de l'analyse de séquences : existe-t-il une structure commune aux séquences des gènes ou à celles des chromosomes? Comment l'expression des gènes sous forme de protéines est-elle régulée et où sont les signaux correspondants dans la séquence? Quelles sont l'utilité et la forme des séquences non codantes qui séparent les gènes? etc. Toute une communauté de recherche s'est créée pour réunir biologistes et informaticiens autour de problèmes et d'objectifs communs, en particulier **la conception de méthodes d'analyse de séquences.**

La théorie de la complexité de Kolmogorov mesure la quantité d'informations contenues dans un objet par la longueur de sa plus courte description. Par définition, la complexité de Kolmogorov d'un objet est la longueur du plus petit programme capable de le générer sur un calculateur universel. La plus courte représentation d'un objet contient toutes les informations nécessaires à sa construction, c'est à dire les informations essentielles et seulement elles. Par conséquent, une représentation plus longue que sa complexité de Kolmogorov contient des redondances. La théorie de la complexité de Kolmogorov prouve que cette complexité n'est pas calculable en général, mais seulement approximable.

Les algorithmes de compression effectifs recherchent les redondances d'un objet pour les éliminer et ainsi diminuer la taille de sa représentation. Ses redondances confèrent à un objet de la régularité, régularité qui se traduit en une propriété. Un algorithme de compression exploite un type de régularités, une propriété particulière, pour trouver une description plus courte de l'objet. **Plus on compresse sa description, plus la propriété est pertinente pour l'objet en question.** Par exemple, dans un texte en français la lettre «e» est la plus courante. Cela donne aux textes la propriété d'avoir une distribution des symboles qui n'est pas une répartition au hasard, i.e. qui s'éloigne de l'équi-répartition. L'algorithme de Huffman exploite ce type de propriétés en associant un code court aux symboles courants et un code long aux symboles rares. Par cet algorithme, un texte est d'autant plus compressé que sa

distribution des symboles s'éloigne de l'équi-répartition. On peut dire que **compression est synonyme de compréhension**, puisqu'elle équivaut à détecter les propriétés d'un objet. En outre, un algorithme de compression mesure sa performance par un taux de compression, cela quantifie la propriété sur laquelle l'algorithme est basé (dans l'exemple, la propriété est : de combien la distribution des symboles diffère de l'équi-répartition, mais bien d'autres propriétés sont exploitables) et permet de comparer des objets entre eux selon cette propriété.

De cette situation est née l'idée d'utiliser les techniques de compression pour aider à l'analyse des séquences génomiques. De là est parti notre travail qui s'est attaché aux trois thèmes suivants :

- la recherche et l'exploitation de régularités réparties sur une longue séquence avec comme critère la compression ;
- la localisation de répétitions en tandem d'un petit motif pour la détection de zones de faible complexité, que les biologistes appellent le DOS-DNA ;
- l'élaboration d'une nouvelle approche construite sur la complexité de Kolmogorov et qui a pour but de définir une notion effective et générale de représentation optimale.

Les deux premiers thèmes sont très liés à la problématique biologique alors que le troisième concerne la théorie de l'information et les aspects conceptuels de la compression.

Partie I.

L'application de la compression à l'analyse de séquences génétiques requiert des méthodes capables de comprimer ces séquences. Une séquence est un texte sur l'alphabet des nucléotides {A,C,G,T}, on peut donc lui appliquer les méthodes classiques de compression de texte. Malgré les évidences biologiques de présence de régularités telles que les répétitions dans l'ADN, ces compresseurs ne parviennent pas à les comprimer et donnent des taux étonnamment bas (au maximum de quelques pour-cents). Partant de cette constatation, nous nous sommes fixé pour objectif, de concevoir pour les séquences génétiques, le compresseur le plus évolué possible qui n'encode que les répétitions. Cet objectif se justifie triplement :

- Si l'on sait qu'il existe des répétitions dans l'ADN, on ne sait pas dans quelle proportion, ni avec quelle répartition. Les méthodes biologiques expérimentales n'autorisent pas l'accès à des informations précises sur ces questions. Il est donc utile de créer un outil pour localiser des répétitions significatives dans une séquence.
- En outre, l'utilisation de cet outil permettra de savoir quel niveau de compression on peut généralement atteindre, uniquement grâce à des répétitions. Un intérêt de la méthode est qu'elle quantifie objectivement par le taux de compression, la présence de répétitions (plus généralement la présence de la régularité recherchée).
- Beaucoup de recherches génétiques se servent de ressemblances entre séquences pour émettre des hypothèses sur leur fonction ou sur leur conformation spatiale. Or, l'application de compresseurs à un ensemble de séquences permet des comparaisons globales et par là, des classifications de séquences.

Un compresseur qui encode des répétitions, substitue des portions répétées du texte par un code binaire. Il exécute deux fonctions :

la factorisation : la détection et le choix des portions «intéressantes» à coder,

l'encodage : la traduction en code binaire des portions choisies par la factorisation.

Une compression efficace passe par un outil puissant de détection des répétitions : nous montrons que l'arbre des suffixes est une structure de données performante et adaptée au problème (chapitre I.2). Un nœud interne de l'arbre des suffixes mémorise un facteur répété, nous les avons nommés : facteurs maximaux. En outre, l'arbre des suffixes est de construction linéaire par rapport à la taille du texte et permet de localiser facilement toutes les occurrences d'un facteur. Nous utilisons l'arbre des suffixes pour obtenir une liste de facteurs intéressants du point de vue de la compression (chapitre I.3). Après avoir passé en revue les méthodes existantes de compression utilisables pour les textes d'ADN (chapitre I.4), nous présentons notre algorithme «*Cfact*» (chapitre I.5). *Cfact* utilise la liste des facteurs maximaux obtenue de l'arbre des suffixes, dans sa procédure de factorisation qui choisit les facteurs selon les deux principes suivants :

- les facteurs sont examinés par ordre d'intérêt pour la compression, i.e. par ordre de longueur décroissante, et non dans l'ordre d'apparition dans le texte comme dans Biocompress, Lempel-Ziv 77 et 78 (cf. chapitre I.4 et [GT93a, ZL77, ZL78]) ;
- on s'assure que le gain potentiel d'une occurrence d'un facteur est positif : le nombre de bits potentiellement gagnés, si l'on substitue l'occurrence par un code binaire, doit être supérieur à 0. Aucune autre méthode ne fait cette vérification de manière aussi précise.

Nous démontrons un théorème qui établit une garantie de compression pour *Cfact*. Si une séquence contient au moins une «longue» répétition alors *Cfact* parvient à la compresser, et inversement, si *Cfact* obtient un taux de compression nul, cela prouve qu'il n'existe pas une seule répétition suffisamment «longue». La condition suffisante de compressibilité permet de calculer précisément le critère de longueur. Enfin, nous comparons les résultats de douze compresseurs sur des séquences d'ADN d'espèces différentes (chapitre I.6). On constate la difficulté de comprimer des séquences génétiques et l'inaptitude particulière des méthodes classiques à dictionnaire, i.e. les méthodes Lempel-Ziv 77, 78 et leurs dérivés. Ces tests révèlent aussi les aptitudes de *Cfact* à bien comprimer des séquences qui contiennent des répétitions. Nous concluons par quelques suggestions d'emploi des compresseurs pour l'annotation et la classification de séquences (le problème est abordé dans [Mil93]). Le schéma de compression de *Cfact*, s'adapte parfaitement à d'autres types de régularités algébriques, tels que les symétries dans les mots : ceci nous permet de traiter le cas des palindromes génétiques (chapitre I.5).

Partie II.

La seconde partie de notre travail est consacré à un autre algorithme de compression basé sur la présence de répétitions «en tandem», i.e. côte à côte, d'un motif. L'objectif biologique est d'identifier et de caractériser des segments considérés par les généticiens comme étant de faible complexité, ainsi que d'en étudier la répartition dans les chromosomes. L'intérêt pour ces zones est motivé par leur rôle dans certaines maladies génétiques humaines et par leur influence sur la conformation spatiale et la structure de l'ADN chromosomique.

Les répétitions approximatives en tandem d'un motif sont classées parmi ce type de segments de faible complexité. Une répétition est approximative si certaines de ses bases ont été modifiées. Deux exemples de séquences sont données dans la figure 1.1, pour les motifs *GA* et *ACG* ; dans la première un «C» a été inséré dans la répétition, dans la seconde une base fut supprimée, une autre mutée en «G» et un «T» inséré. Pour des motifs de longueur 1, 2 et 3, nous avons conçu et implanté respectivement 3 algorithmes capables d'identifier les répétitions en tandem approximatives de ces motifs. Nous les avons appliqués à des chromosomes de levure. Les résultats qui font l'objet d'une analyse approfondie avec une équipe de biologistes du Centre de Génétique Moléculaire, ont permis d'établir des conclusions biologiques intéressantes qui sont résumées ci-dessous. Cette partie met donc en pratique l'analyse de textes grâce à la compression, dont nous pensons qu'elle s'insère parmi l'ensemble des méthodes utilisées pour la biologie «in silico» (cf. [HD94, MJ93a]).

GAGAGACGAGAGAGA

ACGAGACGGCGACGACGTACG

FIG. 1.1 - *Exemple de répétitions en tandem approximatives pour les motifs GA et ACG.*

Nous détaillons la biologie des répétitions en tandem approximatives ou RTA, notamment les hypothétiques processus moléculaires capables de les engendrer (chapitre II.2). Ceci nous permet de préciser les interrogations biologiques auxquelles la méthode et les expériences tentent d'amener une réponse. Nous établissons la nécessité de concevoir une méthode propre au problème, encore du domaine de la recherche, de la détection de RTA (chapitre II.3). En effet, les algorithmes existants répondent partiellement au problème ([LS93, Sch94]) ou l'abordent avec d'autres critères de sélection de type statistique ([BW94]). Après une définition formelle des objets recherchés, la méthode est détaillée en commençant par le codage des répétitions et de leurs erreurs. Vient ensuite l'étude complète de l'algorithme de localisation des zones, on y montre particulièrement une spécificité de la méthodologie «compression» : comment elle prend en compte le critère du gain de compression pour sélectionner les RTA significatives sans obliger l'utilisateur à fixer un seuil arbitraire de significativité (chapitre II.3).

Puisque ce travail est autant une application concrète qu'une recherche pour la conception d'algorithmes, une grande attention est portée aux résultats (chapitre II.4). Nous expliquons précisément la procédure de test employée et donnons les résultats bruts : répartition et quantité de segments identifiés, motifs utilisés, etc, en prenant soin de souligner la capacité explicative et l'objectivité de la méthode. Sur les résultats et du point de vue méthodologique, notre approche est comparée à une analyse statistique de χ^2 faite sur ces mêmes chromosomes. Pour finir, nous discutons les conclusions élaborées à partir des observations biologiques que sont les résultats bruts. Trois conclusions semblent pertinentes aux biologistes moléculaires.

- De nombreux segments des séquences de chromosomes contiennent des RTA dont les motifs sont de 2 ou 1 lettres. Donc, l'apparition ou la présence de zones RTA n'est pas limitée aux triplets, comme dans les maladies humaines liées à l'expansion des RTA de motifs de trois nucléotides.

- Un mécanisme hypothétique de génération des RTA, dû au blocage de la réplication de l'ADN, impose une contrainte sur le motif qui est amplifié : celui-ci doit pouvoir s'auto-apparier chimiquement avec lui-même. Or parmi les motifs identifiés dans les chromosomes de levure, la grande majorité ne satisfont pas cette contrainte, invalidant l'hypothèse de l'emploi et même de l'existence de ce mécanisme chez la levure.
- Le phénomène le plus étonnant est que la présence de segments constitués de RTA semble uniforme sur les 4 chromosomes testés (plus de 2.3 millions de paires de bases). La disponibilité récente de séquences complètes de chromosomes a mis jusqu'à présent hors de portée expérimentale, ce type de caractéristiques globales de la structure du chromosome. L'uniformité de la présence de RTA apporte une des premières pierres à la caractérisation de l'objet biologique qu'est le chromosome.

Partie III.

Notre dernier travail s'intéresse à un aspect plus théorique de la compression : l'optimalité en taille de la représentation d'un objet. Cette question est naturelle : compresser un objet c'est trouver une représentation plus courte de cet objet. Cette représentation est-elle la meilleure possible? Est-ce bien celle qui permet la transmission la plus courte ou le stockage en mémoire le moins gourmand? D'un autre point de vue, la nouvelle représentation est elle-même un objet, on peut donc essayer de la comprimer et même itérer le processus au-delà. Quand doit-on arrêter l'itération?

Par exemple, considérons un émetteur qui doit transmettre des messages de la forme :

$$\underbrace{x \cdots x}_{m \text{ fois}} \qquad \text{notés} \quad x^m$$

où x est un texte et m un entier, tous deux quelconques. Il existe une infinité de messages de cette forme. Supposons que l'on ne sache rien sur la distribution des messages que la source fournit à l'émetteur. Sans informations statistiques sur la répartition des textes, celui-ci ne peut utiliser une méthode de compression statistique pour réduire le coût moyen de transmission d'un message. Pourtant, tous les messages ont une structure spéciale et semblent compressibles à cause de cela. Nous appelons une «hypothèse de structure» le fait que tous les messages adoptent une forme particulière. Si l'émetteur envoie pour chaque message le couple (x, m), il réalise une compression. Or, la théorie de la complexité de Kolmogorov nous apprend que parmi tous les couples (x, m) possibles, la plupart des items x et m sont incompressibles. L'émetteur réalise donc une compression optimale, puisque pour un couple quelconque, il ne peut pas transmettre moins d'informations que d'envoyer le couple lui-même, sinon x et m seraient compressibles en général. Notre travail propose une définition de représentation optimale en moyenne, qui permet de prouver que l'émetteur transmet en moyenne, la meilleure représentation pour les textes d'une forme donnée, c'est à dire de la forme x^m pour l'exemple. Cette définition est relative à une hypothèse de structure sur les textes à transmettre.

Notre approche propose donc une notion de représentation optimale en moyenne basée sur une hypothèse de structure, tout comme la théorie de l'information de Shannon ([SW49]), qui permet de calculer la taille d'une représentation optimale en moyenne, non plus d'après la forme des messages, mais étant donnée une distribution de probabilités des messages. Conceptuellement, la théorie de l'information de Shannon utilise la répartition des symboles

dans le texte, i.e. a une vision statistique du message, alors que notre approche en exploite des propriétés liées à l'ordre des symboles, une vision algorithmique du texte.

Nous avons vu qu'à la question de l'optimalité d'une représentation, la théorie algorithmique de l'information amène une réponse quasi-générale, puisque la complexité de Kolmogorov d'un objet est la longueur du plus petit programme capable de le générer. Ce plus petit programme est donc sa plus petite représentation. Malheureusement, cette représentation n'est pas calculable et donc cette théorie ne permet pas de savoir quand une représentation est la plus courte possible pour un objet quelconque.

Cependant, les objets que nous manipulons sur nos ordinateurs ne sont pas n'importe quels objets. Nous avons en général une idée de la forme qu'ils ont, et cette idée permet de caractériser leur structure. Par exemple, les programmes Pascal ont une structure contrainte par la syntaxe du langage. Les chansons constituent un tout autre exemple. Elles alternent souvent couplets et refrains, exploiter cette caractéristique structurale doit permettre d'obtenir en moyenne pour toutes les chansons une représentation comprimée, si on «factorise» le refrain. Cette partie du travail tente de formaliser cette intuition. Elle amène des réponses aux deux questions suivantes :

- Peut-on définir une notion d'optimalité en moyenne pour une représentation d'un objet qui tienne compte de caractéristiques structurales?
- Peut-on définir un concept d'optimalité en moyenne pour qu'il soit effectif, qu'il puisse aider à la conception d'un codage pour un compresseur et rende compte de nos intuitions légitimes?

Après avoir explicité plus longuement ces motivations, nous examinons comparativement d'autres approches pour la définition d'une optimalité ([GS91, LV93, SW49], chapitre III.2). Nous introduisons la notion de *forme* qui permet de donner une représentation pour des objets et d'exprimer qu'ils vérifient une hypothèse de structure. Une forme propose un codage ou représentation pour des objets, ainsi que le décompresseur associé (une forme est toujours décodable). Par exemple, pour les messages fournis à l'émetteur, la forme préconisée est la fonction :

$$\begin{array}{rccc} f: & \{0,1\}^* \times \mathbf{N} & \longrightarrow & \{0,1\}^* \\ & (x,m) & \longmapsto & x^m \end{array}$$

La représentation d'une forme est appropriée, si la longueur moyenne de toutes les représentations de taille n est égale, à un terme en $o(n)$ près, à la quantité d'informations moyenne de tous les objets correspondants : on dit alors que la forme correspondante est **K-optimale** (la quantité d'information est mesurée avec la complexité de Kolmogorov notée K). Dans notre exemple, f est K-optimale si et seulement si :

$$\underset{|t|=n}{Moy}\, K(f(t)) \quad = \quad n + o(n)$$

Par la suite, nous présentons plusieurs versions équivalentes de cette définition d'optimalité, montrant par là qu'elle est :

- naturelle et robuste puisque des changements dans la définition ne modifient pas la théorie ;
- en accord avec deux théories algorithmiques de l'information, qui sont similaires et néanmoins toutes les deux importantes : l'une contraint les programmes à être autodélimités et l'autre non.

Nous soulignons ensuite que notre théorie est plus générale que l'approche de [GS91], puisqu'elle n'exige pas qu'un seul code soit associé à chaque mot, i.e. que les formes soient injectives. En conséquence, une fonction qui compresse optimalement un langage d'après la définition de [GS91], correspond dans notre théorie à une forme nécessairement optimale.

Nous montrons que notre définition est utilisable en pratique et prouvons l'optimalité ou la non-optimalité pour des formes simples et naturelles (chapitre III.3). Nous abordons la question du gaspillage des codes pour les formes qui ne sont pas injectives. En effet, associer univoquement un code à un mot à encoder, est la façon la plus économique de concevoir un codage. Cependant, certains codages réalistes admettent plusieurs codes pour un même mot et gaspillent ainsi un certain nombre de codes. Nous examinons comment le gaspillage des codes influe sur l'optimalité d'une forme. Enfin, nous inspectons les possibilités de composition des formes (chapitre III.4). Nous prouvons plusieurs théorèmes qui permettent de décider de l'optimalité ou de la non-optimalité d'une forme complexe selon l'optimalité des formes plus simples qui la composent.

Chapitre 2

Généralités biologiques à l'usage de l'informaticien.

Les êtres vivants transmettent à leur descendance des caractéristiques dites héréditaires. Il existe un support moléculaire sur lequel ces informations sont inscrites : les **chaînes d'ADN** (Acide DésoxyriboNucléique). L'unité fondamentale de transmission héréditaire est le **gène** ; l'ensemble des gènes forme le **génome**.

Les êtres vivants se constituant par duplication de cellules, la totalité des informations héréditaires est conservée dans chaque cellule, sous la forme de **chromosomes**. Chez les **eucaryotes**, classe d'espèces à laquelle appartient l'homme, les chromosomes sont renfermés à l'intérieur du noyau. L'absence de noyau chez les **procaryotes**, classe d'espèces contenant les bactéries par exemple, implique que les chromosomes ne sont pas séparés des autres constituants de la cellule. Dans tous les cas, les chromosomes sont constitués de chaînes d'ADN enroulées, puis sur-enroulées pour être stockées dans un volume limité. Ces chaînes d'ADN ont les propriétés d'être :

- constituées d'une suite de **nucléotides** (groupement d'un acide, d'un phosphate et d'une base), dans laquelle seule la **base** varie suivant quatre possibilités {Adénine, Cytosine, Guanine, Thymine}, où les bases des couples (Adénine, Thymine) et (Guanine, Cytosine) peuvent s'apparier chimiquement parce qu'elles sont complémentaires selon les **appariements de Watson-Crick** ;
- orientées selon un sens de lecture dit de «5′ vers 3′» ;
- elles-mêmes appariées à une séquence inverse et complémentaire où les bases d'une chaîne sont liées aux bases de l'autre chaîne, cela constitue une molécule double-brin d'ADN faite par l'empilement de paires de bases[1] : la **double hélice d'ADN.**

Chaque gène est associé à une portion d'une chaîne d'ADN. **La fonction d'un gène est de «coder» pour une protéine** : c'est ainsi que le génome influence effectivement la construction de l'individu. En d'autres termes, les molécules dont le rôle est central dans le fonctionnement des êtres vivants, les protéines, ont leur plan décrit dans un gène. Ce lien est matérialisé par le processus **d'expression génique**, pendant lequel on construit la protéine d'après la séquence du gène. Il comporte trois étapes :

La transcription : La séquence du gène est copiée en une molécule simple-brin **d'ARN**

1. Dans la suite du document, l'unité de longueur des séquences, «paire de bases», est abrégée en pb

(Acide RiboNucléique) quasi-identique, si ce n'est qu'elle est écrite sur l'alphabet : {Adénine, Cytosine, Guanine, Uracyle}.

La maturation : Cette molécule subit diverses transformations dans sa séquence. Par exemple chez les eucaryotes, certains segments, les **introns**, sont excisés, et les portions restantes, les **exons**, sont concaténées pour produire l'ARN dit messager parce qu'il traverse la membrane nucléaire pour sortir du noyau.

La traduction : L'ARN final est lu par un ribosome qui construit **la séquence d'acides aminés qui forme la protéine**. À partir d'un signal dans la séquence du gène, le ribosome associe chaque groupe de trois nucléotides appelé **codon**, à un acide aminé. Le code de traduction employé universellement constitue le **code génétique**. Ce code est redondant puisqu'il y a 64 codons possibles et seulement 20 acides aminés.

Le génome est un texte dynamique : il est en perpétuelle évolution. L'hérédité semble contenir une part d'aléatoire. De nombreux mécanismes y participent, parmi les principaux :

- l'altération des séquences d'ADN par des **mutations ponctuelles** : substitution, suppression ou insertion d'une base.
- l'échange de longues portions similaires principalement entre les chromosomes par **recombinaison homologue** ou «crossing-over» durant la duplication cellulaire.
- la reproduction sexuée pour les eucaryotes, où chaque parent donne un stock de chromosomes pour constituer le génome dit diploïde de l'œuf fécondé.

Cette possibilité d'évolution se répercute en polymorphisme intraspécifique : les génomes de deux individus d'une même espèce diffèrent, en polymorphisme intercellulaire : les génomes de deux cellules d'un même organisme, qui proviennent pourtant du même gamète original, diffèrent eux aussi. La répercution la plus notable reste l'évolution des différentes espèces, dont on ne connaît que partiellement les parentés.

Première partie

Algorithmes de compression pour les séquences génétiques.

Chapitre 1

Introduction

Les séquences d'ADN et d'ARN sont des textes sur un alphabet de 4 lettres. La croissance exponentielle du nombre de séquences dans les bases de données et l'apparition récente de séquences de chromosomes complets créent un énorme travail d'analyse nécessaire à leur compréhension. L'ADN contient de nombreuses répétitions dont la fonction biologique est inconnue. Leur répartition n'est pas clairement établie et l'importance de leur proportion dans les séquences est imprécisément évaluée. En plus des méthodes d'alignement qui permettent d'identifier des similarités locales entre des séquences données, il existe un besoin pour des méthodes de comparaison globale des séquences. Il existe un besoin spécifique pour des outils de classification sur des critères tels que le contenu en répétitions ([HD94]). Des méthodes statistiques [KBS+93] proposent d'évaluer des séquences selon de tels critères. Nous proposons une méthode de type algorithmique basée sur schéma de compression adapté au critère du contenu en répétitions.

Notre travail est consacré à l'étude de la compression des séquences d'ADN. L'examen des résultats des méthodes usuelles et répandues de compression de texte, montre que celles-ci s'avèrent inopérantes parce qu'inadaptées (voir chapitre 6). Notre recherche est centrée sur la conception d'algorithmes de compression innovateurs et adaptés aux textes génétiques, pour améliorer les performances de compression. Celles-ci priment sur la robustesse du codage et le temps de calcul, la compression «en ligne» n'est pas requise dans les applications visées. Nos méthodes prennent en compte les caractéristiques des régularités des textes d'ADN, telles que la taille non bornée des répétitions et palindromes ou l'éloignement des occurrences d'une répétition. Notre effort se caractérise par l'utilisation d'une structure de données appropriée pour diverses recherches dans le texte, ainsi que par la gestion du dictionnaire pilotée par le gain de compression. Il s'appuie aussi sur la conception de codages auto-délimités qui évitent les limitations usuellement imposées aux dictionnaires et visent à tirer la meilleure compression des régularités étudiées. *La qualité originale de notre schéma de compression est de pouvoir garantir la compression de la séquence si elle contient des régularités.*

Une compression efficace passe par une structure de données performante pour la localisation de régularités dans le texte. Nous avons utilisé l'arbre des suffixes qui fait l'objet d'une présentation dans le chapitre 2. Ce chapitre aborde en premier lieu une structure de données plus simple, dont un aperçu facilite la compréhension de l'arbre des suffixes. Le chapitre suivant présente les algorithmes de localisation employés pour les répétitions directes et les palindromes génétiques. Il s'agit d'utilisations avantageuses de l'arbre des suffixes. Les généralités sur la compression de texte étant abordées dans la section 1.1, nous pouvons pré-

senter dans le chapitre 4, les méthodes classiques ainsi qu'une méthode dédiée aux séquences génétiques ([GT95]).

Notre apport dans le domaine de la compression de textes est détaillé dans le chapitre 5. Les deux méthodes de compression, une pour les répétitions directes nommée *Cfact*, l'autre pour les palindromes nommée *Cpal*, sont à classer dans les compresseurs à dictionnaire ou substitutionnels. Elles sont basées sur un schéma très similaire. Nous montrons qu'elles incluent une détection sophistiquée des régularités (répétitions ou palindromes) et une construction du dictionnaire adaptée au «langage» de l'ADN. Cette approche permet de prouver une garantie de compression utile pour l'application à la classification de séquences. Les résultats comparatifs résumés au chapitre 6 montrent la performance et l'application pratique du compresseur pour les répétitions directes.

1.1 Vision générale de la compression de textes.

1.1.1 Objectifs de la compression.

La compression est utile dans deux cadres très différents :

Pour le stockage et la transmission : la compression a pour objet de diminuer les coûts de la gestion de données par l'outil informatique, soit en réduisant l'occupation mémoire de fichiers stockés, soit en raccourcissant leur temps de transmission sur un réseau.

Pour l'analyse de textes : elle révèle des régularités dans la description d'un objet et permet de savoir si sa description est proche de la description optimale théorique de longueur égale à sa complexité de Kolmogorov (cf. [LV93]). Cette description optimale est le plus court programme capable de l'engendrer. Ainsi, si le codage des régularités diminue la taille du texte, la représentation se rapproche de la représentation optimale. Comme la théorie établit que seule la découverte de véritables régularités permet cela, une compression effective entraîne une meilleure compréhension de la structure du texte, parce qu'elle identifie des régularités significatives de ce texte.

Dans ce travail, nous utilisons la compression pour l'analyse de séquences génétiques. Ce genre d'analyse vise à comparer des textes entre eux. Un même algorithme peut engendrer une grande compression pour un texte et une petite pour un autre texte, simplement parce que la *quantité de régularités* identifiables par l'algorithme choisi est plus grande dans le premier texte que dans le second. Le *taux de compression* de chacun permet de comparer numériquement cette différence de quantité de régularités, **si et seulement si** l'algorithme compresse *sans perte d'informations*. Ceci signifie, qu'à partir de la version compressée du texte, l'algorithme de décompression reconstitue la version exacte du texte original. Tous les algorithmes présentés sont sans perte d'informations. Dans le domaine de la compression d'image (utilisée pour la réduction de transmission des images animées, par exemple) la compression avec perte d'informations est courante parce que la version approximée obtenue par décompression est suffisante pour la visualisation désirée.

1.1.2 Utilisation pour l'analyse de séquences génétiques.

Ce travail a été motivé par la volonté de créer des algorithmes pour compresser des séquences génétiques et plus particulièrement des séquences d'ADN sur un alphabet à quatre

lettres : {A, C, G, T}. En effet, les méthodes traditionnelles, de type statistique (codage de Huffman) comme de type algorithmique (codage à dictionnaire tel que Lempel-Ziv) ne parviennent pas à compresser ce type de textes (voir le chapitre 6). Plusieurs références font état de tests donnant des taux de compression faibles, de l'ordre de un pour cent de réduction de la taille : [GT93a, Riv94]. Notre objectif est donc de concevoir des algorithmes nouveaux qui codent des régularités, soient inhabituelles, soient envisagées de manière novatrice, dans le but d'atteindre un taux de compression meilleur que les méthodes habituelles. Dans ce cadre de recherche, nous considérons les objectifs usuels de performance en temps ou en espace de calcul, de résistance aux erreurs de transmission comme secondaires.

L'existence de nombreuses occurrences de répétitions et de palindromes génétiques dans les séquences d'ADN constitue un fait biologique. Certaines de ces occurrences peuvent atteindre des tailles de l'ordre de plusieurs milliers de paires de bases (kpb). Cependant, ces loci[1] de faible complexité ne sont pas nécessairement présents dans les séquences disponibles à l'heure actuelle, puisque celles-ci ont été séquencées pour leur contenu génique et non pour leur simplicité, utilité médicale oblige. Une autre caractéristique peut atténuer la possibilité de comprimer : les mutations dans les séquences qui rendent ce genre de régularités approximatives, donc plus difficiles à décrire ou à encoder. Il faut néanmoins concevoir les compresseurs pour des régularités exactes avant de les vouer à des régularités approximatives. Pour obtenir une amélioration sensible, il faut privilégier ces régularités qui représentent des gains potentiels très intéressants. D'où notre démarche vers les compresseurs algorithmiques ou à dictionnaire.

1.1.3 Compression «algorithmique» et langage d'instructions.

Nous présentons ici des compresseurs dits algorithmiques, où *algorithmique* doit être compris comme non *statistique*. Ces algorithmes détectent des propriétés locales de la structure du texte, et liées à l'ordre du texte, telles que :

- la répétition directe d'un facteur du texte, où directe signifie dans le sens de lecture,
- l'apparition d'un facteur qui est le palindrome génétique d'un autre facteur,
- la répétition en puissance d'un facteur, appelé aussi répétition en tandem, i.e. de la forme u^k où k est un entier et u un facteur du texte.

Les segments de structure plus régulière que le reste de la séquence, sont remplacés par un codage binaire. Régulière signifie que l'on peut encoder avantageusement, i.e. telle que le code de remplacement soit plus court que la description originale et engendre une compression. Le codage en binaire du segment représente une suite d'instructions qui permet de reconstituer la séquence originale. Par exemple, dans le cas où une portion est la répétition d'une autre portion de la séquence, l'instruction est : **copier la portion débutant à la position** 10450 **et de longueur** 211. Cette instruction de copie prend deux arguments : la position de début de la portion à copier et sa longueur. Bien sûr, on peut écrire cette instruction de différentes manières. Une autre façon est : **copier la portion allant de la position** 10450 **à** 10660. Ces différents formats ont le même sens et contiennent sensiblement la même quantité d'informations (on choisit le premier format qui est légèrement plus économique).

1. Locus désigne un segment contigu dans une séquence.

Le langage des instructions est fixé lors de la conception de l'algorithme de compression et dépend des régularités envisagées par l'algorithme. **Comme pour un langage de programmation, moins le langage contient d'instructions, i.e. moins il traite de régularités différentes, plus le code binaire des instructions est court.** Les bons algorithmes sont le résultat d'un compromis : le langage doit comprendre suffisamment d'instructions, mais ne pas en contenir trop.

1.2 Définitions et notations préliminaires pour l'algorithmique des mots.

1.2.1 Définitions : alphabet, mot, facteur, préfixe, suffixe.

Nous appelons un alphabet fini A, un ensemble fini de lettres (aussi appelées caractères ou symboles). Un *mot* ou un *texte* sur A désigne une suite de lettres de A. Soit t un texte, la *longueur* de t, notée $|t|$, est le nombre de lettres qui le composent. Le *mot vide* sur A est noté ε. Un mot peut être vu comme un tableau de caractères indicés de 1 à $|t|$, ainsi la lettre d'indice k de t est notée $t[k]$. On appelle *facteur* de t, un mot composé d'une sous-suite contiguë de lettres de t. Un facteur f de t est entièrement défini par un couple (i, j) où i est sa position de début dans t et j sa position de fin, il est noté $t[i..j]$ et $|f| = j - i + 1$. Un *préfixe* de t est un facteur de t qui débute à l'indice 1. Soit $i \leq |t|$, on note t^i le préfixe de t de longueur i, i.e. le facteur $t[1..i]$. Un *suffixe* de t est un facteur de t qui se termine à l'indice $|t|$. Soit $j \leq |t|$, on note t_j le suffixe de t qui débute à l'indice j et est de longueur $|t| - j + 1$, i.e. le facteur $t[j..|t|]$. En outre, on note par $Fact(t)$ l'ensemble des facteurs d'un texte t.
Remarque : La notation t^i est employé dans cette partie pour désigner un préfixe de t et non une puissance de t.

Exemple 1 Soit $A = \{a, b, c\}$ l'alphabet et $t = abacbbcac$ un texte sur A. Alors, $t[4] = c$, $cbbc$ est un facteur de t et est noté $t[4..7]$, tandis que cba n'en est pas un ; $abacb$ est le préfixe de t de longueur 5 (noté t^5) et $cbbcac$ est le suffixe de t qui débute à la position 4 (noté t_4).

Notons que tout facteur d'un texte est le préfixe d'un seul suffixe de ce texte et aussi le suffixe d'un seul de ses préfixes.

1.2.2 Une symétrie particulière : les palindromes génétiques.

Les lettres des séquences d'ADN et d'ARN, qui représentent des nucléotides, ont la propriété de pouvoir s'apparier chimiquement. Ces appariements canoniques découverts par Watson-Crick sont vus algébriquement comme une mise en relation des bases. Les couples possibles sont donnés par la relation de complémentarité.

Définition 1 La relation symétrique de *complémentarité* chimique entre les lettres, notée $Comp$ est définie par : $Comp(A) = T$ et $Comp(C) = G$.

Ces appariements chimiques forment la structure de la double hélice d'ADN, mais permettent aussi à une séquence non liée à une molécule complémentaire, de se replier pour apparier des facteurs complémentaires. Comme les molécules nucléotidiques sont orientées, deux mots qui autorisent cela, doivent être complémentaires et renversés, d'où la notion de *palindrome génétique.*

Définition 2 Soient $u, \overline{u}$ deux textes sur A. $(u, \overline{u})$ forment un palindrome génétique ssi :

$$\begin{cases} |u| = |\overline{u}| \\ \forall\ 1 \leq k \leq |u| \quad u[k] = Comp(\overline{u}[|u| - k + 1]) \end{cases}$$

La *taille du palindrome* est celle de u.

Exemple 2 Une séquence contenant un palindrome génétique :

ttcgtacccGTAACTGCGAacatggTCGCAGTTACcattg

Chapitre 2

L'arbre des suffixes.

Dans ce chapitre, nous considérons un texte t de longueur n sur l'alphabet A , un mot p sur A de longueur m. Étant donné un texte t, chercher les occurrences d'un mot p dans t coûte de l'ordre de $O(n)$ opérations élémentaires (cf. [BM77, CR94, KMP77]). Si l'on désire localiser les occurrences de k différents mots dans t (par exemple pour exécuter plusieurs recherches lors de la consultation d'un texte), la répétition de l'algorithme de recherche d'un mot devient très coûteuse, en $O(k.n)$. En outre, certaines recherches refont des comparaisons déjà effectuées par d'autres qui seraient inutiles si leurs résultats avaient été mémorisés. La construction d'un *arbre des suffixes* apporte une solution à ce problème. D'autres structures de données, le *graphe des sous-mots et l'automate des suffixes*, dont la construction est aussi un pré-traitement du texte t, sont aussi de *bonnes représentations* de tous les facteurs de t (terme employé dans [CR94]), i.e. :

1. ils sont de taille linéaire par rapport à n
2. leur construction est en temps linéaire par rapport à n
3. ils permettent la recherche d'un mot p en temps linéaire par rapport à $|p|$.

De telles structures de données facilitent l'accès à n'importe quel facteur du texte, comme l'index d'un livre pour les mots qu'il contient. Ainsi la recherche de k mots différents de taille inférieure à $|p|$ ne coûte au total que $O(k.|p|)$ au lieu de $O(k.n)$.

Dans ce chapitre, nous présentons la structure de données de l'arbre des suffixes pour la compréhension des utilisations que nous en faisons dans les algorithmes de compression. Nous détaillons d'abord la structure plus simple nommée *trie* (pas de traduction en français) qui est une étape intermédiaire vers l'arbre des suffixes. Il s'agit d'une structure de données plus simple et plus explicite, mais aussi moins économique.

2.1 La structure de données trie.

Dans un premier temps, nous présentons intuitivement la structure de donnée du trie d'un texte et en donnons un exemple. Dans un deuxième temps, nous écrivons formellement les propriétés de cette structure de manière à pouvoir s'y référer plus tard. Ces propriétés découlent naturellement de la structure du trie. La sous-section 2.1.2 est simplement une reformulation des propriétés présentées en sous-section 2.1.1. Cette organisation de la présentation est aussi employée pour l'arbre des suffixes dans la section 2.2.

La première sous-section présente aussi des conditions imposées à la construction du trie, mais qui s'appliquent aussi à l'arbre des suffixes. Nous énonçons ensuite les propriétés de représentation des facteurs du texte, après avoir introduit les définitions permettant de manipuler une structure d'arbre.

2.1.1 Exemple, définitions et conditions structurales.

La figure 2.1 montre le trie pour le texte $t = ababbbac$ sur l'alphabet $\{a, b, c\}$ noté A. On prend la précaution de placer comme dernier caractère de t, un marqueur de fin pour éviter que deux suffixes soient préfixe l'un de l'autre.

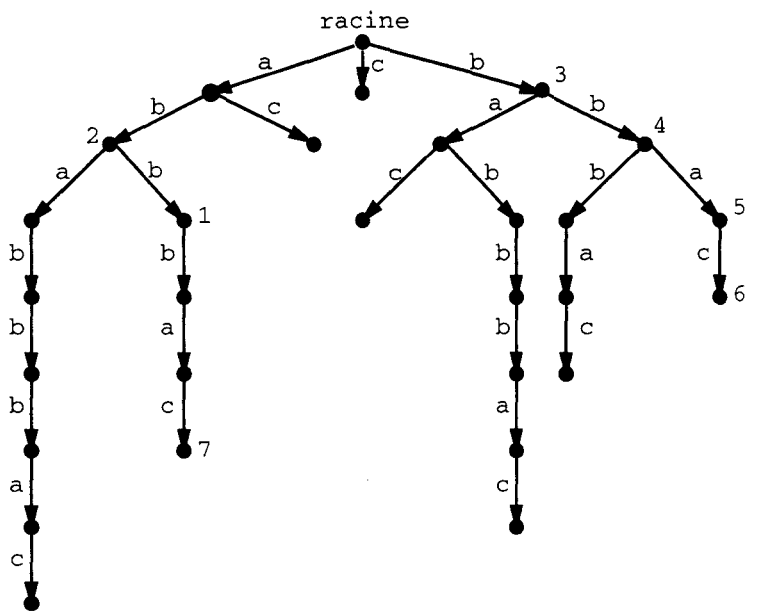


FIG. 2.1 - *Exemple de trie pour le texte $t = ababbbac$. Les nœuds numérotés sont utilisés dans le texte.*

Le trie d'un texte t, noté $trie(t)$, est un arbre dans lequel une branche est associée à chaque suffixe du texte. Les arcs orientés dans le sens père-fils sont étiquetés par une seule lettre. Un suffixe du texte est donc représenté par la suite des étiquettes le long de sa branche de la racine vers la feuille. Par économie, on impose qu'un nœud ait au plus un arc sortant par symbole de l'alphabet. Ainsi, toutes les branches passant par un nœud interne et ayant même étiquette après ce nœud, sont groupées sur le même arc sortant. Comme tout facteur est le préfixe d'un suffixe, tout chemin de la racine à un nœud représente de manière univoque un facteur de t. Le facteur est formé de la concaténation des étiquettes prises en descendant le chemin. On désigne par N l'ensemble des nœuds de $trie(t)$. Par exemple, le chemin de la racine au nœud 1 est étiquetté par le facteur abb. On confond le nœud avec le facteur qu'il représente. Voici un exemple explicite. Le trie de la figure 2.1 contient 8 feuilles, i.e. autant que de suffixes dans t. Tous les nœuds sur le chemin de la racine à la feuille 6 sont associés à un préfixe de $bbac$:

$fact(racine) = \varepsilon$ $fact(3) = b$ $fact(4) = bb$
$fact(5) = bba$ $fact(6) = bbac$

Nous donnons les définitions qui permettent d'écrire ces propriétés de manière formelle.

Définition 3 Soit e un arc reliant deux nœuds du trie, on note $label(e)$ l'étiquette de cet arc. Rappelons que dans $trie(t)$, quel que soit e, $|label(e)| = 1$. Par extension, l'étiquette d'un chemin α formé de la suite d'arcs $e_1 \cdots e_j$, noté $label(\alpha)$ est le mot formé de $label(e_1) \cdots label(e_j)$. Soit g, h deux nœuds. S'il existe un chemin entre g et h, il est unique car la structure de données que nous étudions est un arbre, on note ce chemin par $chemin(g, h)$. Pour tout nœud g, on définit $fact(g)$ le facteur univoquement associé à g par $fact(g) = label(chemin(racine, g))$. On note $fils(g)$ l'ensemble des fils de g. Si $h \in fils(g)$ alors on note $arc(g, h)$ l'arc qui les relie. En outre, $pere(h) = g$.

Définition 4 Le *degré* d'un nœud est le nombre de fils de ce nœud, il est noté $degre(g)$ si $g \in N$. Il est nul si le nœud est une *feuille*. On appelle *nœud interne* un nœud de degré supérieur ou égal à 1.

2.1.2 Propriétés de représentation des facteurs.

L'interêt du trie est que tout facteur y est associé à un nœud, car chaque chemin de la racine vers une feuille représente un suffixe de t. Nous ne pouvons démontrer ces deux propriétés sans détailler l'algorithme de construction du trie, ce qui n'est pas le but de ce chapitre. Nous les écrivons formellement ci-après et les admettons car elles sont utilisées pour le reste de la présentation. Dans celle-ci, nous énonçons des propriétés qui précisent la relation entre un nœud et son fils et montrent que les répétitions de t sont représentés par des nœuds internes.

Propriété 1 Il existe une relation bijective entre l'ensemble des feuilles de l'arbre, i.e. $\{f \in N : degre(f) = 0\}$, et l'ensemble des suffixes du texte, i.e. $\{t_j, 1 \leq j \leq |t|\}$. Par conséquent, si f une feuille du trie, alors il existe un unique j tel que $j \leq |t| : fact(f) = t_j$.

Propriété 2 Il existe une relation bijective entre N et $Fact(t)$.

Ces deux propriétés sont vraies grâce à la précaution de concaténer en fin de texte un caractère unique dans le texte. Dans notre exemple, il s'agit du caractère c. Cela implique qu'aucun suffixe n'est le préfixe d'un autre suffixe. Ceci est exprimé par la condition 1.

Condition 1 t vérifie $\forall\, i < |t| : t[|t|] \neq t[i]$.

Propriété 3 Si t vérifie la condition 1 alors : $\forall\, i < j \leq |t| :\ t_j \neq (t_i)^j$

Preuve Soient $i \leq j \leq |t|$. Raisonnons par l'absurde et supposons que $t_j = (t_i)^j$. Si $t_j = (t_i)^j$ alors :

$$\begin{array}{rrcl} & t_j[|t| - j + 1] & = & (t_i)^j[|t| - j + 1] \\ \Leftrightarrow & t[|t| - j + 1 + (j - 1)] & = & t[|t| - j + 1 + (i - 1)] \\ \Leftrightarrow & t[|t|] & = & t[|t| - j + i] \end{array}$$

ce qui contredit la condition 1 car : $i < j \Leftrightarrow |t| \neq |t| - j + i$ □

Un nœud du trie peut avoir jusqu'à $|A|$ fils car tous les arcs sortant sont étiquettés par des lettres différentes. Ceci est la deuxième condition imposée au trie du texte t.

Condition 2 Soient g un nœud interne de degré supérieur ou égal à 2, e_1, e_2 deux arcs le reliant à deux fils différents, alors $label(e_1) \neq label(e_2)$.

Considérons un nœud interne g et son fils h sur un chemin de la racine à une feuille, chemin qui représente un suffixe de t. Si la longueur de $fact(g)$ est i, alors $fact(g)$ est le préfixe de longueur i de ce suffixe et $fact(h)$ en est le préfixe de longueur $i+1$.

Propriété 4 Soient g un nœud interne, h un de ses fils et i l'entier tel que $|fact(h)| = i+1$. On a : $\forall\ j \leq |t| :\ fact(h) = (t_j)^{i+1}$ alors $fact(g) = (t_j)^i$.

Preuve Soient g un nœud interne, h un de ses fils et i l'entier tel que $|fact(h)| = i+1$. Soient j l'entier tel que $fact(h) = (t_j)^{i+1}$ et f_j la feuille associée au suffixe t_j. Comme g est le père de h, il appartient au chemin de la racine à f_j qui passe par h. Or d'après la définition de $fact$, nous avons :

$$\begin{aligned} fact(h) &= (t_j)^{i+1} \\ &= label(chemin(racine, h)) \\ &= label(chemin(racine, g)).label(arc(g, h)) \\ &= fact(g).label(arc(g, h)) \end{aligned}$$

donc comme $|label(arc(g,h))| = 1$ on a $fact(g) = (t_j)^i$. □

Un nœud interne est de degré supérieur ou égal à 2, ssi son facteur apparaît au moins deux fois dans t, suivi à chaque fois d'une lettre différente. En d'autres termes, s'il est le préfixe de deux suffixes différents, et que la lettre qui le suit est différente dans chacun de ces suffixes. Ou encore, s'il est le préfixe commun maximal de deux suffixes différents. Dans notre exemple, le nœud 2 correspond au plus long préfixe commun aux suffixes t_1 et t_2, i.e. au facteur ab.

Propriété 5 Soient $g \in N :\ degre(g) \geq 2$ et $i \in \mathbb{N} : |fact(g)| = i$. Alors il existe $j < k \leq |t|$ tels que : $fact(g) = (t_j)^i = (t_k)^i$ et $t_j[i+1] \neq t_k[i+1]$.

Preuve Soient $g \in N :\ degre(g) \geq 2$ et $i \in \mathbb{N} : |fact(g)| = i$. g a donc au moins deux fils différents ; soient h_1, h_2 ces deux fils. D'après la propriété 4, comme h_1 est fils de g, on en déduit que $\exists\ j \leq |t| :\ fact(g) = (t_j)^i$ et tel que $fact(h_1) = (t_j)^{i+1}$. En tenant le même raisonnement avec h_2 et g, on obtient $\exists\ k \leq |t| :\ fact(g) = (t_k)^i$ et tel que $fact(h_2) = (t_k)^{i+1}$. Comme $h_1 \neq h_2$ et qu'ils sont fils d'un même père, on sait que $j \neq k$. D'après la propriété 4, on a $fact(g) = (t_j)^i = (t_k)^i$. En outre, d'après la propriété 2 et $h_1 \neq h_2$, on a :

$$fact(h_1) \neq fact(h2) \Rightarrow t_j[i+1] \neq t_k[i+1]$$

□

Un trie contient dans le pire des cas un nombre quadratique de nœuds par rapport à la taille de t. Son nombre de nœuds est en $O(n^2)$ car il y a autant de nœuds que de facteurs dans t. Il n'est donc pas une bonne représentation de $Fact(t)$. Cependant il permet, grâce aux étiquettes des arcs, de savoir si un mot p appartient à t en $O(|p|)$ comparaisons.

2.2 L'arbre des suffixes.

L'arbre des suffixes est simplement une version comprimée du trie. Par conséquent, l'arbre des suffixes contient les mêmes informations que le trie et tous les calculs possibles avec le

trie, le sont aussi avec l'arbre des suffixes. Le fait de comprimer le trie implique que certaines de ses informations soient codées de manière plus astucieuse, ce qui peut compliquer certaines tâches mais aussi en simplifier d'autres (par exemple la recherche d'un facteur dans le texte en est facilitée).

Nous présentons tout d'abord une vision intuitive de l'arbre des suffixes. Ensuite, nous examinons comment l'arbre des suffixes contient toute l'information sur les facteurs du texte, qu'ils soient répétés ou uniques. Nous définissons le concept central de *facteur maximal*. Enfin, nous précisons la complexité de l'arbre des suffixes.

2.2.1 L'arbre des suffixes: une version comprimée du trie.

La différence entre les nœuds de degré 1 et les autres nœuds du trie explique pourquoi l'arbre des suffixes est un trie comprimé. Un nœud de degré 1 a toujours dans sa descendance un nœud de degré différent de 1 (simplement parce qu'il est sur un chemin de la racine à une feuille). On peut donc toujours définir, pour un nœud de degré 1, le plus proche nœud de sa descendance de degré différent de 1, ce nœud est usuellement appelé: *nœud important* (cf. [CR94]).

Dans la figure 2.1, il existe une suite de trois nœuds de degré 1 entre le nœud 2 de degré 2 et la feuille 7 de degré 0. La feuille 7 est le nœud important de tous ces nœuds de degré 1. Le facteur d'un nœud de degré 1 apparaît toujours comme préfixe du facteur de son nœud important. Les informations sur un facteur d'un nœud de degré 1, telles que la présence dans le texte ou ses positions d'occurrence, sont déductibles des informations sur le facteur de son nœud important. En conséquence, un nœud de degré 1 n'apporte pas d'informations par lui-même. Ainsi dans l'exemple, si on connaît toutes ces informations sur la feuille 7, on peut calculer simplement ces informations pour chacun des nœuds de degré 1 entre les nœuds 2 et 7.

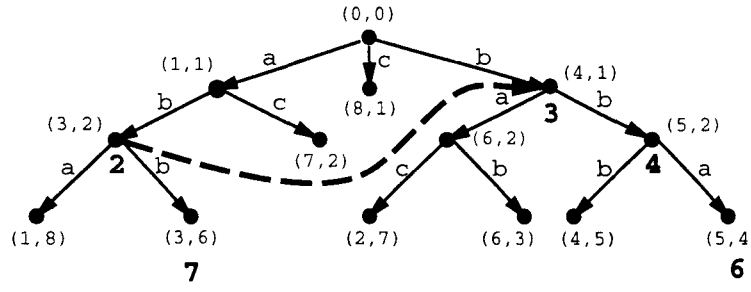


FIG. 2.2 - *Exemple d'arbre des suffixes pour le texte $t = ababbbac$. Le couple (i, j) d'un nœud h donne une position d'occurrence de $fact(g)$, puis sa longueur. Le premier entier du couple (i, j) pour le nœud 2, peut être 3 ou 1. Les numéros des nœuds de la figure 2.2 sont rappelés. Les lettres sur les arcs sont les étiquettes, i.e. la première lettre du label. La flèche en trait discontinu montre un unique exemple de lien suffixe du nœud $(9, 2)$ qui pointe vers le nœud $(4, 1)$ (voir en fin de section).*

Pour comprimer un trie, il suffit donc de ne laisser que les nœuds importants, i.e. de supprimer les nœuds de degré 1, et d'étiquetter les arcs, non plus par des lettres, mais par

des facteurs du texte. Nous emploierons :

label : pour désigner l'étiquette d'un arc de l'arbre des suffixes,

étiquette : pour désigner la première lettre du label d'un arc de l'arbre des suffixes.

Dans l'arbre des suffixes, le chemin entre les nœuds 2 et 7 n'apparaît plus tel quel. Il «s'effondre» et se réduit au seul arc allant du nœud 2 au nœud 7, mais il est étiquetté par le facteur *bbac* (confère la figure 2.2). Du point de vue de la structure mémoire réelle, il est nécessaire de changer la manière de stocker un facteur. Chaque nœud de l'arbre des suffixes représente un facteur par un couple (i, j) où i est l'indice d'une occurrence du facteur et j sa longueur. Les labels des arcs ne sont pas mémorisés mais on peut les trouver grâce à un couple (i, j) et au texte lui-même ; le label d'un arc d'un père à un fils est le mot f tel que $fact(fils) = fact(pere).f$. Le facteur de la feuille 7 dans l'arbre des suffixes est celui qui débute à la position 3 et de longueur 6, donné par le couple $(3, 6)$ qui est mémorisé dans cette feuille.

Notation 1 Soit t un texte sur un alphabet A, on note $ST(t)$ l'arbre des suffixes associé à t. On note N l'ensemble des nœuds de $ST(t)$.

$ST(t)$ doit vérifier la condition suivante :

Condition 3 Pour tout nœud g de N, on a :

- soit $degre(g) = 0$ et g est une feuille,
- soit $degre(g) \geq 2$ et g est un nœud interne.

2.2.2 Un facteur maximal représente une répétition.

Toutes les notations, définitions et propriétés définies pour $trie(t)$ restent valables pour $ST(t)$, sauf les propriétés 2 et 4 qui doivent être modifiées pour s'adapter au fait qu'un arc est étiquetté par un facteur et non plus par une lettre. Dans l'arbre des suffixes, les nœuds de degré 1 du trie sont devenus des *nœuds implicites* (cette définition provient de [CR94].

Définition 5 Soient $g \in N$ et $p \in A^*$. (g, p) est un *nœud implicite* s'il existe h un fils de g tel que p soit un préfixe du label de l'arc qui les relie. Autrement dit si :

$$\begin{array}{ll} \exists\, h \in N : & h \in fils(g), \\ \exists\, i \in \mathbf{N} : & i \neq |label(arc(g,h))| \\ & p = (label(arc(g,h))^i \end{array}$$

Si $p = \varepsilon$ alors (g, p) est un nœud *réel*.

La suppression des nœuds de degré 1 implique que certains facteurs du texte n'ont plus de nœuds associés dans $ST(t)$. Seuls les suffixes et les facteurs qui sont les plus longs préfixes communs à deux suffixes, sont associés à un nœud réel. Nous appelons le deuxième type de facteurs des *facteurs maximaux*. Nous choisissons le qualificatif de maximal, parce que deux suffixes le partagent comme *plus long préfixe commun*.

Définition 6 Soit $f \in Fact(t)$. f est un *facteur maximal* de t si $\exists\, g \in N$: $degre(g) > 1$ et $fact(g) = f$.

Voici le pendant de la propriété 4 pour l'arbre des suffixes.

Propriété 6 Soient g un nœud interne, h un de ses fils, i l'entier tel que $|fact(g)| = i$ et k l'entier tel que $k = |label(arc(g,h))|$. On a: $\forall\, j \leq |t| :\ fact(h) = (t_j)^{i+k}$ on a $fact(g) = (t_j)^i$.

Preuve La preuve est identique à celle de la propriété 4 sauf que $|label(arc(g,h))| = k$ et non plus 1. □

La propriété 7 montre qu'un facteur maximal est bien le plus long préfixe commun à deux suffixes.

Propriété 7 . Soit $f \in Fact(t)$ un facteur maximal de t. Alors il existe $j < k \leq |t|$ tels que: $f = (t_j)^{|f|} = (t_k)^{|f|}$ et $t_j[|f|+1] \neq t_k[|f|+1]$.

Preuve Puisque f est un facteur maximal, il est associé à un nœud du $ST(t)$ de degré supérieur à 1. On obtient la conclusion avec le même raisonnement que dans la propriété correspondante pour le trie, i.e. la propriété 5. □

On en déduit la propriété suivante.

Propriété 8 Il existe une bijection entre $\{h \in N :\ degre(h) > 0\}$ et l'ensemble des facteurs maximaux de t.

Preuve Par construction de l'arbre des suffixes, deux nœuds internes différents représentent des facteurs différents à cause de la condition 2. □

Un facteur, qui n'est ni un suffixe, ni maximal, a un *locus étendu*: le plus petit facteur maximal du texte qui a ce facteur pour préfixe (définition de [Cre76]). Nous n'introduisons pas cette définition, mais plutôt la propriété suivante qui prouve l'existence de cette relation.

Propriété 9 Soit f un facteur qui n'est ni un suffixe, ni maximal, alors il existe un unique nœud $g \in N$ tel que:

- f soit un préfixe propre de $fact(g)$
- $fact(pere(g))$ soit un préfixe propre de f.

On appelle le nœud g, le *nœud étendu* de f.

Preuve Soit f un facteur qui n'est ni un suffixe, ni maximal. On sait qu'il existe $i \in \mathbb{N}$ tel que $f = (t_i)^{|f|}$. Grâce à la propriété 1, on sait qu'il existe une seule feuille $f_i \in N$ telle que $fact(f_i) = t_i$. Appelons c le chemin de la racine à f_i. Tout facteur a pour préfixe le mot vide qui est un facteur maximal, on peut donc définir pour tout facteur différent de ε, le plus grand facteur maximal qui soit préfixe propre de ce facteur. Soit f_1 le plus long facteur maximal préfixe propre de f et g_1 le nœud tel que $fact(g_1) = f_1$. f_1 est aussi un préfixe de t_i et donc g_1 appartient donc à c. g_1 a un fils sur c puisque $degre(g_1) > 1$ et il est unique puisque $ST(t)$ est un arbre. Notons g le seul fils de g_1 sur c. Alors nous avons:

- f est un préfixe de $fact(g)$ puisque ce dernier est aussi un préfixe de t_i et que $|fact(g)| > |f|$,
- de plus f en est un préfixe propre, sinon il serait lui-même un facteur maximal,
- $fact(pere(g)) = fact(g_1)$ est un préfixe propre de f par définition de g_1.

□

Remarque: f est aussi associé de manière unique au nœud implicite (g_1, w) où w est tel que $f = fact(g_1).w$.

2.2.3 Complexité de l'arbre des suffixes.

Quelques différences entre le trie et l'arbre des suffixes sont dues à la représentation des facteurs par un pointeur :

- L'arbre des suffixes seul ne permet plus de connaître la suite des lettres d'un facteur, car les labels des arcs ne sont pas mémorisés ; pour cela il faut garder le texte lui-même en mémoire.

- Par contre, l'arbre des suffixes contient grâce à cette notation, toutes les positions d'apparition de chaque facteur. Pour un nœud g et son facteur $fact(g)$, ses positions sont tous les entiers i des couples (i, j) de chaque feuille du sous-arbre de racine g. Pour le facteur ab stocké dans le nœud 2, il apparaît en position 1 et 3, premier nombre des couples $(1, 8)$ et $(3, 6)$ mémorisés dans les feuilles 7 et 8.

La conséquence la plus importante de ces modifications est la complexité de la structure de données qui devient linéaire alors qu'elle est quadratique pour $trie(t)$.

Propriété 10 Soient t un texte sur l'alphabet A et n sa longueur. La taille mémoire de $ST(t)$ est en $O(n)$.

Preuve L'arbre des suffixes contient exactement n feuilles, et donc au plus $2n$ nœuds, puisque tous les nœuds internes sont de degré supérieur ou égal à 2. □

Il existe trois principaux algorithmes de construction de l'arbre des suffixes : deux méthodes en temps linéaire par rapport à n [Wei73], [Cre76] et une dernière plus récente, dite «on-line» [Ukk92]. Leur présentation et leur comparaison se trouvent dans [CR94]. Nous avons utilisé l'algorithme de McCreight, parce qu'il est plus simple que celui de Weiner et aussi plus économique en espace. En outre, à tout instant lors de l'utilisation de l'arbre des suffixes, on peut constituer la liste des positions de tous les facteurs en temps en $O(n)$).

La construction de [Cre76] définit et utilise des liens suffixe entre les nœuds de l'arbre. Ces liens facilitent la procédure de recherche des nœuds étendus. On note $SL(g)$ le nœud pointé par le lien suffixe de $g \in N$. Soient $g, h \in N$.

$$SL(g) = h \quad \Leftrightarrow \quad fact(h) = fact(g)_2 \tag{2.1}$$

En d'autres termes, le nœud désigné par $SL(g)$ représente le suffixe de $fact(g)$ de longueur $|fact(g)| - 1$, si h existe. Sinon $SL(g)$ pointe sur la racine.

Chapitre 3

Utilisations de l'arbre des suffixes.

Nous utilisons maintenant l'arbre des suffixes pour comprimer un texte. La localisation des occurrences d'un facteur du texte ou la détermination de la présence d'un mot comme facteur du texte, illustrent les principales utilités de l'arbre des suffixes. Un autre exemple : les répétitions du texte sont extraites de l'arbre des suffixes pour permettre la factorisation du texte. Nous collectons ces informations sous la forme d'une liste des facteurs maximaux, avec pour chacun la liste de ses positions. L'arbre des suffixes permet aussi de localiser d'autres régularités que les répétitions, telles que des couples de facteurs symétriques. Nous verrons comment identifier les palindromes génétiques d'une séquence en temps linéaire par rapport à la taille de la séquence.

Nous détaillons dans ce chapitre, ces diverses utilisations ainsi que leur complexité, pour simplifier les présentations ultérieures des algorithmes de compression. Dans l'ordre nous présentons la structure de données employée, la localisation d'un mot et d'un facteur dans le texte (algorithmes qui correspondent aux fonctions *FastFind et SlowFind* dans [CR94]), la construction de la liste des positions pour tous les facteurs et la recherche des palindromes.

3.1 Implantation de l'arbre des suffixes.

Nous avons dit que chaque nœud de l'arbre des suffixes représente de manière univoque un facteur maximal du texte. Dans l'algorithme de construction de McCreight, le facteur représenté par le chemin de la racine à «son» nœud interne est appelé *locus*. Dans l'arbre des suffixes, chaque nœud est mémorisé dans la structure donnée en table 3.1.

Nom	Type	Sémantique
DebutLocus	entier	position de début du facteur maximal associé
LongueurLocus	entier	longueur de ce facteur
LienFils	pointeur	arc reliant un nœud à son premier fils
LienFrere	pointeur	arc reliant un nœud à son frère droit
EtiquetteFils	caractère	lettre qui étiquette l'arc vers le fils
EtiquetteFrere	caractère	lettre qui étiquette l'arc vers le frère
LienSuffixe	pointeur	lien de construction de l'arbre

TAB. 3.1 - *Structure d'un nœud NST (Nœud du Suffix Tree).*

DebutLocus et *LongueurLocus* forment le couple d'entiers (i, j) qui pointe vers une occurrence du facteur associé au nœud. Les liens *LienFils* et *LienFrere* matérialisent les arcs de l'arbre. La mémorisation de l'arbre ne correspond pas exactement au schéma 2.2 car chaque nœud ne contient pas un pointeur vers tous ses fils, mais elle ressemble à celle de la figure 3.1. La condition 2 oblige les labels des fils à débuter par une lettre différente. Ainsi, pour guider une recherche dans l'arbre, on mémorise les étiquettes. Le champ *EtiquetteFils* contient l'étiquette de l'arc vers le premier fils du nœud, et *EtiquetteFrere* celle de l'arc vers son frère droit. Le lien *LienSuffixe* est affecté et mémorisé pour la construction de l'arbre. Sa signification est donnée en fin du chapitre 2 et son utilité est explicitée dans [CR94, Cre76]. Pour mémoriser

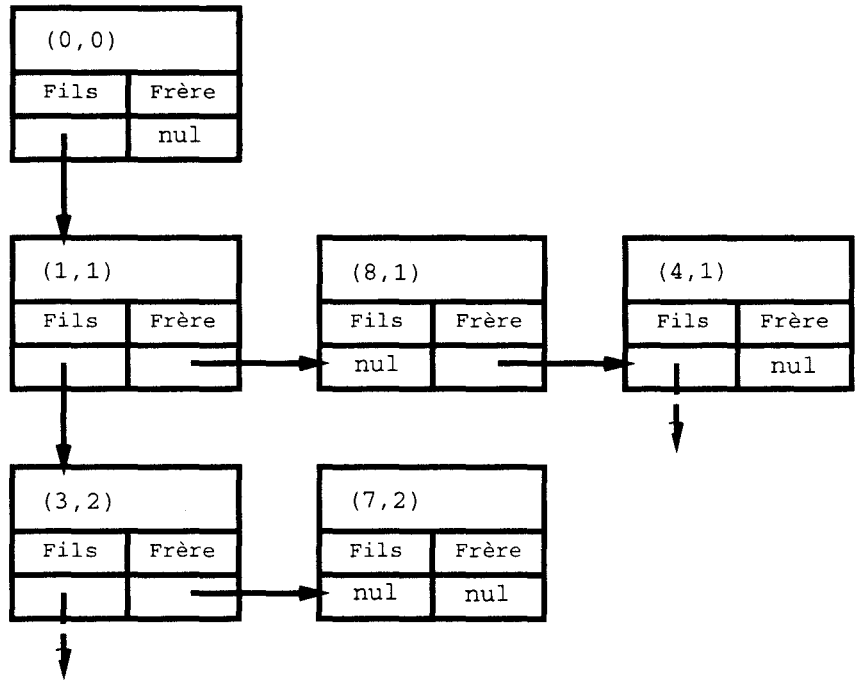


FIG. 3.1 - *Organisation de la structure du haut de l'arbre des suffixes de la figure 2.2. Les nœuds sont représentés par des boîtes où l'on trouve les coordonnées du facteur et les liens PremierFils et FrereDroit. Les liens désignent d'autres nœuds par des flèches ou bien valent le pointeur «nul». Les flèches en trait discontinu indiquent que l'arbre se poursuit mais n'est pas entièrement dessiné ici.*

la liste des positions d'occurrence d'un facteur, nous ajoutons le champ *ListePos* qui est une liste chaînée d'entiers.

3.2 Localisation d'un mot dans le texte.

Nous avons dit que l'arbre des suffixes fut conçu pour faciliter la recherche d'un mot dans un texte. Pendant toute la consultation d'un texte, l'utilisateur d'un éditeur effectue plusieurs recherches de mots différents. Supposons qu'il en fasse 10. S'il emploie une procédure de recherche telle que celle de Knuth, Morris et Pratt ou celle de Boyer-Moore, il lui en coûte 10 fois $O(n)$ opérations élémentaires, où n est la taille du texte. En utilisant une «bonne»

représentation du texte, telle que l'arbre des suffixes, le nombre d'opérations pour chaque recherche est en $O(m)$, où m est la taille du mot à rechercher. On distingue deux sortes de recherche :

- d'une part *ChercherMot*, la procédure qui cherche si un mot p apparaît comme facteur du texte t ; sa réponse est soit le nœud associé au mot, s'il existe, soit le pointeur nul,

- d'autre part *ChercherFacteur*, qui recherche le nœud étendu d'un facteur p dans $ST(t)$; la recherche est plus facile car on sait que ce nœud existe.

Les deux procédures se ressemblent dans leur forme générale. Il s'agit de la recherche d'un nœud par une descente dans l'arbre, guidée grâce aux étiquettes. Elles diffèrent par le fait que l'argument p de ChercherFacteur est forcément un facteur de t, donc il existe soit un nœud étendu soit un nœud réel qui lui est associé. Tandis que *ChercherMot* a pour paramètre un mot sur A^* et ne peut trouver qu'au mieux un nœud étendu pour le plus long préfixe de p qui appartient à $Fact(t)$. Nous ne détaillons que *ChercherMot* et une de ses variantes parce que nous les utilisons, pour ChercherFacteur se reporter à [CR94]. Pour les deux algorithmes, nous supposons que le texte est en mémoire dans une variable globale nommée t, la racine de $ST(t)$ est dans *Racine*.

3.2.1 Algorithme *ChercherMot*.

Nous présentons l'algorithme de *ChercherMot*. Il prend en paramètre 1/ une chaîne de caractères p à rechercher, 2/ dans le texte t, grâce à l'arbre des suffixes dont la racine est *Racine*. Il renvoie un pointeur sur un NST qui est le nœud étendu de p si celui-ci appartient à $Fact(t)$ et le pointeur nul sinon.

```
CHERCHERMOT (CHAR *P, CHAR *T, NST *RACINE)
Var
   pointeur sur NST nœudcourant, pere;
   entier IndiceMot = 0, fin, fin2;
Début
   nœudcourant = Racine;
   TantQue (IndiceMot < |p|) Faire
      pere = nœudcourant;
      Si (nœudcourant→EtiquetteFils == p[IndiceMot]) Alors
         nœudcourant = nœudcourant→PremierFils;
      Sinon
         nœudcourant = nœudcourant→PremierFils;
         TantQue ((nœudcourant != NULL) &&
           (nœudcourant→EtiquetteFrere != p[IndiceMot])) Faire
            nœudcourant = nœudcourant→FrereDroit;
         FinTantQue
         Si (nœudcourant == NULL) Alors
            renvoyer NULL;
         Sinon
            nœudcourant = nœudcourant→FrereDroit;
         FinSi
      FinSi
      fin = nœudcourant→DebutLocus + nœudcourant→LongueurLocus;
      fin2 = IndiceMot +
      min(nœudcourant→LongueurLocus-pere→LongueurLocus+1,|p| - IndiceMot);
      Si (t[nœudcourant→DebutLocus + pere→LongueurLocus + 1..fin] ==
        p[IndiceMot..fin2]) Alors
         Si (IndiceMot == |p|) Alors
            renvoyer nœudcourant;
         Sinon
            renvoyer NULL;
         FinSi
      FinSi
   FinTantQue
Fin CHERCHERMOT.
```

ALGO. 1 - *Algorithme de recherche d'un mot dans l'arbre des suffixes.*

Détail de l'algorithme *ChercherMot.*

Que p soit un facteur de t ou non, on peut définir le plus grand facteur maximal qui soit préfixe de p, notons g le nœud de ce facteur maximal. La variable *nœudcourant* parcourt le seul chemin possible de l'arbre pour aboutir à g. A chaque étape, le prochain nœud à atteindre est choisi parce qu'il est le seul tel que son étiquette égale la lettre courante de p. À chaque fois que *nœudcourant* prend une nouvelle valeur, g' par exemple, on vérifie que $fact(g')$ est préfixe de p. En fait, on ne connaît pas $fact(g)$ mais on le détermine au fur et à mesure de

la descente. Après plusieurs parcours de la boucle principale, *nœudcourant* vaut g. Appelons a la lettre telle que $p^{|fact(g)|+1} = fact(g).a$ si $|p| > |fact(g)|$. En outre s'il existe, notons h le seul fils de g tel que $label(arc(g,h))[1] = a$, i.e. tel que son étiquette soit a. Trois cas se présentent :

- p est un facteur maximal, alors $fact(g) = p$, on retourne g comme résultat.
- $p \in Fact(t)$ mais n'est pas un facteur maximal: la comparaison lettre à lettre de p et de $fact(h)$ montre que p est un préfixe de $fact(h)$. En outre $fact(pere(h)) = fact(g)$ est un préfixe propre de p. Alors d'après la propriété 9, h est le nœud étendu de p et on renvoie h.
- $p \notin Fact(t)$. On sait que :
  - p n'est pas un préfixe de $fact(h)$,
  - $|fact(h)| \geq |p|$,
  - mais que $fact(pere(h)) = fact(g)$ est un préfixe propre de p.

  Donc h est le nœud étendu du plus long préfixe de p qui est un facteur de t et on retourne le pointeur nul car la descente est «bloquée».

3.2.2 Variante de *ChercherMot* : *ChercherPrefixe*.

ChercherMot peut être modifié pour déterminer le plus long préfixe du mot p qui appartient à $Fact(t)$. Nous définissons donc :

*ChercherPrefixe(char *p, char *t, NST *Racine, int LgPrefixe)*

la procédure qui renvoie le locus étendu du plus long préfixe de p qui appartient à $Fact(t)$. Elle renvoie aussi en sortie la longueur de ce préfixe dans *LgPrefixe*. La recherche démarre du nœud pointé par *Racine*. Une autre amélioration consiste à passer en paramètre un nœud interne différent de *Racine* lorsque l'on sait déjà que le locus de ce nœud est un préfixe de p ; ainsi la recherche débute à l'intérieur de $ST(t)$ et est plus rapide. Il convient alors de mettre en entrée dans le paramètre *LgPrefixe* la longueur du préfixe déjà déterminé.

Les complexités en temps de ces deux procédures sont identiques car elles effectuent le même calcul dans le cas où p est un facteur de t, qui est le cas le plus coûteux.

Propriété 11 Les algorithmes *ChercherMot* et *ChercherPrefixe* sont en $O(m)$ où m représente la taille du mot p à localiser.

Preuve Soit m la taille du mot p recherché. L'opération de base est la comparaison de symboles. *ChercherMot* compare au pire chaque symbole de p à un symbole du texte, dans le cas où $p \in Fact(t)$. De plus, pour chaque branchement, *ChercherPrefixe* et *ChercherMot* effectuent au plus $|A|$ comparaisons de l'étiquette d'un arc par rapport au même symbole de p. Le nombre d'opérations de branchement est de l'ordre de m (par exemple avec un texte de la forme a^n et un mot tel que a^m). Si l'on considère A fixé et de taille connue, la complexité en temps est en $O(m)$. □

3.3 Construction de la liste des positions des occurrences d'un facteur.

Une manière de comprimer un texte consiste à remplacer chaque occurrence d'un facteur qui réapparaît plusieurs fois par un pointeur sur sa première occurrence. Appelons cette opération une substitution. L'arbre des suffixes permet d'obtenir facilement la liste de toutes les positions d'un facteur. Ainsi on peut substituer toutes ses occurrences en une étape. Ce travail étant répété pour une multitude de facteurs, nous présentons un algorithme qui construit la liste des positions pour tous les facteurs maximaux, i.e. ceux associés aux nœuds internes de l'arbre des suffixes. Lorsque l'on connaît la liste des positions d'un facteur maximal, on en déduit aisément la liste de positions de n'importe quel facteur dont le nœud étendu est le nœud du facteur maximal.

L'algorithme est basé sur la propriété suivante: pour tout facteur maximal f représenté par un nœud g, il y a égalité entre l'ensemble des positions d'apparition de f et l'ensemble des *DebutLocus* des feuilles du sous-arbre de g. Nous introduisons tout d'abord deux définitions qui formalisent respectivement, l'ensemble des positions d'apparition d'un facteur et l'ensemble des feuilles du sous-arbre d'un nœud. Elles permettent de formaliser et de prouver la propriété ci-dessus. Le schéma de l'algorithme est expliqué ensuite.

Définition 7 Soient $f \in Fact(t)$ et j sa longueur. On définit l'ensemble des positions d'apparition de f dans t, noté $Pos(f)$, l'ensemble suivant $\{i \in [1, |t| - j + 1] : t[i..j + i - 1] = f\}$.

Définition 8 Soit $g \in N$. On définit l'ensemble des feuilles du sous-arbre de g, noté $Feuilles(g)$, l'ensemble suivant $\{h \in N : degre(h) = 0, \exists\ chemin(g, h)\}$.

Nous prouvons d'abord le lemme qui indique que le facteur de tout nœud appartenant à un chemin de $ST(t)$, est préfixe du facteur de ses descendants.

Lemme 1 Soit $(e_1, \ldots, e_i)$ un chemin de $ST(t)$ orienté dans le sens père-fils, alors:

$$\forall\ 1 \leq j \leq k \leq i, \exists\ l : \quad fact(e_j) = (fact(e_k))^l$$

Preuve D'après la propriété 6, on sait que pour tout $1 \leq j < i$, $fact(e_j)$ est préfixe de $fact(e_{j+1})$. Or la propriété de préfixe est transitive, on obtient que quels que soient $1 \leq j \leq k \leq i$, $fact(e_j)$ est préfixe de $fact(e_k)$. □

Propriété 12 Soient f un facteur maximal de t, $g \in N$ tel que $fact(g) = f$, alors on a:

$$Pos(f) = \{DebutLocus(h) : h \in Feuilles(g)\} \tag{3.1}$$

Preuve Soient f un facteur maximal de t et $g \in N$ tel que $fact(g) = f$. Nous devons montrer l'inclusion dans les deux sens.

Preuve de $Pos(f) \subset \{DebutLocus(h) : h \in Feuilles(g)\}$.

Soit i tel que $t[i..i + j - 1] = f$. Grâce à la propriété 1, on sait qu'il existe une feuille, notons la h_i, telle que $fact(h_i) = t_i$. Appelons $c = chemin(racine, h_i)$ le chemin qui la relie à la racine. Il nous suffit de montrer que g est dans c, car dans ce cas f est un préfixe de t_i d'après le lemme 1 et $i \in \{DebutLocus(h) : h \in Feuilles(g)\}$.

Raisonnons par l'absurde et supposons que g ne soit pas dans c. D'après la propriété 8 il n'existe qu'un nœud g possible. Comme f est un préfixe de t_i, il a un nœud étendu dans c d'après la propriété 9. Notons g_1 le plus proche ancêtre commun de ce nœud et de g et g_2, g_3 les deux fils respectifs de g_1 respectivement sur c et sur le chemin de g_1 à g. Comme $f = fact(g) = fact(g_1)^{|f|}$ cela implique que $label(arc(g_1, g_2))$ et $label(arc(g_1, g_3))$ commencent par la même lettre, ce qui contredit la condition 2 et l'hypothèse que $g \notin c$.

Preuve de $\{DebutLocus(h) : h \in Feuilles(g)\} \subset Pos(f)$.

Soient $h_i \in Feuilles(g)$ et $t_i = fact(h_i)$, par définition il existe un chemin de g à h_i. D'après le lemme 1, $fact(g)$ i.e. f est un préfixe t_i, donc $i \in Pos(f)$. □

Un corollaire de cette propriété est que la liste des positions d'un nœud interne est l'union des listes des positions de ses fils.

Corollaire 1 Soit $g \in N$ tel que $degre(g) > 1$, notons $h_1, \ldots, h_j$ ses fils ; alors $Pos(g) = \cup_{i=1}^{j} Pos(h_i)$.

Preuve Soit $g \in N$ tel que $degre(g) > 1$, notons $h_1, \ldots, h_j$ ses fils. Nous devons prouver l'inclusion dans les deux sens.

Preuve de $Pos(g) \subset \cup_{i=1}^{j} Pos(h_i)$.

Soit $k \in Pos(g)$. Par définition de $Pos(g)$, il existe $f \in N$ une feuille telle que $DebutLocus(f) = k$. Comme une feuille d'un arbre ne peut appartenir à deux sous-arbres distincts, il existe $i \in [1, j]$ tel que $k \in Pos(h_i)$. Donc $k \in \cup_{i=1}^{j} Pos(h_i)$.

Preuve de $Pos(g) \supset \cup_{i=1}^{j} Pos(h_i)$.

Soit $k \in \cup_{i=1}^{j} Pos(h_i)$. Par définition de Pos, on sait que $\cap_{i=1}^{j} Pos(h_i) = \emptyset$ puisque les h_i appartiennent à des sous-arbres différents. Donc il existe $i \in [1, j]$ tel que $k \in Pos(h_i)$. Dans ce cas, on sait que k est le *DebutLocus* d'une feuille du sous-arbre de h_i. Or cette feuille appartient aussi au sous-arbre de g, donc $k \in Pos(g)$. □

L'algorithme qui permet de construire la liste des positions d'un facteur maximal du nœud g, est constitué d'un parcours en profondeur d'abord du sous-arbre dont la racine est g. Pendant ce parcours, on collecte dans une liste les valeurs du champ *DebutLocus* de toutes les feuilles rencontrées. Nous décrivons l'algorithme qui permet de constituer **les listes de positions de tous les facteurs maximaux** i.e de tous les nœuds internes. Nous l'appelons *TousFacteurs*. Cet algorithme est aussi un parcours en profondeur d'abord, mais à partir de la racine et il traite les nœuds de deux manières différentes :

- si le nœud est une feuille : il met la valeur de *DebutLocus* dans la liste courante des positions,
- si le nœud est interne : la procédure est appelée récursivement pour construire la liste de chacun de ses fils (grâce au corollaire 1), lorsque tous les appels sont finis, il fait l'union des listes des fils dans le champ *ListePos* du nœud courant.

En outre, cet algorithme retourne comme résultat la liste des facteurs maximaux. Cette liste des facteurs, appelée *ListeFacteurs*, est ordonnée selon la longueur décroissante des facteurs maximaux, parce que nos algorithmes de compression substituent préférentiellement les plus grands facteurs.

ListeFacteurs est en fait une liste chaînée de nœuds de l'arbre des suffixes avec le champ ListePos à jour. Comme elle est à la base d'un algorithme de compression, nous en détaillons

la complexité. Il ne peut y avoir qu'au plus $n-1$ nœuds dans cette liste, puisqu'il s'agit de la liste des nœuds internes de $ST(t)$, qu'ils sont tous de degré supérieur ou égal à 2 et que l'arbre contient exactement n feuilles.

Propriété 13 Le nombre d'éléments de *ListeFacteurs* est en $O(n)$.

Preuve Son nombre d'éléments est borné par le nombre de nœuds de $ST(t)$, qui est en $O(n)$ (confère la propriété 10). □

Pour un facteur maximal donné, i.e. pour un élément de *ListeFacteurs*, sa liste des positions est aussi en $O(n)$.

Propriété 14 Le nombre d'éléments de la liste des positions d'un facteur de t est en $O(n)$.

Preuve Le nombre de positions d'un facteur est au plus linéaire, puisque ce facteur ne peut apparaître qu'au plus n fois. Pour le a dans le texte a^n, le nombre de positions est exactement de n. □

3.4 Localisation des palindromes.

Pour comprimer, on peut substituer les répétitions des facteurs maximaux par un pointeur qui désigne la première occurrence du facteur dans le texte. Si le code du pointeur est moins grand que l'occurrence du facteur, on comprime effectivement le texte. On peut appliquer la même méthode avec n'importe quel couple de facteurs (u, v) de t, à condition que $1/u$ et v ne se chevauchent pas et que $2/\ v$ se déduise de u. Les couples (u, v) non chevauchants, qui forment un palindrome génétique de t, remplissent ces conditions.

Dans ce but, nous esquissons un algorithme qui permet de déterminer les plus grands palindromes génétiques d'un texte t, i.e. ceux qui représentent la meilleure compression potentielle, grâce à l'arbre des suffixes. *ChercherPalindrome1* permet d'obtenir pour toute position i dans le texte, le plus long palindrome débutant en i, i.e. le couple de facteurs $(u, \overline{u})$ tel que :

- u et $\overline{u}$ ne se chevauchent pas
- $u = t[i..i + |u|]$
- $|u|$ soit maximale
- $(u, \overline{u})$ forme un palindrome génétique.

Cet algorithme constitue une liste ordonnée selon la taille de u, de tous les couples trouvés. Il consiste à rechercher pour une position i, le plus long préfixe de $(\overline{t})_i$ dans $ST(t)$ grâce à la

procédure *ChercherPrefixe*. Le pseudo-code est donné ci-dessous.

```
CHERCHERPALINDROME1
Var
   NST : n
   entier : i, LgPal
Début
   i ← 1
   TantQue (i < |t|) Faire
      LgPal ← 0
      n ← CherherPrefixe(t̄_i, t, Racine, LgPal)
      Si (LgPal > TAILLE_MIN_PAL) Alors
         Inserer (|t| − i − LgPal + 2, n → DebutLocus, LgPal) dans la liste
      FinSi
      i ← i + 1
   FinTantQue
   renvoyer la liste des palindromes
Fin CHERCHERPALINDROME1.
```

ALGO. 2 - *Algorithme de recherche des palindromes dans un texte t (première version).*

Pour toutes les positions i, on recherche un palindrome dont la taille est limitée par n en $O(n)$ opérations, grâce à *ChercherPrefixe*. La complexité temporelle de *ChercherPalindrome1* est donc en $O(n^2)$. Par ailleurs, si l'on trouve un palindrome de longueur k à la position i, on trouve au moins un palindrome de longueur $k-1$ en $i+1$, un palindrome de longueur $k-2$ en $i+2$, etc. *ChercherPalindrome1* a deux désavantages pour une utilisation en compression :

- il insère dans la liste des palindromes strictement inclus les uns dans les autres, et donc inutilisables parce qu'ils se chevauchent,
- il fait des comparaisons inutiles, dont le résultat est déjà connu.

Nous évitons ces recherches en utilisant le lien suffixe du nœud n. À l'étape i, la recherche de *ChercherPrefixe* démarre de $(n \rightarrow SuffixLink)$ à lieu de *Racine*, avec *LgPal* ayant pour valeur : la longueur du palindrome de l'étape $i-1$ moins 1. En outre, nous n'insérons le palindrome maximal à la position i que s'il n'est pas strictement inclus dans celui trouvé à la position $i-1$. Ceci forme l'algorithme de *ChercherPalindrome2*.

Notation 2 On note *ListePalindromes* : la liste des palindromes qui est résultat de l'exécution de *ChercherPalindrome2*.

Propriété 15 La complexité temporelle de *ChercherPalindrome2* est en $O(n)$. Il en va de même pour la complexité en espace.

Preuve Grâce à l'utilisation du lien suffixe du nœud de l'étape $i-1$, dans l'appel de Cherher-Prefixe à l'étape i, un caractère de $\bar{t}$ n'est jamais comparé qu'une fois avec un caractère de t.

```
CHERCHERPALINDROME2
Var
   NST : n, depart
   entier : i, LgPal, LgPrec
Début
   LgPal ← 1
   i ← 1
   depart ← Racine
   TantQue (i < |t|) Faire
      LgPrec ← LgPal-1
      LgPal ← LgPrec
      n ← ChercherPrefixe(t̄_i, t, depart, LgPal)
      depart ← n → SuffixLink
      Si (LgPal > TAILLE_MIN_PAL) et (LgPal > LgPrec) Alors
         Inserer (|t| − i − LgPal + 2, n → DebutLocus, LgPal) dans la liste
      FinSi
      i ← i + 1
   FinTantQue
   renvoyer la liste des palindromes
Fin CHERCHERPALINDROME2.
```

ALGO. 3 - *Algorithme de recherche des palindromes dans un texte t (deuxième version).*

Comparativement à *ChercherPalindrome1*, cela rend la complexité temporelle linéaire. En espace, l'algorithme utilise simultanément les chaînes de caractères t, $\bar{t}$ et $ST(t)$, ce qui ne coûte que $O(n)$ emplacements mémoire d'après la propriété 10. Les éléments de *ListePalindromes* sont de taille bornée et on en compte au plus n dans la liste, soit un par position dans t. Donc la taille mémoire de *ListePalindromes* est en $O(n)$. □

Chapitre 4

Autres méthodes de compression de texte.

Le premier objectif de ce chapitre est d'apporter une compréhension globale des méthodes de compression statistiques et substitutionnelles, classiques dans le domaine de la compression de textes. Le second but est d'expliquer la méthode développée dans [GT95] qui est dédiée aux séquences nucléiques. Elles appartiennent au monde des méthodes de compression sans perte d'informations, parce qu'elles s'appliquent aux textes. Nous séparons la présentation des méthodes généralistes en deux sections distinctes parce qu'elles sont fondamentalement différentes. Elles s'opposent dans leur «vision» du texte à comprimer, dans la compréhension ou dans le modèle qu'elles se font du texte. La dernière section est consacrée à l'algorithme de [GT95] : Biocompress.

Nous employons le concept de *schéma de compression* pour désigner une méthode (ou algorithme) de compression et de décompression. Quelles que soient les méthodes abordées ici, leur complexité est linéaire par rapport à la taille du texte comprimé.

4.1 Compresseurs statistiques.

Le qualificatif de «statistique» provient du fait que la compression n'utilise que des informations de type statistique pour comprimer un texte. Le modèle de processus de génération d'une séquence de caractères est un processus aléatoire qui tire chacun des symboles au sort dans l'alphabet, en respectant des probabilités d'apparition données. Garder ceci en tête est nécessaire pour identifier les limites de ces méthodes.

Pour coder une suite de nombre réels, par exemple une suite d'échantillons d'un morceau de musique, nous pouvons coder chaque valeur en pensant qu'elle diffère peu de la précédente et en ne donnant que leur différence. Il s'agit là d'une prédiction d'une valeur à venir et d'un codage de la différence entre la prédiction et la valeur réelle. Une idée similaire règle le fonctionnement d'une méthode statistique, mais d'un point de vue global. Ce genre de méthode se fait un modèle du texte grâce à des informations statistiques. Le code du texte sera d'autant plus court que le modèle est adéquat vis à vis du texte, et inversement. Cette vision de la compression est logique, puisque si le décompresseur sait prédire parfaitement les prochains symboles, il suffit de lui donner le début et il devine le reste : dans ce cas idéal la compression est énorme. Dans la réalité, la prédiction est erronée mais les méthodes statistiques parient que, coder les symboles en fonction des prévisions est moins coûteux que

de transmettre les symboles originaux. La tâche de prédire au mieux le prochain caractère est nommée *modélisation* et celle de coder au mieux les différences est appelée *codage*. D'où, la notion importante de modèle du texte ou du processus de création du texte. Par exemple, un modèle simple est donné par les probabilités d'apparition des lettres dans le texte. Meilleur est le modèle, plus élevée sera la compression obtenue.

Nous présentons ici les schémas de codage statistique les plus connus : le codage de Huffman et le codage arithmétique. Ces méthodes de codage sont paramétrées par le modèle, le code qu'elles produisent dépend de ce modèle, mais pas la méthode pour le construire. Les tâches de modélisation et de codage sont parfaitement séparables. Suivant la sophistication du modèle en paramètre, le code produit pour un texte sera plus ou moins court. Nous décrivons les deux méthodes de codage, puis nous abordons les divers modèles employés.

4.1.1 Le codage de Huffman.

En 1952, Huffman imagina une méthode pour construire des codes de redondance minimale ([Huf52]). **À partir d'une distribution de probabilités des symboles d'un alphabet** (le modèle), cette méthode associe à chaque symbole, un code binaire, dont la longueur est la plus proche possible **de l'entropie du symbole** (voir ci-dessous).

Notions de la théorie de l'information de Shannon.

Nous précisons, tout d'abord quelques notions de la première théorie de l'information, formulée en 1948 par Shannon ([Sha48, SW49]). L'entropie du symbole i de probabilité p_i, notée E_i, est donnée par :

$$E_i = -\log(p_i) \tag{4.1}$$

si le log est en base 2, son unité est le bit. Cette entropie mesure le nombre de bits moyen nécessaire pour coder le symbole i (bien sûr, la moyenne dépend de la distribution des probabilités). La notion d'entropie est étendue pour s'appliquer à la distribution de probabilités. Alors elle calcule le nombre de bits moyen nécessaire pour exprimer le résultat d'un choix entre tous les symboles de l'alphabet. Notée E, elle est définie par :

$$E = -\sum_i p_i \log(p_i) \tag{4.2}$$

Comme cette distribution des probabilités modélise le processus de choix, cette entropie est aussi appelée entropie de la source.

L'entropie d'un texte, qui donne le nombre moyen de bits par symbole, utilise la même formule que l'équation 4.2, mais avec les probabilités d'apparition des caractères observées dans le texte. Ces probabilités observées «approximent» la distribution de probabilité d'une source imaginaire qui a généré le texte. Dans l'article «A Mathematical Theory of Communication», Shannon démontre que la taille moyenne du codage **de chaque symbole** peut au mieux atteindre l'entropie de la source. Ce théorème est connu sous le nom «Noiseless source coding theorem» ou théorème pour le codage d'une source sans bruit.

Algorithme du codage de Huffman.

L'algorithme de Huffman construit un arbre binaire dont les arcs sont étiquetés par $\{0, 1\}$ et dont chaque feuille est associée à un symbole. L'idée principale sous-jacente au codage

de Huffman et à celui de Shannon-Fano (une méthode similaire mais moins bonne que le lecteur peut trouver dans [BCW90, Nel91]) est la suivante. Puisque nous sommes doté d'une distribution de probabilités des symboles, nous savons quels sont les symboles courants et ceux moins courants. Pour associer à chacun d'eux un code binaire de longueur entière qui minimise la longueur du code moyen, il suffit d'associer un code court aux symboles probables et un code long aux symboles rares. L'algorithme est écrit ci-dessous.

HUFFMAN

Début
lister les symboles et leurs probabilités
trouver les deux symboles de plus basses probabilités
créer une feuille pour chaque symbole
créer un nœud interne reliant ces deux feuilles
étiqueter l'arc de gauche par 0 et celui de droite par 1
remplacer dans la liste, les deux symboles par le nœud interne
associer au nouveau nœud la somme des probabilités des symboles supprimés
appeler récursivement jusqu'à ce qu'il n'y ait plus qu'un symbole
Fin.

ALGO. 4 - *Algorithme de construction de l'arbre d'Huffman, selon les probabilités des symboles de l'alphabet.*

Symbole	Probabilité	Entropie	Code	$\|Code\|$ en bits	Approximation
A	$\frac{1}{2}$	1	0	1	0
C	$\frac{1}{4}$	2	10	2	0
G	$\frac{1}{8}$	3	110	3	0
T	$\frac{1}{8}$	3	111	3	0
A	0.96	0.059	0	1	- 0.41
C	0.02	5.64	10	2	+ 3.64
G	0.01	6.64	110	3	+ 3.64
T	0.01	6.64	111	3	+ 3.64

TAB. 4.1 - *Comparaison des codes de Huffman et de l'entropie, pour deux distributions de probabilités différentes. Entropie de la première distribution: 1.75 bits, de deuxième: 0.30 bits ! Cependant les codes attribués aux lettres sont identiques.*

Une fois l'arbre construit, chaque symbole peut être traduit en binaire. La transformation en binaire de tous les symboles produit le codage d'un texte complet. Le codage des symboles par cette méthode ne prend en compte que le classement des symboles par ordre des probabilités et pas les probabilités elles-mêmes. Ainsi, pour deux distributions éloignées, le codage peut être identique ; voir les deux exemples sur un alphabet à quatre lettres de la figure 4.1. Pour chaque caractère, la taille du code approxime l'entropie à l'entier le plus proche et en restant le plus économique possible; donc la taille du code moyen par Huffman vaut l'entropie seulement si les probabilités sont des puissances de $\frac{1}{2}$. Pour les codages de Huffman de la

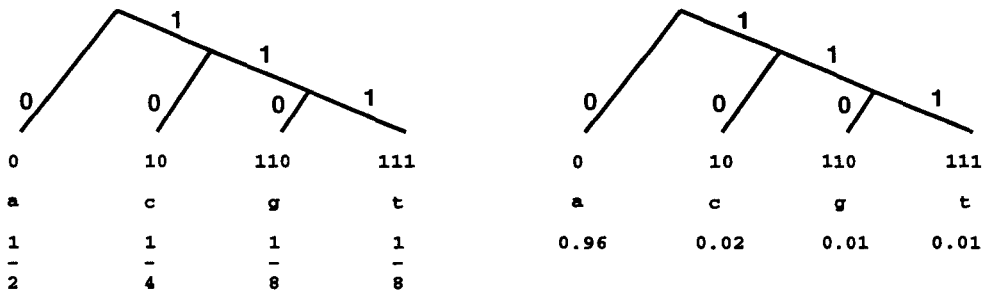


FIG. 4.1 - *Exemple d'arbres de Huffman pour des distributions de probabilités différentes.*

figure 4.1, le tableau 4.1 donne pour chaque symbole: son entropie, la taille de son code et l'approximation de celui-ci par rapport à l'entropie. Comme un texte est codé par la suite des codes de ses symboles, on accumule les approximations à chaque symbole. La longueur du code du texte peut alors être bien plus long que ne le dicte son entropie. On peut approcher l'entropie en groupant les lettres par 2, 3, etc, pour former de nouveaux «symboles», cependant l'arbre des codes à transmettre devient vite prohibitif (avec un modèle statique).

4.1.2 Le codage arithmétique.

Comparativement à la méthode de Huffman, ce codage évite l'écueil de l'approximation entière de l'entropie pour le code d'un symbole. Il ne code pas les symboles, mais le texte entier. Chaque texte est codé par un intervalle de $[0, 1]$, dont la taille vaut la probabilité du message. Le code est d'autant plus long que la probabilité est faible, i.e. que la taille du sous-intervalle de $[0, 1]$ est petite, et inversement. Son avantage sur Huffman est donc d'ajuster la taille de son code sur l'entropie du message et non grâce à celle des symboles. **Ainsi le code d'un texte atteint exactement la longueur dictée par son entropie: le codage arithmétique est donc optimal** d'après le «Noiseless source coding theorem». Son deuxième avantage est de calculer ce code incrémentalement. Enfin, il convient bien à l'utilisation d'un modèle adaptatif.

Pour une distribution de probabilités sur les symboles, on calcule les probabilités cumulées des symboles dans un ordre quelconque mais fixe. On stocke ces valeurs dans un tableau *ProbaCum* indicé par les symboles (la probabilité cumulée d'un caractère i est dans $ProbaCum[i]$ et celle du caractère précédent dans $ProbaCum[i-1]$). De cette manière, on découpe l'intervalle $[0, 1]$ en associant un sous-intervalle $[ProbaCum[i-1], ProbaCum[i]]$ à chaque symbole, et la taille de ce sous-intervalle vaut sa probabilité (voir l'exemple dans la table 4.2).

À chaque préfixe du texte, l'algorithme va associer un sous-intervalle de $[0, 1]$, qui ne dépend que de celui du préfixe précédent et du caractère courant. Le codage du texte est effectué par la boucle qui construit incrémentalement l'intervalle de chaque préfixe, en lisant les symboles de gauche à droite. Initialement, l'intervalle vaut $[0, 1]$, à chaque symbole nouveau on transforme l'intervalle courant de la manière suivante. On lui applique proportionnellement le même découpage en sous-intervalles que celui initialement appliqué à $[0, 1]$. Le sous-intervalle correspondant au caractère courant devient le nouvel intervalle courant. Cette transformation est contenue dans la boucle principale de l'algorithme 4.1.2, elle est aussi illustrée dans

Symbole	Indice i	Probabilité	$ProbaCum[i]$	Intervalle
–	0	0	0	
A	1	0.2	0.2	[0, 0.2[
C	2	0.4	0.6	[0.2, 0.6[
G	3	0.15	0.75	[0.6, 0.75[
T	4	0.25	1	[0.75, 1[

TAB. 4.2 - *Tableau des probabilités cumulées pour le codage arithmétique. On donne pour chaque caractère, son indice dans le tableau et sa probabilité cumulée dans l'entrée* $ProbaCum[i]$.

```
CODAGE ARITHMÉTIQUE
Var
   caractère: car \\caractère courant
   entier: haut, bas \\borne haute et basse de l'intervalle courant
Début
   bas ← 0
   haut ← 1
   TantQue (non fin de texte) Faire
      lire(car)
      largeur ← haut - bas
      haut ← bas + largeur * ProbaCum[car]
      bas ← bas + largeur * ProbaCum[car-1]
      envoyer en sortie les bits de poids fort commun à haut et bas
   FinTantQue
Fin CODAGE ARITHMÉTIQUE.
```

ALGO. 5 - *Algorithme d'encodage arithmétique selon les probabilités des symboles de l'alphabet.*

la figure 4.2.

Au fur et à mesure que l'intervalle se rétrécit, la limite supérieure (variable *haut* dans l'algorithme 4.1.2) se rapproche de la limite inférieure (variable *bas*). Les bits de poids fort de ces deux variables deviennent identiques et donc inutiles à mémoriser puisque définitivement fixés jusqu'à la fin du codage du texte (en effet, l'intervalle ne peut que rétrécir). Les bits de poids fort sont imprimés en sortie dès qu'ils sont communs à *haut* et à *bas* durant le codage. On imprime ainsi jusque et y compris le premier bit qui les différencie. Un caractère de fin de texte est codé en plus de ceux du texte, pour que le compresseur puisse déterminer la fin du code. La figure 4.2 montre pour la distribution des probabilités donnée dans la table 4.2, la construction de l'intervalle et le codage pour le texte CGA.

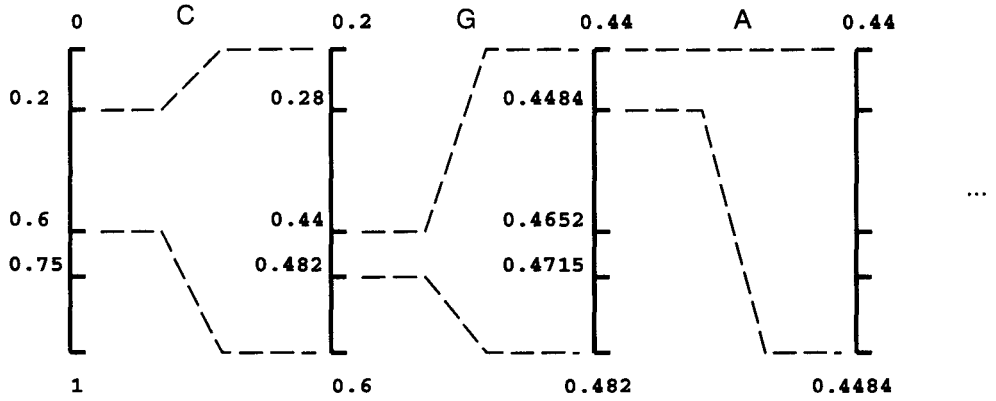


FIG. 4.2 - *Exemple de transformation de l'intervalle par la procédure de codage arithmétique pour le texte CGA. Le segment vertical représente l'intervalle courant à chaque étape. Les traits discontinus montrent la «dilatation» du sous-intervalle choisi pour la lettre courante : intervalle* $[0.2, 0.6]$ *pour la lettre C. Il est ensuite redécomposé en sous-intervalles pour coder la lettre suivante, etc. La distribution des probabilités est celle de la table 4.2.*

4.1.3 Les modèles.

Rappelons-le, un schéma de compression statistique comprend un modèle et un encodeur. L'intérêt pour la compression est que le modèle prédise au mieux le texte, ce qui est la tâche la plus difficile puisque nous avons vu qu'il existe une méthode de codage optimale pour un modèle donné : le codage arithmétique. On distingue deux catégories de modèles :

- Soit le compresseur et le décompresseur se mettent d'accord sur un modèle précis, ils utilisent alors un modèle *statique*. Par exemple, pour un schéma de compression dédié à la compression de textes français, on peut inclure dans le compresseur et le décompresseur, une distribution fixée des probabilités caractéristiques de la langue française. Dans ce cas, les informations du modèle ne sont pas extraites du texte.

- Le deuxième cas oblige compresseur et décompresseur à modifier de la même manière une distribution des probabilités en fonction du texte déjà lu. Ils doivent initialiser pareillement le modèle et utiliser une procédure commune d'altération du modèle, alors la décompression est garantie. Le modèle est dit *adaptatif*.

Le modèle le plus efficace est celui qui engendre la meilleure prédiction. Si l'on veut un modèle parfait de ce point de vue, il suffit de le calculer en une première passe sur le texte, avant d'en effectuer le codage. Il s'agit alors d'un modèle *ad hoc*. L'inconvénient est que le décompresseur ne connaît pas ce modèle et il faut donc le lui transmettre dans le fichier compressé. Or la proportion du modèle dans le fichier compressé, augmente avec sa complexité, qui va de pair avec sa capacité prédictive. L'autre alternative offerte par un modèle statique, est celle du modèle fixe indépendant du texte en entrée qui risque donc de mal le prédire et de mal le compresser. Parce qu'ils ne souffrent pas de ces inconvénients, les modèle adaptatifs forment le meilleur compromis.

Les modèles que nous utilisons sont, soit le modèle statique ad hoc à cause de la taille réduite de l'alphabet des séquences génétiques, soit des modèles adaptatifs à contexte fini.

Dans ces derniers, la probabilité dépend d'un nombre fini de lettres précédant le caractère courant. L'ordre du modèle indique la taille du contexte pris en compte dans le calcul des probabilités d'apparition de chaque symbole. Nous utilisons ces modèles, dits markoviens, d'ordre 0 jusqu'à 3. Dans un modèle à contexte fini d'ordre 2 pour un texte en français, la probabilité d'un caractère, *e* par exemple, est calculé selon le couple de symboles qui le précède, elle sera différente si ce couple est *qu* ou *s*.

4.2 Compresseurs à dictionnaire ou substitutionnels.

Les compresseurs à dictionnaire n'utilisent pas des propriétés statistiques du texte, mais codent des régularités telles que des répétitions. Certains facteurs (segments contigus du texte) qui sont réguliers forment le dictionnaire, alors une occurrence dans le texte d'un de ces facteurs est remplacée par un pointeur vers l'élément du dictionnaire. Ce pointeur constitue le code binaire qui remplace le facteur, d'où le qualificatif de *substitutionnel.* Les schémas de compression diffèrent par leur dictionnaire et/ou par le codage de leur pointeur. Deux méthodes maintenant célèbres, des mêmes auteurs sont devenues les paradigmes de cette classe de compression : elles sont appelées méthodes de Lempel-Ziv. La première date de 1977 ([ZL77]) et la seconde de l'année 1978 ([ZL78]). Elles font l'objet des deux sous-sections suivantes et sont aussi notées *LZ77 et LZ78.*

Nous notons A un alphabet, t un texte sur A, n sa taille, p un pointeur substitué à une répétition et D un dictionnaire.

4.2.1 Méthode de Lempel-Ziv 77 ou compression à fenêtre glissante.

[ZL77] parcourt le texte de gauche à droite et recherche le plus grand facteur qui débute à la position courante et qui apparaît dans les N précédents caractères. Ces N précédents symboles ont déjà été encodés et constituent la fenêtre glissante qui sert de dictionnaire à la méthode. La taille de la répétition recherchée est limitée à T, qui est un paramètre de la méthode tout comme N. Les T caractères à droite de la position courante constituent le tampon ou «lookahead buffer». Le pointeur envoyé en sortie prend la forme d'un triplet d'items qui sont : la position de début dans la fenêtre, la longueur de la répétition, le caractère qui suit l'occurrence du facteur dans le tampon.

Ainsi, le pointeur ne peut référencer qu'un facteur de taille limitée, qui est répété dans un dictionnaire local de taille fixée. Cela limite drastiquement les répétitions encodables pour la compression, mais permet de coder le pointeur sur un nombre fixe de bits. La taille de n'importe quel pointeur est :

$$|p| = \lceil \log(N) \rceil + \lceil \log(T) \rceil + \lceil \log(|A|) \rceil \tag{4.3}$$

de plus, le dictionnaire est :

$$D = \{m.a : \quad m \in Fact(t[i-N+1..i]), \quad |m| \leq T, \quad a \in A\} \tag{4.4}$$

Outre la limitation de la fenêtre, le principal désavantage de ce schéma est sa complexité. Elle est linéaire, mais chaque pas exige que la fenêtre soit parcourue pour trouver la plus longue répétition. Si la taille de la fenêtre n'était pas bornée et fixe, la complexité serait en $O(n^2)$.

La suppression du caractère dans le pointeur est une amélioration de [Bel86] dans la méthode notée LZSS, nous employons cette version dans nos tests. La limitation de la taille de la fenêtre fût levée par [RPE81], mais n'apporte pas d'améliorations sensibles d'après [BCW90]. L'utilisation de pointeurs de taille variable selon la taille de la répétition codée ([Bel87]), n'atteint pas le niveau de compression de la méthode LZ78.

4.2.2 Méthode de Lempel-Ziv 78.

LZ78 est un schéma qui évite élégamment la limitation de la taille de la fenêtre. Un pointeur peut y référencer n'importe quelle partie du texte déjà codé. L'autre principal avantage est que la recherche de la plus grande répétition pour la position courante est effectuée rapidement. Ceci grâce au fait que le dictionnaire est mémorisé dans une structure de données adéquate et de taille fixe. Le dictionnaire est *adaptatif*: à chaque position, le décompresseur et le compresseur modifient le dictionnaire de la même manière. Ainsi, la taille du pointeur peut varier suivant la position courante, tant qu'elle permet de référencer n'importe quel élément dans le dictionnaire courant. Comme le dictionnaire augmente de taille, le pointeur aussi. Le dictionnaire est un index des mots déjà rencontrés, à chaque mot est associé un numéro d'ordre dans D.

L'algorithme de compression est le suivant. À chaque position, on recherche le plus grand facteur qui y débute et appartient à D. On le substitue par un pointeur à deux items : l'index du facteur dans le dictionnaire et le symbole qui le suit à la position courante. On a donc :

$$|p| = \lceil \log(|D|) \rceil + \lceil \log(|A|) \rceil \tag{4.5}$$

et si i dénote la position courante, le dictionnaire est :

$$D = \{m.a : m \in Fact(t), a \in A\} \tag{4.6}$$

En outre, pour chaque position on rajoute une entrée dans D, constituée par le mot que l'on vient de substituer i.e. $m.a$. Initialement, D est vide. Le dictionnaire peut être implanté dans une structure de trie, voir le chapitre 2.

Le seul paramètre de l'algorithme est la taille maximale d'un index dans le dictionnaire, qui limite le nombre de mots mémorisés. On dira ainsi : un compresseur LZ78-12 bits ou LZ78-15 bits selon la taille d'index employée. De nombreuses versions tentent d'améliorer LZ78. [Wel84] exclut le caractère dans le pointeur en initialisant D avec une entrée par lettre de A (nous utilisons la méthode de Welch, notée LZW, dans nos tests). Les autres versions connues se distinguent le plus souvent par le problème crucial de *la gestion du dictionnaire*. Elle se résume en deux questions :

1. Que faire lorsque D est plein?
   En effet, le nombre d'entrées dans D étant limité à cause de la taille maximale d'un index, D peut être plein. LZ78 comme LZW se contentent de réinitialiser D. D'autre heuristiques ont été testées. La version de [TMD+85], célèbre parce qu'elle constitue l'utilitaire «compress» d'UNIX, réinitialise D lorsque le taux de compression courant faiblit. [Tis87] supprime les entrées les moins souvent utilisées, méthode dite du «Least Recently Used (LRU) replacement» en anglais.

2. Quels mots y insérer?
   Le problème est crucial puisqu'il influe sur la taille de D et donc sur celle de l'index

qui s'adapte à cette dernière. Outre LZ78, une autre heuristique qui se rapproche de nos méthodes est celle de [MW84], où les entrées du dictionnaire sont formées par la concaténation des deux derniers mots codés. Ainsi, les facteurs dans D sont plus grands en taille, ce qui permet une amélioration du taux de compression. Cependant, jamais ces méthodes ne cherchent à savoir si un mot qu'elles insèrent est répété à droite dans le texte, et correspond effectivement à un index qui sera employé dans le codage. Ceci est fait par notre compresseur qui n'insère que des mots qui seront encodés (voir chapitre 5).

4.3 Une méthode dédiée aux séquences génétiques : Biocompress.

Nous détaillons un schéma de compression dédié aux séquences nucléiques qui se nomme Biocompress ([GT93a, GT93b, GT95]). Par ailleurs, nous signalons une recherche faite à Argonne qui promeut l'applicabilité de la compression pour l'identification de séquences simples d'ADN (voir [Mil93, MJ93a, MJ93b]). Cette recherche utilise un compresseur classique de type LZ78 (elle n'est donc pas détaillée ici).

Biocompress.

Développé à Rocquencourt, Biocompress résulte de la première tentative dans notre communauté de compression de séquences biologiques. Biocompress est basé sur le parcours séquentiel de gauche à droite ou $5' - 3'$ de la séquence. Il détecte et exploite deux types de régularités déjà citées : répétitions exactes et palindromes génétiques. Son principe est similaire à celui de LZ77, mais sans limitation sur la taille de la fenêtre. Un facteur répété est remplacé par un pointeur de la forme $(longueur, position)$ où les items entiers sont donc auto-délimités. À l'instar de [RPE81], le codage contient donc alternativement, des suites de pointeurs et des suites de symboles encodés sur 2 bits (les symboles sont les 4 bases). Pour le détail des codages utilisés, voir [GT93a].
À chaque position dans la séquence, Biocompress :

- recherche le plus long palindrome et le plus long facteur dans la partie de la séquence déjà examinée (i.e. dans une fenêtre allant du premier caractère au caractère courant), apparaissant aussi à la position courante
- si l'un des deux est suffisamment long, il est remplacé dans le texte par un code indiquant sa longueur et sa position de première apparition. Notons ce code (l, p).

La condition "suffisamment long" signifie : le code (l, p) que l'on désire substituer à une apparition du facteur, est-il plus court que le facteur lui-même ? Si c'est le cas, on choisit le code de remplacement et la position suivante examinée est la première à la suite du facteur recodé. Sinon, le caractère courant est encodé sur deux bits et le caractère suivant devient position courante.

La condition qui prévaut à la décision de substituer ou non un facteur par un pointeur ne prend pas en compte le fait que celui-ci peut être précédé par une suite de caractères. Rappelons qu'il y a alternance de suites de caractères et de pointeurs dans le codage. L'action de substituer la répétition oblige à auto-délimiter cette suite et ce coût n'est pas intégré dans la condition. En conséquence, cela ne permet pas d'être sûr que la substitution du facteur apporte un gain et donc que la séquence ne sera pas globalement étendue par l'encodage.

Pour la recherche de répétitions, Biocompress utilise un automate des facteurs ([BBE+85, CR94]) de taille bornée. On peut obtenir les positions d'apparition de tous les facteurs de taille inférieure à h (où h est fixé arbitrairement). Cela permet de trouver à quelles positions le facteur de longueur h qui débute à la position courante, apparaît dans le texte déjà codé. Le reste de la détection du plus long facteur est effectuée grâce à des comparaisons lettre à lettre, pour chaque position du préfixe de longueur h. Le même système est employé pour la recherche des palindromes.

Biocompress 2.

Il s'agit d'une deuxième version de Biocompress, où les suites de bases sont encodées grâce à un codage arithmétique avec un modèle à contexte fini d'ordre 2 (cf. sous-section 4.1.3), plutôt qu'en associant 2 bits à chaque base. Comme nous pourrons le voir dans le chapitre 6, les codages arithmétiques obtiennent presque toujours un certain gain sur une séquence (inférieur à 5% de réduction). Les résultats de Biocompress sont donc améliorés.

Chapitre 5

Algorithme de compression par codage de répétitions.

Nous avons indiqué en introduction de cette partie combien l'analyse des répétitions dans l'ADN est importante du point de vue biologique. Nous précisons qu'à l'heure actuelle, peu de méthodes permettent des comparaisons globales de séquences sur des critères tels que la répétitivité. Nous présentons une méthode répondant à ce besoin, basée sur un algorithme de compression.

Cet algorithme est dédié à la régularité structurale la plus simple et la plus naturelle : la répétition directe de facteurs dans un texte. Par opposition aux palindromes génétiques, nous la qualifions de directe pour signifier qu'un facteur est répété dans le sens de lecture du texte (comme B_1 et B_2 dans l'exemple de la prochaine section). Cette régularité est naturelle parce qu'elle apparaît dans de nombreux langages, de ce fait elle est à la base des algorithmes de compression classiques par codage à dictionnaire (pour un aperçu des différentes méthodes voir le chapitre 4 ou les ouvrages de référence [BCW90, Nel91]). L'idée primordiale est qu'une nouvelle occurrence d'un facteur est remplacée par un pointeur vers une occurrence précédente. Simplement grâce à ce procédé, il est concevable de construire autant d'algorithmes que de manières de choisir quels facteurs seront substitués, i.e. appartiendront au dictionnaire. Une occurrence d'un facteur, i.e. une portion du texte qui peut être remplacée est aussi appelée *zone*. De même, coder une occurrence est synonyme de la substituer, i.e. la remplacer par un code binaire.

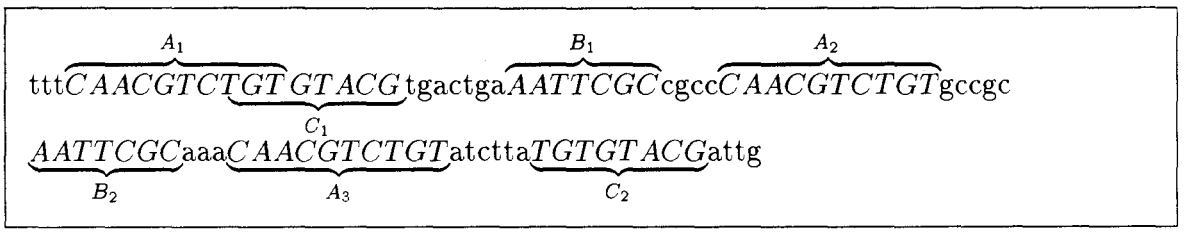
Comme nous le voyons dans les résultats présentés dans le chapitre 6, les algorithmes de compression à dictionnaire classiques donnent des taux de compression négatifs. Notre travail vise à produire un algorithme qui améliore ces taux **uniquement en codant des répétitions directes**. Le compresseur présenté ici est nommé *Cfact*. *Cfact* présente deux particularités : il sépare l'exécution de la factorisation et du codage, et se caractérise par **une garantie de compression**. Nous détaillons sa manière d'opérer par un exemple de décomposition du texte, puis nous en présentons le codage. Les particularités de la factorisation sont décrites pour ensuite discuter les choix de conception et les apports. La dernière section concerne l'algorithme similaire qui est dédié aux palindromes génétiques.

5.1 Exemple de compression.

Considérons l'exemple de la séquence s suivante :

tttcaacgtctgtgtacgtgactgaaattcgccgcccaacgtctgtgccgcaattcgcaaacaacgtctgtatcttatgt
gtacgattg

L'utilisation de l'arbre des suffixes de s et le calcul de *ListeFacteurs* permettent de savoir quels sont les grands facteurs répétés dans s (cf. section 3.3). Les facteurs sont nommés A, B, C et leurs occurrences sont indicées à partir de 1. Notons que C_1 chevauche A_1.

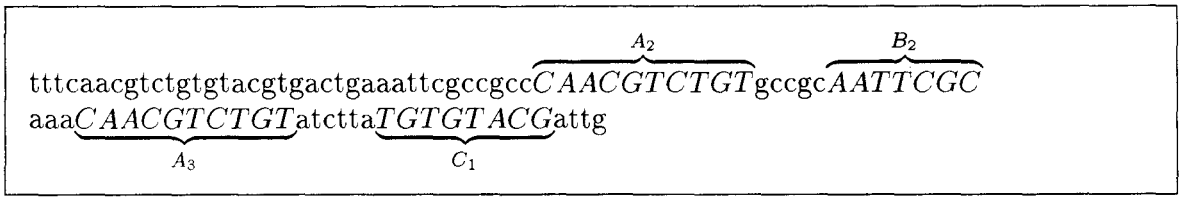


Les occurrences des facteurs répétés sont classées en deux groupes :

A_1, B_1, C_1 : les occurrences les plus à gauche dans s sont les occurrences de référence,

A_2, B_2, A_3, C_2 : sont choisies pour être substituées par des «pointeurs» qui renvoient sur les occurrences de référence du facteur correspondant.

Le choix des facteurs qui sont substitués, décompose s ainsi :



On remarque que les portions qui seront substituées durant le codage :

- représentent des facteurs qui apparaissent au moins une fois à gauche,
- ne se chevauchent pas entre elles,
- sont séparées par d'autres segments de la séquence qui eux, ne seront pas substitués.

Pour les désigner aisément, notons ces segments s_1, s_2, s_3, s_4, s_5 dans l'ordre où ils apparaissent dans la séquence. Alors on note ainsi la décomposition de s :

$$s = s_1.A_2.s_2.B_2.s_3.A_3.s_4.C_2.s_5$$

Le codage des substitutions est décrit dans la figure 5.1. Les flèches en trait plein désignent le codage de chaque substitution. Celles en trait discontinu montrent la construction du dictionnaire. Celui-ci nécessite d'être mémorisé dans une table seulement au décodage, car lors du codage de toutes les occurrences d'un facteur, l'index vient juste d'être déterminé. À chaque substitution de la deuxième occurrence, on associe les informations sur le facteur à un index dans le dictionnaire, pour encoder les occurrences suivantes. Le dictionnaire est une structure en mémoire construite par le décompresseur, mais n'est pas inclus dans le fichier compressé.

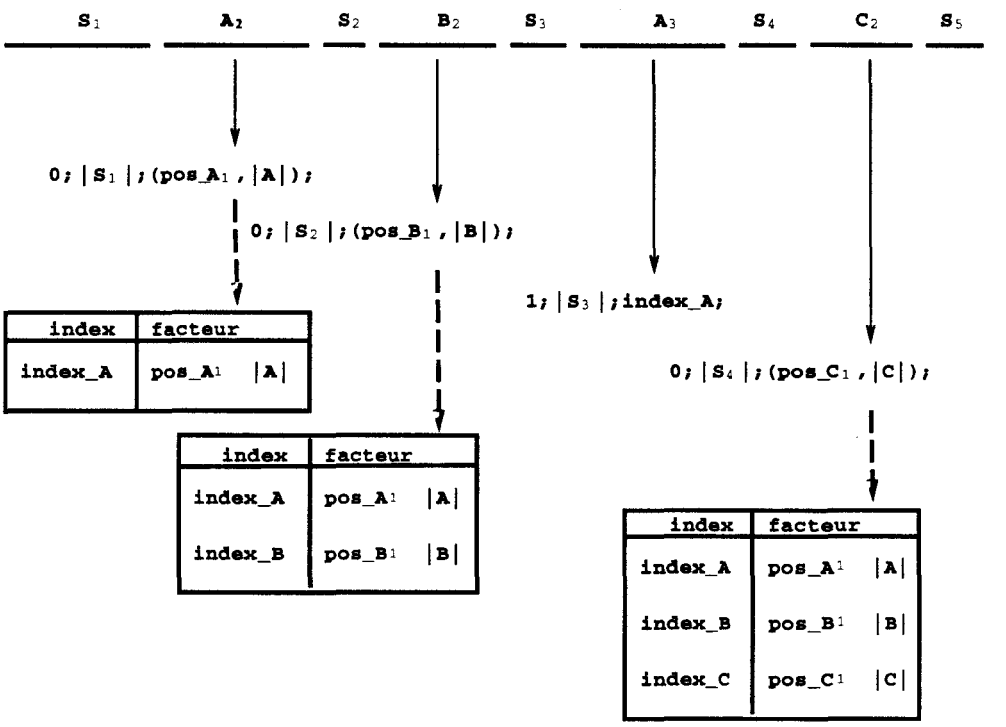


FIG. 5.1 - *Codage des occurrences substituées pour s. Les traits horizontaux montrent la décomposition de s.*

Les portions de séquence non substituées, sont encodées simplement en donnant la suite des codes binaires de chacune de leur base sur 2 bits. Nous notons ce codage *Base_Binaire*. De plus, la suite des codes de substitution est précédée par son nombre d'éléments pour permettre son décodage (plus exactement la séparation des items lors de sa lecture). Le code complet est écrit dans la table 5.1. Les items sont séparés par des «;» qui indiquent la possibilité de les identifier et les pointeurs de substitution sont entre parenthèses; différentes lignes sont utilisées pour la lisibilité (tous ces symboles supplémentaires ne font pas partie du codage réel) :

5.2 Codage du compresseur.

Nous exposons l'organisation globale du codage qui donne la structure du texte comprimé, puis le codage de chaque zone répétée. Nous indiquons pour cela les méthodes utilisées pour coder des entiers en binaire.

5.2.1 Organisation du codage.

Le codage se divise en deux parties distinctes. On trouve en premier **la liste des zones de structure régulière** précédée de son nombre d'éléments. Pour un compresseur une zone est régulière s'il peut l'encoder avantageusement (cf. le chapitre 1); pour *Cfact*, ce sont nécessairement les facteurs répétés maximaux (cf. définition au chapitre 2).

Code pour	**Valeur du code**		
nombre de zones codées	4;		
	Type de zone	**Position relative**	**Pointeur**
zone A_2	0;	$\|s_1\|$;	$(pos_{A_1}, \|A\|)$;
zone B_2	0;	$\|s_2\|$;	$(pos_{B_1}, \|B\|)$;
zone A_3	1;	$\|s_3\|$;	$(index_A)$;
zone C_2	0;	$\|s_4\|$;	$(pos_{C_1}, \|C\|)$;
séquence restante	$Base_B inaire(s_1.s_2.s_3.s_4.s_5)$		

TAB. 5.1 - *Code complet pour la séquence s.*

Cette liste contient la description codée des occurrences des facteurs choisis, i.e. ceux effectivement mis dans le dictionnaire. Les occurrences sont indicées de gauche à droite. La première apparition du facteur est appelée *occurrence de référence*, celle-ci est laissée intacte dans la séquence, de manière à ce qu'une occurrence plus à droite puisse être remplacée par un pointeur sur l'occurrence de référence. Toutes les autres occurrences du facteur, hormis celle de référence, sont examinées et éventuellement choisies pour être substituées. Parmi celles-ci on distingue deux types d'occurrence (pour un facteur qui a au moins 3 occurrences) :

La «deuxième» occurrence : la première apparition du facteur qui est codée par une référence *directe* vers l'occurrence de référence. Lors du codage de cette occurrence, le facteur est inséré dans le dictionnaire.

Les «ni-ème» occurrences : toutes les occurrences à la droite de la deuxième qui sont aussi codées, mais par une référence *indirecte*. Elles sont remplacées par l'index du facteur dans le dictionnaire.

Après la liste des zones, la deuxième partie du codage est la **séquence restante**. Toutes les zones de la séquence qui ne sont pas codées, sont concaténées au fur et à mesure du codage et forment la séquence restante. Celle-ci contient les portions non régulières de la séquence et toutes les occurrences de référence des facteurs codés. Ainsi lors du décodage, avec un pointeur, on peut retrouver la séquence des lettres d'un facteur du dictionnaire dans la séquence restante.

5.2.2 Codage des entiers.

Dans le codage de *Cfact*, nous traduisons certaines zones régulières d'un texte en un code binaire. Celui-ci est une suite d'items séparés qui permettent de reconstituer la zone lors de la décompression. Cette sous-section aborde la manière de coder ces items. Principalement, pour *Cfact* les items sont des nombres entiers. Pour interpréter correctement le code du texte comprimé, le décompresseur doit pouvoir isoler chaque item en lisant le code de gauche à droite. Cette contrainte empêche d'utiliser le codage binaire pour les entiers et oblige à auto-délimiter les codes utilisés.

Notation binaire des entiers.

Si l'on donne à un décompresseur une séquence sur $\{0, 1\}$ qui contient la concaténation d'une liste d'entiers codés sous cette forme, il est incapable d'y retrouver chaque entier. Par

exemple, la séquence suivante 10110 peut être interprétée en autres comme :

- 1, 0, 1, 1, 0 qui correspond à la liste : $1, 0, 1, 1, 0$
- 10, 1, 10 qui correspond à la liste : $2, 1, 2$
- 10, 110 qui correspond à la liste : $2, 6$
- 10110 qui correspond à la liste à un seul entier : 22

On ne peut donc pas utiliser le code binaire sans savoir par un autre moyen, avant de procéder à la lecture, quel est la longueur du code d'un item. Le remède tient dans l'emploi de codes *auto-délimités*. Un code est auto-délimité si lorsqu'on connaît le début du code dans la séquence binaire à décomposer, on est capable d'en trouver la fin (évidement le code est écrit de manière contiguë). L'usage d'un code auto-délimité coûte plus cher que l'utilisation d'un code binaire, au sens propre du terme, puisqu'un code auto-délimité est plus long que le code binaire. On troque une propriété du code indispensable contre un surcoût en longueur.

Plusieurs techniques permettent d'auto-délimiter un code. On note n l'entier à coder.

1. Utiliser des codes de taille fixée à l'avance, mais cela limite l'ensemble des entiers encodables (nous voulons éviter ce genre de limitation pour *Cfact*, elle est présente dans LZ77).

2. Employer un code en deux parties, où l'item est précédé d'un sous-item qui indique la longueur de son écriture binaire et le code binaire est utilisé pour l'entier lui-même. On reporte le problème sur le sous-item qui doit aussi être auto-délimité. Cependant, l'écriture de la longueur d'un entier est plus courte (en $O(\log(\log(n)))$ au lieu de $O(\log(n))$) et le prix à payer pour l'auto-délimiter est moins cher que l'auto-délimitation directe.

3. On peut itérer la technique précédente et employer récursivement cette auto-délimitation indirecte : il suffit de faire précéder la longueur de l'entier par sa propre longueur. Tous les niveaux d'itération sont théoriquement intéressants et l'on sait qu'asymptotiquement la longueur du code ainsi construit, se rapproche de $\log(n)$ (i.e. de l'écriture binaire).

4. Utiliser des codes qui sont algébriquement construits pour être auto-délimités. Dans cette catégorie se place le code de Fibonacci qui fut étudié dans [AF85].

Les *auto-délimitations indirectes*, où un code est précédé de sa longueur, avec différents niveaux d'itération, sont présentées et comparées dans l'annexe intitulée «Variable-length Representations of the Integers» de [BCW90].

Code de Fibonacci pour les entiers.

Le code de Fibonacci utilise les nombres de la suite de Fibonacci comme base pour l'écriture des entiers. Le terme de la suite implique que jamais deux symboles binaires 1 ne se suivent dans l'écriture de n'importe quel entier. En outre, quelle que soit la base, l'écriture d'un entier débute par un «1». Pour obtenir un code auto-délimité, il suffit de renverser l'écriture dans la base de Fibonacci et d'y concaténer un «1» supplémentaire. La table 5.2 montre les codages de quelques entiers.

Notation 3 L'écriture du code Fibonacci d'un item entier est notée $Fibo(item)$.

Entier	Écriture Fibonacci							Code de Fibonacci
	21	13	8	5	3	2	1	
1							1	11
2						1	0	011
3					1	0	0	0011
4					1	0	1	1011
5				1	0	0	0	00011
6				1	0	0	1	10011
8			1	0	0	0	0	000011
16		1	0	0	1	0	0	0010011
32	1	0	1	0	1	0	0	00101011

TAB. 5.2 - *Exemple de codage des entiers selon la méthode de Fibonacci. La deuxième colonne donne l'écriture de gauche à droite dans la base de Fibonacci. Chaque symbole est mis sous le nombre de Fibonacci correspondant.*

Ce codage est présenté dans l'article [LH87]. On y trouve la précision qu'il est asymptotiquement optimal dans le sens de la définition 1.13 page 71 dans [LV93]. L'important est de choisir la méthode de codage qui est appropriée à la distribution de probabilités des entiers à coder. Un problème survient lorsque l'on désire encoder l'entier «0» par la méthode de Fibonacci. La solution coûteuse mais simple est de décaler les codes, en codant l'item plus 1.

5.2.3 Description complète du code d'une zone.

Le code de chaque zone contient :

1. un bit de différenciation pour distinguer les deux types d'occurrence, «deuxième» ou «énième»,
2. une position relative dans la séquence par rapport à la fin de la zone précédente,
3. le pointeur de référence qui permet de retrouver la séquence du facteur.

On note P la position relative d'une zone, OR la position d'apparition de l'occurrence de référence, I l'index du facteur dans le dictionnaire, L la longueur du facteur. $TailleMin$ est la taille minimale requise pour qu'un facteur puisse engendrer un gain de compression. Le codage complet d'une occurrence est détaillé dans la table 5.3.

	Type	Bit	Position relative	Pointeur de référence
Code	2éme	0	$Fibo(P+1)$	$Fibo(OR+1).Fibo(L)$
	énième	1	$Fibo(P+1)$	$Fibo(I)$
Longueur du code	2éme	0	$\|Fibo(P+1)\|$	$\|Fibo(OR+1)\| + \|Fibo(L)\|$
	énième	1	$\|Fibo(P+1)\|$	$\|Fibo(I)\|$

TAB. 5.3 - *Détail du codage d'une occurrence d'un facteur répété.*

Codage d'une 2ème occurrence.

Dans le cas d'une deuxième occurrence, le pointeur est direct. Il contient deux items:

- la position dans la séquence de l'occurrence de référence, celle-ci est déjà codée du point de vue du compresseur et donc déjà décodée du point de vue du décompresseur;
- la longueur du facteur répété dont on substitue les occurrences.

Codage d'une énième occurrence.

Le codage est similaire à celui d'une deuxième occurrence, sauf que le couple d'entiers du pointeur de référence est remplacé par l'index du facteur dans le dictionnaire adaptatif. Cet index est codé par Fibonacci.

5.3 Factorisation.

Une originalité de *Cfact* réside dans la séparation des deux tâches accomplies par un compresseur à dictionnaire : *la factorisation et l'encodage du texte.* La factorisation consiste à déterminer quels facteurs du texte vont être remplacés par un pointeur, tandis que l'encodage constitue la version compressée du texte en substituant un code binaire aux zones choisies par la factorisation. L'exécution séquentielle plutôt que simultanée de ces deux tâches, permet d'utiliser de meilleurs critères de sélection des facteurs et évite des limitations dues à la simultanéïté. Une conséquence est le théorème qui garantit la compression d'une séquence contenant une répétition longue. Nous détaillons ces apports dans la section 5.4.
La factorisation sélectionne les occurrences de facteurs répétés pour qu'elles soient codées. Les principaux critères de sélection sont:

- le fait d'être un facteur maximal, i.e. d'appartenir à *ListeFacteurs*,
- l'examen dans l'ordre des tailles décroissantes,
- l'évaluation du gain potentiel engendré par le codage d'une occurrence.

La procédure de factorisation constitue *la Liste des Zones Codées* ou **LZC** qui sera son résultat final. Elle sera passée en paramètre à la procédure d'encodage. À chaque occurrence sélectionnée, elle crée un élément dans la LZC.

Nous détaillons consécutivement la sélection des facteurs, la structure de la LZC, puis la manière dont le gain d'une occurrence est évalué.

5.3.1 Schéma de la factorisation.

La factorisation est un examen de tous les facteurs maximaux du texte. Afin de les obtenir, elle construit l'arbre des suffixes du texte t, et appelle la procédure *TousFacteurs*, définie à la section 3.3, qui lui renvoie *ListeFacteurs*: la liste chaînée des facteurs maximaux. Cette liste est ordonnée selon la taille décroissante des facteurs. L'algorithme consiste en un parcours de ListeFacteurs. Pour chaque facteur, on parcourt simultanément 1/ la liste des positions de ses occurrences pour envisager leur substitution et 2/ la LZC. L'élément courant de LZC est toujours le plus proche segment dans la séquence, déjà sélectionné, par rapport à l'occurrence courante. L'apparition la plus à gauche du facteur devient l'occurrence de référence. Ensuite,

pour toutes les autres occurrences, on regarde si elles ne chevauchent pas une zone déjà codée grâce à un parcours simultané de la liste des zones codées ; sinon on passe à la suivante. Dans ce cas, on évalue le gain propre que l'on obtiendrait par le codage de l'occurrence, s'il est positif, on prend la décision de la substituer. Cette zone est alors insérée dans la LZC selon sa position dans le texte. Si le gain est négatif ou nul, on passe à la zone suivante.

Notons que la factorisation est gloutonne, puisqu'elle élabore sa solution par étape, sans jamais remettre en cause le choix d'un facteur inclus dans la LZC.

5.3.2 Structure de données de la liste des zones codées.

Nous montrons la structure de la LZC parce qu'elle aide à comprendre comment la description d'un segment à encoder est mémorisée pour la phase d'encodage. La structure, nommée Zone, d'une cellule de la liste chaînée est détaillée dans la table 5.4.

Nom	Type	Sémantique
Deb	entier	début de la Zone dans le texte
Fin	entier	fin de la Zone dans le texte
OccPos	entier	position absolue de la première occurrence du facteur
Index	entier	index du facteur dans le dictionnaire
suiv	ptr. Zone	cellule suivante de la liste

TAB. 5.4 - *Détail de la structure d'une cellule de la Liste des Zones Codées.*

Les deux premiers champs donnent la longueur de l'occurrence et sa position, cela permet de calculer pendant le codage, la position relative par rapport à la zone précédente. Selon le type de la zone, *OccPos* pour les «deuxièmes» occurences ou *Index* pour les «énièmes», permettent d'encoder le pointeur. *suiv* pointe vers l'élément suivant de la LZC. En outre, à chaque pas la LZC est utilisée pour :

- examiner les chevauchements de l'occurrence courante avec les zones déjà codées, chevauchements qui rendraient inutile sa substitution,
- évaluer le gain du codage de l'occurrence.

Cette liste est ordonnée par ordre d'apparition dans le texte et son examen coûte au pire $O(n)$ pas.

Évaluation du gain du codage d'une occurrence.

Il s'agit de la deuxième caractéristique originale de *Cfact*. On mesure le gain local engendré par le codage d'une occurrence. **Il est toujours sous-évalué.** Ainsi avant de sélectionner un segment pour qu'il soit encodé dans la deuxième phase on sait s'il permet d'augmenter le gain global de la séquence.

Le calcul du gain compare la taille du segment avec la longueur de son code binaire de remplacement. La longueur est donné par la longueur du facteur. Pour le code de remplacement, le calcul somme la longueur des items selon le type de la zone. Ces longueurs sont résumées dans la table 5.3. Le seul élément approximé dans cette évaluation préalable du gain du codage d'une occurrence est la position relative de la zone, puisque la LZC n'est pas dans son état final. Cette position est surévaluée car d'autres zones seront peut-être intercalées

dans la liste entre la zone précédente et la zone dont on évalue le gain. Cet approximation sous-estime le gain.
Remarque : pour les premiers facteurs examinés le gain est fortement sous-estimé puisque la LZC n'est pas encore pleine, mais leur taille entraine à elle-seule leur élection. Par contre, l'évaluation des derniers facteurs est meilleure, car la LZC est proche de son état final. Ce sont donc pour les plus petits facteurs, dont l'éléction est plus difficile, que nous avons la meilleure évaluation.

5.4 Complexité, garantie de compression et apports.

Cet algorithme de compression se différencie nettement des algorithmes à codage par dictionnaire adaptatif et de Biocompress, soit grâce à ses critères de factorisation, soit par son codage. Bien qu'elle entraine une complexité en $O(n^2)$, la factorisation donne à *Cfact* une propriété qui n'existe à notre connaissance, chez aucun autre algorithme à dictionnaire : la garantie de compresser la séquence. En effet, nous prouvons un théorème qui garantit la compression pour une séquence incluant au moins une répétition «longue», où la condition de longueur est calculable.
Pour réaliser notre objectif d'améliorer les taux de compression en n'envisageant que les répétitions directes comme régularités, deux possibilités s'offrent à nous :

1. Optimiser le codage : mais celui-ci est très contraint car les informations qu'il contient doivent y être, et le codage d'items tels que les entiers sont déjà bien étudiés.

2. Effectuer une meilleure factorisation : ce qui à notre avis, est le champ de réflexion le plus ouvert aux améliorations, puisque les textes usuels n'exigent pas des factorisations complexes pour être comprimés.

5.4.1 Complexité de l'algorithme.

Propriété 16 Les complexités en temps et en espace de *Cfact* sont en $O(n^2)$.

Preuve On note n la taille du texte à compresser. Le codage s'effectuant par un parcours de la LZC et en une passe sur le texte, sa complexité est linéaire par rapport à n. La complexité de *Cfact* provient de la procédure de factorisation et plus particulièrement de la boucle qui balaye ListeFacteurs. En effet, l'arbre des suffixes et ListeFacteurs sont construits séparément en $O(n^2)$, d'après les propriétés 10 et 14. La boucle principale examine ListeFacteurs et pour chaque facteur parcourt en même temps :

- la liste des positions de taille inférieure à n,

- la LZC, elle aussi de taille inférieure à n.

D'après la propriété 13 qui prouve qu'il y a au plus n facteurs maximaux, on obtient une complexité temporelle en $O(n^2)$. La plus grande structure en mémoire est ListeFacteurs dont la taille mémoire est en $O(n^2)$. □

5.4.2 Garantie de compression.

Nous prouvons un théorème qui garantit la compression pour une séquence incluant au moins une répétition «longue». Ceci représente un avantage pour l'utilisation de *Cfact* comme outil de classification. À notre connaissance, aucun autre schéma de compression à dictionnaire, i.e. les plus utilisés pour les compression de textes, n'offre cette garantie.

Théorème 1 Si le plus long facteur répété, noté f, dans le texte t est de longueur l et apparaît au moins deux fois aux positions i, j telles que $i + l < j$, alors la condition suivante est suffisante pour que le gain de compression sur t soit positif :

$$2l - |Fibo(l)| \quad > \quad 3|Fibo(n)| + 1 \tag{5.1}$$

Preuve L'équation 5.1 est une sur-évaluation de la longueur minimale de la plus longue répétition, tel que son gain local soit positif. Cette formule peut être rendue plus précise si l'on remplace les longueurs de codes de Fibonacci, à propos desquels nous savons que, pour un item i :

$$\left\lceil \frac{\log(i\sqrt{5}-1)}{\log(\frac{1+\sqrt{5}}{2})} \right\rceil \quad \leq \quad |Fibo(i)| - 1 \quad \leq \quad \left\lceil \frac{\log(1+i\sqrt{5})}{\log(\frac{1+\sqrt{5}}{2})} \right\rceil$$

L'équation 5.1 contraint la répétition à être suffisament longue pour engendrer un gain plus grand que l'écriture du nombre de zones encodées (ce nombre est 4 dans notre exemple, voir la table 5.1).

f est nécessairement un facteur maximal et donc mémorisé dans le premier élément de *ListeFacteurs*. Au début de la factorisation, la LZC est vide, donc l'occurrence de f en j ne chevauche aucune zone déjà sélectionnée. Lorsque son gain potentiel est sous-évalué, l'équation 5.1 force le résultat à être positif. Alors on crée un élément correspondant dans la LZC, et quelles que soient les zones sélectionnées ensuite, leur gain est positif. Comme le gain global sur t est la somme des gains locaux positifs, il est lui-même positif. □

5.4.3 Optimisation du codage.

L'amélioration du codage par rapport aux méthodes usuelles est triple :

1. **Seules les zones régulières sont véritablement encodées** (du moins, celles qui sont vues telles par *Cfact*). L'algorithme n'est pas capable de déceler des régularités dans les autres zones, de son point de vue elles sont «aléatoires». Il n'est donc pas judicieux d'essayer de leur attribuer un code binaire qui transforme leur représentation. En effet, de manière absolue la complexité de Kolmogorov nous dit que leur plus courte représentation est leur séquence elle-même. Biocompress intègre cette idée mais pas les compresseurs de Lempel-Ziv (versions 77 ou 78). En effet, dans les méthodes de Lempel-Ziv, toute partie du texte est codée comme une répétition, i.e. une zone régulière. Or, avec un alphabet fini, il y a toujours des répétitions courtes, même dans une séquence aléatoire (pourvu qu'elle soit assez longue).

2. **Absence de restriction sur la taille et la position des facteurs référencés** : un pointeur peut référencer n'importe quel facteur situé à gauche dans le texte, cela implique que les entiers «position» et «longueur» de l'occurrence soient codés de manière

auto-délimitée. Ceci se justifie par le fait que la structure des répétitions est différente dans les textes génétiques par rapport à ceux en langages naturels. En effet, les répétitions exactes d'ADN sont parfois de milliers de bases et ont des occurrences très distantes. Biocompress utilise aussi l'autodélimitation. LZ78 évite la limitation en la reportant sur la taille du dictionnaire qui est limitée.

3. **Le codage admet des pointeurs à référence directe** (de type (*position*, *longueur*)) **et indirecte** (index dans un dictionnaire) pour économiser des codages auto-délimités de la localisation de l'occurrence de référence. LZ78 n'utilise que des références indirectes. Dans Biocompress, le pointeur d'une «énième» occurrence est toujours de la forme (*position*, *longueur*). Or, à une position ultérieure, on a déjà donné la longueur et la position de l'occurrence référencée, items qui sont arbitrairement longs. Au contraire, nous utilisons pour seul item, **un index** dans un dictionnaire qui permet de retrouver ces informations. Notons que cela coûte un bit de différenciation par zone.

5.4.4 Optimisation de la factorisation.

Usuellement les factorisations employées sont gloutonnes (greedy) ou semi-gloutonnes. C'est le cas des méthodes de Lempel-Ziv et de Biocompress. Cependant, le critère de sélection des mots pour la décomposition du texte est la longueur de la décomposition (en nombre de mots) ou la taille des mots. Pour ce critère la factorisation semi-gloutonne donne une décomposition optimale (voir [CR94]). **Mais la taille des mots n'est pas le gain.** La factorisation de *Cfact* est au sens propre gloutonne, mais tient compte de deux critères: la taille et le gain potentiel. Le résultat est une garantie de compression. Notons que les seuls algorithmes naturels pour effectuer une factorisation optimale, suivant le critère du gain global, sont exponentiels.

Dans *Cfact*, la factorisation est séparée de l'encodage et utilise *ListeFacteurs*. Il existe plusieurs avantages à cette manière de faire:

1. **Un facteur maximal est obligatoirement répété dans le texte**, et donc tous les facteurs examinés offrent des possibilités de substitution. LZ78 inclut dans son dictionnaire des facteurs dont l'index ne sera jamais utilisé dans le fichier compressé. Pour l'éviter il faudrait qu'il cherche à déterminer en avance si le facteur est répété ou non.

2. **La décomposition ne subit pas la contrainte de devoir être faite de gauche à droite.** Cette contrainte provient de la simultanéïté de la factorisation et du codage, dans les algorithmes usuels. Biocompress, LZ77 et LZ78 la subissent pleinement. Cette conception est avantageuse pour la complexité, mais empêche clairement un choix optimal. Empiriquement, les essais rencontrent peu de cas où cette limitation influe nettement sur le taux de compression.

3. Étant donné que la fonction du gain d'une zone croît linéairement avec sa taille, nous puisons les facteurs dans **ListeFacteurs qui est classée par ordre de taille décroissante.** On donne ainsi priorité au choix des facteurs dont le gain potentiel est le plus grand. Il est optimal pour une étape donnée, puisque les facteurs plus longs ont déjà été soit codés, soit examinés. Bien sûr, globalement le chevauchement des zones peut empêcher d'atteindre l'optimum global.

5.5 Algorithme de compression par codage de palindromes.

Une autre régularité importante dans les séquences d'acides nucléiques est due aux palindromes génétiques (pour un exemple voir la figure 1.1). La complémentarité des bases dans les ADN ou les ARN permet à des segments distants de la même séquence, de s'apparier, lorsqu'elles sont au voisinnage l'une de l'autre dans l'espace. Les deux segments doivent être renversés dans la séquence et complémentaires. D'où le nom de palindrome génétique. Du point de vue du texte, un palindrome génétique est un couple de facteurs (f, g) où g (respectivement f) s'obtient à partir de f (respectivement g), non plus par une duplication comme pour les facteurs maximaux, mais par une symétrie. En fait, g équivaut à f écrit à l'envers où chaque base est changée en sa base complémentaire. Ce type de régularité convient donc parfaitement à un schéma de compression substitutionnel.

Nous avons montré dans la section 3.4, qu'une utilisation de l'arbre des suffixes d'un texte permet d'obtenir le plus grand palindrome commençant à chaque position dans la séquence. Grâce à *ChercherPalindrome2*, cette recherche est résolue en temps linéaire par rapport à la séquence. Cet algorithme produit la liste des palindromes classés par ordre de longueur décroissante: *ListePalindromes*. Nous avons donc construit un schéma de compression similaire à *Cfact*, où les régularités utilisées sont les palindromes génétiques de *ListePalindromes* plutôt que les facteurs de *ListeFacteurs*. Ce schéma de compression se nomme **Cpal**.

Grâce à *ListePalindromes* notamment, *Cpal* est quasiment identique à *Cfact*, sauf pour quelques points que nous allons préciser ici.

5.5.1 Constitution de *ListePalindromes*.

Une différence majeure entre les palindromes génétiques et les facteurs répétés est à l'origine des différences entre *Cpal* et *Cfact*. Un facteur peut être répété n fois le long de la séquence, tandis qu'un palindrome est constitué par un couple de facteurs. Ainsi, un élément de *ListePalindromes* ne contient pas une liste de positions d'occurrences, de taille variable, mais seulement deux indices, un pour chaque facteur du couple.

Comme la liste des facteurs maximaux, *ListePalindromes* est ordonnée par taille décroissante des palindromes, mais sa complexité est linéaire à cause du nombre borné de positions à mémoriser. Le pré-traitement du texte qui permet d'obtenir *ListePalindromes* l'est aussi.

5.5.2 Codage simplifié.

Lorsqu'on trouve un palindrome génétique dans la séquence, seul un facteur du couple peut être substitué par un pointeur référençant l'autre. Donc, seuls des pointeurs à référence directe sont utiles dans le codage de *Cpal*. La notion de type des zones de *Cfact*, qui distinguait la «deuxième» et les «énièmes» apparitions d'un facteur, n'ont plus de sens dans *Cpal*. Ceci simplifie le codage sur trois points:

1. les pointeurs sont des références directes,
2. il n'y a plus besoin de bit de différenciation car il n'y a qu'un seul type de zones codées,
3. la mise en place d'un dictionnaire spécifique au codage des «énièmes» apparitions d'un facteur devient inutile.

5.5.3 Combinaison de plusieurs régularités : palindromes et répétitions.

La détection et le codage des palindromes étant au point, une poursuite du travail consiste à combiner les schémas de *Cpal* et de *Cfact*, pour créer un compresseur plus «intelligent» capable d'identifier deux types de régularités. Ce nouveau schéma, nous l'appelons **Cfactpal** se calquerait sur *Cfact*, mais lors de la factorisation consulterait simultanément *ListePalindromes* et *ListeFacteurs* pour y choisir la zone la plus avantageuse a priori (i.e. la plus longue). La décision de son éventuelle substitution prendrait aussi en compte une évaluation de son gain potentiel. Le codage serait à peine modifié, puisqu'il suffirait de distinguer 3 types de zones : les deux de *Cfact* et une de *Cpal*. Leur encodage resterait le même mais serait précédé de deux bits de différenciation.

Les résultats de *Cfactpal* et de Biocompress seraient directement comparables (voir la section 4.3). En outre, si la séquence restante est ensuite encodée par une méthode arithmétique adéquate, le schéma pourra être comparé à Biocompress-2 (voir la discussion du chapitre 6).

Chapitre 6

Comparatif des compresseurs.

Ce dernier chapitre est consacré à la validation de l'algorithme *Cfact*, tout autant qu'à l'étude du comportement de divers compresseurs sur des séquences génétiques. D'une part, nous voulons valider empiriquement les apports de *Cfact*. D'autre part, nous désirons comparer *Cfact* à d'autres compresseurs et connaître le comportement de ces derniers face à des données biologiques. Les expériences de comparaison de compresseurs réalisées sur des séquences d'ADN ont 4 objectifs :

1. Connaître le comportement des schémas de compression classiques sur des séquences génétiques, grâce à nos propres tests. Il n'y a pas en effet, de publications reprenant autant de résultats commentés.

2. Identifier les possibilités de compression dues au biais d'utilisation des mono-, di-, tri- et quadrinucléotides dans l'ADN.

3. Comparer le comportement de deux versions de *Cfact* vis à vis des schémas classiques ; principalement pour savoir si la mise en œuvre d'un algorithme de complexité en $O(n^2)$ est intéressante pour la compression de séquences génétiques?

4. Valider empiriquement le théorème de garantie de compression de *Cfact*.

Nous présentons les différents schémas de compression utilisés, puis le matériel biologique conjointement aux tableaux récapitulatifs. Une dernière section se consacre aux commentaires de ces résultats.

6.1 Compresseurs mis en compétition.

Cette courte section indique la nature de tous les schémas de compression (ou compresseurs) utilisés dans nos tests. Elle est à lire en connaissance des principales méthodes du domaine. Le chapitre 4 et des ouvrages de référence tels que [BCW90, Hel91, Nel91, Sto88] en présentent les idées.

Nous employons trois types de compresseurs : les classiques à dictionnaire fonctionnant selon les deux méthodes de Lempel-Ziv (5 compresseurs), les classiques statistiques (4 compresseurs) et deux versions de *Cfact*. Nous donnons une définition pour chaque, ainsi que le nom abrégé qui apparaît dans les tableaux. Les implémentations des méthodes classiques sont issues de [Nel91], moyennant parfois quelques modifications. Nous avons contacté les

auteurs de [GT93b] pour comparer avec leur algorithme Biocompress, mais celui-ci n'est pas disponible actuellement.

Les compresseurs classiques à dictionnaire.

lzss : schéma basé sur la méthode LZ77 où le pointeur ne contient pas d'item de type caractère, il provient de [Bel86]. Le pointeur a une taille fixe et on vérifie que le mot à coder est plus long que le pointeur.

lzw1 : comme tous les autres noms commençant par «lzw», il s'agit de la méthode LZ78 modifiée par Welch dans [Wel84]. Elles varient principalement par la taille maximale de l'index en bits. Cet index référence une entrée dans le dictionnaire. Cette taille limite donc le nombre de mots de ce dernier. Pour lzw1, elle vaut 12 bits.

lzw2 : méthode LZW avec la taille maximale de l'index à 15 bits. L'alphabet compris par lzw1 et lzw2 est le code ASCII.

lzw3 : identique à lzw2, mais elle connaît l'alphabet des séquences génétiques limité à 4 lettres.

lzw4 : identique à lzw3 mais avec la taille maximale de l'index à 20 bits.

Les compresseurs classiques statistiques.

Toutes ces méthodes acceptent des textes sur l'alphabet du code ASCII.

huff : codage de Huffman utilisé avec un modèle statique ad hoc. L'arbre des codes est donc intégré au fichier compressé.

arith0 : méthode de codage arithmétique utilisée avec un modèle adaptatif à contexte fini, aussi appelé markovien (cf. la sous-section 4.1.3 dans le chapitre 4). Toutes les méthodes suivantes ne différent que par l'ordre du modèle, i.e. la taille du contexte pris en compte dans les statistiques sur le texte. La probabilité d'un symbole dépend des k symboles précédents, où k est l'ordre du modèle. Ici le modèle est d'ordre 0, il calcule la probabilité d'apparition des symboles seuls.

arith1, arith2, arith3 : idem avec respectivement des modèles d'ordre 1, 2 ou 3.

Deux versions de *Cfact*.

cf2 : implante *Cfact* tel que décrit dans le chapitre 5,

cf4 : diffère de *Cfact* par l'utilisation exclusive de pointeurs directs pour substituer aux répétitions, i.e. de la forme (position, longueur). La structure mémoire de dictionnaire et les différents types de zones codées disparaissent, à l'instar du compresseur pour les palindromes génétiques (voir section 5.5).

6.2 Matériel biologique et récapitulatif des résultats.

Nous décrivons le matériel biologique employé: la provenance et la nature des séquences, avant de montrer quelques tableaux de résultats. L'ensemble des tableaux de résultats sont disponibles par FTP anonyme à *ftp.lifl.fr:/pub/BIC/biologie/Cfact*. Chaque tableau concerne les expériences sur les séquences d'une seule espèce.

6.2.1 Matériel biologique.

Les expériences portent sur des séquences d'ADN de différents organismes : Escherichia coli (bactérie à *gram*−), Bacillus subtilis (bactérie à *gram*+), Saccharomyces cerevisiae (levure), Arabidopsis thaliana (plante), Caenorabditis elegans (vers), Drosophila melanogaster (mouche du vinaigre), Gallus domesticus (poule), Rattus norvegicus (rat), Mus musculus (souris), Xenopus laevis (crapaud), Homo sapiens (vous et moi). Une expérience est appliquée aux séquences de génomes mitochondriaux. Ces espèces ont été choisies, soit pour leur qualité de «génome modèle» (E. coli, B. subtilis, S. cerevisiae, A. thaliana, H. sapiens), soit parce qu'elles sont largement étudiées et qu'une quantité importante d'ADN est disponible dans les banques.
Les critères de sélection des séquences sont:

- le fait d'être une séquence d'ADN propre et de ne pas provenir du séquençage de c-DNA,
- la «représentativité» que nous avons associé à la longueur. Certains schémas de compression sont asymptotiquement optimaux, ils se comportent d'autant mieux que les séquences sont longues. D'autre part, plus une séquence est longue, plus elle s'apparente au véritable objet biologique qu'est le chromosome.

Leur taille va de 5000 pb à 250000 pb. Nous avons expérimenté sur un total de 5,362,207 pb, divisé en 244 séquences de longueur moyenne égale à 21,976 pb. L'utilisation d'ACNUC sur le serveur génome de l'école Polytechnique à Villejuif, a permis leur extraction de Genbank ([GMM+84]). À noter que certaines très grandes séquences d'E. coli sont préalablement découpées en fenêtres consécutives de 50000 pb à cause de la complexité de l'algorithme.

6.2.2 Récapitulatifs des résultats.

Nous avons réalisé la compression des séquences extraites pour toutes les espèces citées plus haut. Un tableau est associé à chaque espèce. Nous montrons uniquement ceux d'E. coli, de S. cerevisiae, d'A. thaliana et de l'homme, dans les tables 6.1, 6.2, 6.3. Nous donnons d'abord une description du contenu d'un tableau et une explication sur le taux de compression.

Organisation des résultats.

Ils prennent la structure d'un tableau ayant en colonne 1 et 2, le nom et la taille des séquences, puis dans les 12 colonnes suivantes, le taux de compression réalisé par le schéma dont le nom est en début de colonne. Une ligne y est associée à une séquence. Les trois dernières colonnes donnent respectivement: le taux maximal de la ligne, le nom du compresseur qui l'a réalisé, le nom de celui qui détient la moins bonne performance. Les trois dernières lignes récapitulent les taux maximal, moyen et minimal de la colonne.

Taux de compression.

Il est donné en pourcentage de réduction de la taille de la séquence. Un taux de 20% signifie que la taille de la version comprimée vaut 80% de la taille originale. En outre, le taux s'affranchit de la traduction triviale de chaque base en un code de 2 bits. Dans le calcul du taux, la taille en bits de la séquence comprimée est comparée à la taille en bits de la séquence originale : les unités sont bien identiques. Pour obtenir la taille en bits de cette dernière, il faut la traduire en remplaçant chaque base par un code sur 2 bits. La taille de la séquence ainsi traduite vaut :

- la taille originale en base multipliée par 2
- la taille en bits du fichier divisée par 4.

Exemple
Ainsi, si une séquence contient 80 pb et que la version comprimée mesure 160 bits, le taux de compression vaut :

$$taux = \frac{(80 * 2 - 160)}{80 * 2} = 0$$

En outre, le taux peut être négatif si une méthode augmente la taille de la séquence en essayant de la comprimer (le terme comprimer est à double tranchant, nous l'utilisons dans le sens de tenter de comprimer).

6.3 Discussion : compression de séquences génétiques par des schémas substitutionnels et statistiques.

Les résultats sont commentés de deux points de vue, celui de la technique de compression : les schémas en compétition sont-ils adaptés pour comprimer des séquences génétiques? Enfin, celui plus biologique, des possibilités d'analyses de séquences grâce à la compression.

6.3.1 Aptitude des différents schémas pour la compression de séquences.

Le constat le plus étonnant est la quasi-incompressiblité des séquences génétiques pour les méthodes considérées. En effet, les taux de compression affichés sont faibles, en moyenne inférieurs à 1%, et ne ressemblent pas à ceux des utilisations courantes de la compression qui sont eux très forts. 50% de compression pour un texte en langage naturel n'a rien de surprenant. Puisque la compression est synonyme de compréhension, cela permet de savoir combien la structure de ces séquences nous est encore inconnue ! Mais regardons plus en détail.

Les schémas Lempel-Ziv.

- Les taux moyens réalisés par ces compresseurs sont très fortement négatifs, même les taux maximaux sont tous négatifs sauf pour une seule séquence (YSCMTCG pour la levure). **On voit par là combien ces compresseurs se révèlent inaptes à comprimer des séquences génétiques.** Ceci est d'autant plus surprenant que LZ78 est le paradigme des compresseurs les plus utilisés pour les autres types de textes.

Séquence	Taille	cf2	cf4	lzss	lzw1	lzw2	lzw3	lzw4	huff	ari0	ari1	ari2	ari3	Max	MaxC	MinC
ec2.1	50000	0.00	0.00	-40.24	-13.10	-13.54	-11.42	-11.42	-12.06	0.05	0.96	1.58	1.26	1.58	ari2	lzss
ec2.2	50000	0.17	0.16	-40.61	-12.99	-13.68	-11.30	-11.30	-11.46	0.12	1.06	1.86	1.35	1.86	ari2	lzss
ec2.3	50000	0.04	0.03	-40.96	-13.14	-13.97	-11.46	-11.46	-12.06	-0.06	0.74	1.58	1.25	1.58	ari2	lzss
ec2.4	43964	0.00	0.00	-41.51	-13.43	-14.83	-11.52	-11.52	-12.16	0.15	0.96	1.54	0.92	1.54	ari2	lzss
ec621	40288	0.93	0.93	-40.63	-13.42	-15.32	-11.65	-11.65	-11.89	0.15	1.03	1.66	1.14	1.66	ari2	lzss
ec635.1	50000	0.01	0.00	-39.66	-13.08	-13.90	-11.39	-11.39	-11.94	0.61	1.27	1.92	1.51	1.92	ari2	lzss
ec635.2	50000	0.08	0.06	-41.74	-13.79	-14.18	-12.10	-12.10	-12.17	0.02	0.68	1.18	0.79	1.18	ari2	lzss
ec635.3	50000	0.16	0.16	-41.23	-13.31	-13.95	-11.62	-11.62	-12.26	0.04	0.81	1.50	1.13	1.50	ari2	lzss
ec635.4	50000	0.80	0.79	-40.57	-13.58	-14.26	-11.90	-11.90	-12.06	0.11	0.80	1.34	0.86	1.34	ari2	lzss
ec635.5	50000	0.00	0.00	-41.34	-13.64	-14.71	-11.94	-11.94	-11.87	0.43	1.16	1.36	0.77	1.36	ari2	lzss
ec635.6	27980	0.00	0.00	-42.54	-15.58	-18.46	-13.18	-13.18	-12.09	-0.19	0.84	1.16	0.34	1.16	ari2	lzss
Max	50000	0.93	0.93	-39.66	-12.99	-13.54	-11.30	-11.30	-11.46	0.61	1.27	1.92	1.51	1.92		
Moy	46566	0.20	0.19	-41.00	-13.55	-14.62	-11.77	-11.77	-12.00	0.13	0.94	1.52	1.03	1.52		
Min	27980	0.00	0.00	-42.54	-15.58	-18.46	-13.18	-13.18	-12.26	-0.19	0.68	1.16	0.34	1.16		

TAB. 6.1 - *Comparatif des compresseurs pour les séquences d'Escherichia coli (procaryote).*

Séquence	Taille	cf2	cf4	lzss	lzw1	lzw2	lzw3	lzw4	huff	ari0	ari1	ari2	ari3	Max	MaxC	MinC
SCDCHR11	36272	0.56	0.54	-41.20	-13.56	-15.56	-11.59	-11.59	-9.54	2.01	2.27	2.06	1.03	2.27	ari1	lzss
SCPEKGA	38477	0.00	0.00	-41.81	-13.56	-15.36	-11.69	-11.69	-9.75	1.79	2.11	1.96	0.86	2.11	ari1	lzss
SCRACII	69748	0.63	0.61	-39.73	-11.87	-11.57	-10.66	-10.66	-9.73	1.93	2.28	2.24	1.74	2.28	ari1	lzss
YSCCHR1RAA	51950	0.39	0.36	-40.14	-12.99	-13.32	-11.37	-11.37	-9.48	1.94	2.09	1.91	1.17	2.09	ari1	lzss
YSCCHROMI	41987	0.73	0.73	-41.17	-12.64	-14.37	-10.94	-10.94	-9.42	2.22	2.52	2.29	1.32	2.52	ari1	lzss
YSCH8179	44113	0.62	0.61	-41.02	-13.32	-14.58	-11.40	-11.40	-9.85	1.62	1.87	1.70	0.75	1.87	ari1	lzss
YSCH9177	52664	2.02	1.99	-39.92	-12.74	-13.08	-11.13	-11.13	-9.66	2.07	2.26	2.11	1.38	2.26	ari1	lzss
YSCH9986	41663	0.00	0.00	-42.20	-13.22	-14.81	-11.50	-11.50	-9.78	1.81	2.06	1.92	0.90	2.06	ari1	lzss
YSCMTCG	78520	4.56	4.54	-9.34	8.83	8.67	9.90	9.90	7.55	15.74	19.91	20.57	21.00	21.00	ari3	lzss
YSCSYGP2	36772	0.02	0.02	-42.25	-14.25	-16.04	-12.29	-12.29	-10.18	1.47	1.70	1.42	0.46	1.70	ari1	lzss
Max	78520	4.56	4.54	-9.34	8.83	8.67	9.90	9.90	7.55	15.74	19.91	20.57	21.00	21.00		
Moy		0.95	0.94	-37.88	-10.93	-12.00	-9.27	-9.27	-7.98	3.26	3.91	3.82	3.06	2.36		
Min	36272	0.00	0.00	-42.25	-14.25	-16.04	-12.29	-12.29	-10.18	1.47	1.70	1.42	0.46	1.70		

Séquence	Taille	cf2	cf4	lzss	lzw1	lzw2	lzw3	lzw4	huff	ari0	ari1	ari2	ari3	Max	MaxC	MinC
HSG6PDGEN	52173	3.59	3.33	-32.42	-10.04	-11.43	-8.42	-8.42	-11.89	0.67	3.00	3.14	3.00	3.59	cf2	lzss
HSMHCAPG	66109	0.49	0.43	-38.12	-11.43	-11.60	-10.16	-10.16	-10.82	1.53	3.31	3.32	2.62	3.32	ari2	lzss
HSTCRBV	77743	2.11	2.06	-35.03	-10.01	-9.58	-8.92	-8.92	-10.08	1.66	3.65	3.82	3.35	3.82	ari2	lzss
HUMFMR1S	61613	0.35	0.33	-37.58	-10.94	-10.47	-9.57	-9.57	-9.20	2.63	4.10	3.91	3.21	4.10	ari1	lzss
HUMHBB	73323	2.89	2.84	-37.53	-10.08	-9.91	-8.93	-8.93	-9.73	2.20	4.12	4.08	3.44	4.12	ari1	lzss
HUMHDABCD	58864	3.26	2.98	-33.68	-10.40	-11.49	-8.97	-8.97	-11.49	1.07	2.77	2.87	2.66	3.26	cf2	lzss
HUMHPRTB	56737	2.32	2.10	-34.14	-10.17	-10.48	-8.68	-8.68	-10.03	1.76	3.32	3.57	3.23	3.57	ari2	lzss
HUMMMDBC	68505	3.91	3.65	-31.70	-9.23	-9.35	-8.00	-8.00	-11.75	0.49	2.49	3.02	3.12	3.91	cf2	lzss
HUMVITDBP	55136	0.54	0.49	-37.32	-10.32	-10.65	-8.79	-8.79	-8.79	2.68	4.59	4.52	3.80	4.59	ari1	lzss
Max	77743	3.91	3.65	-31.70	-9.23	-9.35	-8.00	-8.00	-8.79	2.68	4.59	4.52	3.80	4.59		
Moy	63355	2.16	2.02	-35.28	-10.29	-10.55	-8.94	-8.94	-10.42	1.63	3.48	3.58	3.16	3.81		
Min	52173	0.35	0.33	-38.12	-11.43	-11.60	-10.16	-10.16	-11.89	0.49	2.49	2.87	2.62	3.26		

TAB. 6.2 - *Comparatif des compresseurs pour les séquences 1/ de Saccharomyces cerevisiae et 2/ de l'Homme (eucaryotes).*

Séquence	Taille	cf2	cf4	lzss	lzw1	lzw2	lzw3	lzw4	huff	ari0	ari1	ari2	ari3	Max	MaxC	MinC
ATACCSYNG	5613	0.23	0.22	-51.65	-22.86	-43.67	-14.52	-14.52	-9.46	3.01	2.94	1.37	-3.55	3.01	ari0	lzss
ATAHA9	6110	0.04	0.03	-53.58	-23.21	-43.31	-15.55	-15.55	-10.51	0.82	1.80	0.16	-4.55	1.80	ari1	lzss
ATATPGP1	6300	0.00	0.00	-53.65	-23.81	-43.75	-16.44	-16.44	-10.22	1.08	1.27	-0.13	-4.89	1.27	ari1	lzss
ATATSGS	9647	10.75	10.48	-31.19	-17.96	-32.35	-11.83	-11.83	-9.13	3.02	3.10	2.44	-0.55	10.75	cf2	lzw2
ATCSCH42	6801	0.00	0.00	-51.27	-21.92	-40.98	-15.04	-15.04	-9.87	1.43	1.90	0.72	-3.57	1.90	ari1	lzss
ATCYSKIN	6436	0.00	0.00	-51.58	-22.87	-42.45	-15.60	-15.60	-10.25	1.55	2.11	0.81	-4.10	2.11	ari1	lzss
ATEF1A23	6022	20.77	20.54	-32.31	-22.62	-42.81	-14.85	-14.85	-10.79	0.83	1.30	0.30	-3.42	20.77	cf2	lzw2
ATGLYRP	5500	3.38	3.25	-46.55	-21.67	-42.62	-13.16	-13.16	-8.29	2.62	1.60	1.31	-1.09	3.38	cf2	lzss
ATGRPG	9619	3.74	3.63	-38.73	-18.64	-33.07	-12.49	-12.49	-9.16	2.32	1.99	1.61	-0.76	3.74	cf2	lzss
ATHACOACAR	9581	0.00	0.00	-50.76	-21.11	-35.60	-14.94	-14.94	-10.01	0.68	1.35	0.43	-3.29	1.35	ari1	lzss
ATHANSYNAA	5908	0.00	0.00	-54.71	-23.97	-44.41	-16.05	-16.05	-9.61	1.42	1.49	-0.14	-5.35	1.49	ari1	lzss
ATHANSYNAB	6661	0.00	0.00	-52.47	-22.32	-41.60	-15.30	-15.30	-10.01	1.70	1.76	0.20	-4.43	1.76	ari1	lzss
ATHATP	5646	0.00	0.00	-54.73	-23.98	-44.88	-15.69	-15.69	-9.74	1.88	2.23	0.74	-4.36	2.23	ari1	lzss
ATHATPASEP	6807	0.00	0.00	-51.61	-22.11	-41.21	-15.29	-15.29	-9.89	1.87	2.39	0.87	-3.25	2.39	ari1	lzss
ATHCTR1B	6312	0.19	0.18	-55.26	-23.45	-43.28	-16.03	-16.03	-9.57	1.65	1.65	0.13	-4.75	1.65	ari0	lzss
ATHGLAB	7710	0.10	0.10	-50.40	-20.88	-38.94	-14.81	-14.81	-10.30	1.58	1.12	0.08	-3.45	1.58	ari0	lzss
ATHLEAFY	7308	0.05	0.04	-50.30	-21.51	-40.01	-15.11	-15.11	-8.87	2.41	1.97	0.66	-3.67	2.41	ari0	lzss
ATHPHTOCHB	6509	0.00	0.00	-54.86	-23.83	-43.43	-16.64	-16.64	-9.94	1.06	1.00	-0.84	-5.70	1.06	ari0	lzss
ATHPHYTOA	7277	0.00	0.00	-52.92	-22.08	-40.66	-15.71	-15.71	-9.72	1.39	1.88	0.40	-4.11	1.88	ari1	lzss
ATHPOLYUBQ	5255	1.96	1.76	-53.76	-26.89	-48.81	-18.06	-18.06	-11.06	-0.10	0.13	-1.69	-7.10	1.96	cf2	lzss
ATHRD29AB	8048	0.61	0.57	-43.69	-19.88	-37.13	-13.72	-13.72	-9.10	1.84	2.49	1.39	-2.14	2.49	ari1	lzss
ATHREPEATS	5311	8.16	7.80	-37.60	-22.84	-44.30	-14.10	-14.10	-7.78	3.37	2.99	1.56	-2.88	8.16	cf2	lzw2
ATHRPB3A	5031	0.03	0.02	-53.93	-25.14	-47.33	-15.84	-15.84	-9.56	2.37	2.21	0.14	-5.27	2.37	ari0	lzss
ATLTA12	5262	7.41	7.29	-54.69	-24.52	-46.18	-15.70	-15.70	-9.08	1.10	2.32	0.72	-4.29	7.41	cf2	lzss
ATMET11	11883	0.00	0.00	-48.48	-21.18	-32.86	-16.20	-16.20	-9.40	1.20	0.90	-0.28	-3.41	1.20	ari0	lzss
ATPGIC	5137	0.06	0.05	-55.81	-24.12	-46.00	-15.09	-15.09	-9.87	2.04	2.43	0.64	-5.12	2.43	ari1	lzss
ATRBCSB	9647	10.75	10.48	-31.19	-17.96	-32.35	-11.83	-11.83	-9.13	3.02	3.10	2.44	-0.55	10.75	cf2	lzw2
ATRDNAF	10014	9.30	9.07	-38.73	-20.91	-34.77	-15.00	-15.00	-11.88	0.06	0.82	-0.46	-3.46	9.30	cf2	lzss
ATRDNAI	5287	26.60	26.22	-26.88	-20.82	-42.08	-11.97	-11.97	-11.59	-0.17	1.19	0.28	-2.14	26.60	cf2	lzw2
ATRNAPIIG	9064	0.08	0.06	-47.13	-20.26	-35.57	-13.72	-13.72	-9.75	1.94	2.29	1.32	-2.38	2.29	ari1	lzss
ATRPB1	8050	0.09	0.07	-49.52	-21.39	-38.63	-15.08	-15.08	-10.71	0.72	1.02	-0.22	-4.00	1.02	ari1	lzss
ATRPII	9235	0.15	0.14	-47.70	-20.97	-36.00	-14.56	-14.56	-10.49	1.03	1.29	0.16	-3.17	1.29	ari1	lzss
ATRPS14	5036	0.00	0.00	-59.57	-27.48	-49.88	-18.27	-18.27	-10.96	0.16	0.24	-1.67	-7.23	0.24	ari1	lzss
ATSUS1	5007	0.01	0.00	-54.98	-23.99	-46.12	-14.64	-14.64	-10.25	1.74	2.70	0.86	-4.57	2.70	ari1	lzss
ATTA13	5258	7.55	7.49	-53.52	-24.46	-46.06	-15.56	-15.56	-9.01	1.10	2.32	0.80	-4.22	7.55	cf2	lzss
ATU01843	5255	0.01	0.00	-54.44	-23.84	-45.46	-15.01	-15.01	-9.61	2.42	2.57	0.67	-4.74	2.57	ari1	lzss
CHATTPROG	9045	6.15	5.97	-42.05	-17.94	-33.29	-11.49	-11.49	-9.05	3.86	3.59	2.44	-0.61	6.15	cf2	lzss
Max	11883	26.60	26.22	-26.88	-17.94	-32.35	-11.49	-11.49	-7.78	3.86	3.59	2.44	-0.55	26.60		
Moy	6988	3.19	3.12	-48.44	-22.31	-41.13	-14.89	-14.89	-9.83	1.62	1.88	0.55	-3.68	4.41		
Min	5007	0.00	0.00	-59.57	-27.48	-49.88	-18.27	-18.27	-11.88	-0.17	0.13	-1.69	-7.23	0.24		

TAB. 6.3 - *Comparatif des compresseurs pour les séquences d'Arabidopsis thaliana (eucaryote et plante).*

- Lzss, la seule méthode de type LZ77, **est toujours la moins performante**, à trois exception près (voir les colonnes MinC). Les méthodes basées sur LZ78 compriment mieux, cela souligne l'importance d'éviter la limitation de la fenêtre imposée dans LZ77.

- Lzw2 comprime presque toujours moins bien que lzw1. Cela établit l'influence négative d'une taille d'index trop grande. Les séquences longues contiennent tout le vocabulaire possible pour les petites tailles de mots. Ainsi lzw2 génère un dictionnaire plus grand que lzw1, ce qui l'oblige à encoder les répétitions avec des index de plus grande taille. Contrairement aux textes usuels, les séquences ne semblent pas contenir assez de répétitions courtes pour que les taux soient positifs.

- Remarquons aussi que lzw3 et lzw4 donnent les mêmes performances : un index pouvant atteindre une taille supérieure à 15 bits ne sert sûrement à rien. Ces deux compresseurs se comportent relativement mieux que lzw1 et 2, à cause de la connaissance de l'alphabet réel des séquences.

Les schémas statistiques.

- Le résultat le plus intéressant est le bon comportement moyen des encodeurs arithmétiques. Il existe toujours des régularités statistiques qu'un des trois modèles (d'ordre 0, 1 ou 2) décèle. Rappelons que le codage arithmétique est optimal pour exploiter une régularité statistique. Il repère bien les vraies régularités statistiques que l'on savait exister dans les séquences nucléotidiques : disproportion du contenu en $G + C$ ([Ber89, BMGB89]), biais dans l'utilisation des dinucléotides et des codons.

- Les récapitulatifs confirment comparativement la «non-optimalité» de la méthode de Huffman. Celle-ci ne tire aucun avantage d'un modèle statique ad hoc. Elle est la méthode la moins performante parmi les schémas statistiques.

- Contrairement à ce qui est affirmé dans [GT93b], l'encodeur qui obtient le plus constamment le meilleur taux est ici arith1, et non pas arith2. Cette suprématie n'est pas permanente. Il semble donc opportun, pour optimiser le taux de compression, de surcompresser les séquences restantes produites par *Cfact* ou *Cpal*, grâce à un encodeur arithmétique.

Aptitudes de *Cfact*.

Les résultats de cf2 et cf4 se ressemblant fort, nous utilisons *Cfact* pour parler des résultats des deux compresseurs. *Cfact* est capable de compresser fortement les séquences régulières, i.e. qui contiennent des répétitions significativement longues. Les taux maximaux obtenus avec *Cfact* sont plus élevés qu'avec n'importe quel autre schéma. La grande différence entre ses performances moyennes et maximales démontre la faible proportion de telles répétitions dans l'ensemble de l'ADN testé. Rappelons que les séquences incluent des répétitions courtes, mais elles ne sont pas significatives (elles peuvent exister dans une séquence aléatoire) et ne permettent pas de compresser : d'ailleurs les taux résultants des schémas Lempel-Ziv qui encodent des répétitions courtes sont négatifs ! ***Cfact* est donc le seul de notre ensemble à identifier et encoder avantageusement les longues répétitions. Il représente un algorithme comparativement «optimal» pour ces régularités, i.e. pour le codage de répétitions.**

***Cfact* ne donne jamais un taux négatif.** Ils sont les seuls compresseurs à ne jamais provoquer une expansion de la séquence en tentant de la comprimer. Ceci est dû conjointement à l'évaluation préalable du gain utilisée durant la factorisation et à l'examen des facteurs par ordre de longueur décroissante.

Remarquons que la méthode employée pour cela est incrémentale. En effet, un compresseur peut toujours effectuer la compression puis vérifier que la séquence encodée est moins longue que la séquence originale. Dans le cas contraire, il remet la séquence originale dans le fichier «compressé», en signalant cet état de fait par un bit spécifique en début de fichier. Il faut donc que ceci soit prévu dans le codage. En plus, cela conduit à une absence de compression, alors que d'autres choix de factorisation auraient peut être comprimé réellement la séquence. Pour le savoir, il faudrait refaire toute la compression, codage y compris ! Les choix de *Cfact* sont moins risqués et plus faciles à remettre en cause, parce que pris localement pour chaque zone encodable. **Nous qualifions la méthode de choix des zones encodées, d'incrémentale, parce que *Cfact* examine la validité de chaque choix individuel plutôt que de décider d'une combinaison de choix et de tester globalement le gain in fine.**

6.3.2 Tests de compression et analyse de séquences.

Nous ne pouvons considérer les expériences relatées ci-dessus comme des expérimentations biologiques. Les critères de sélection des séquences n'ont pas été déterminés en fonction d'une expérience biologique précise à mener, i.e. une hypothèse à tester sur des données appropriées du point de vue génétique. Les remarques de cette sous-section envisagent donc simplement l'intérêt de la compression pour l'analyse de séquences. Elles soulignent le pouvoir discriminant d'un test comparatif de compression. De tels tests sont capables de générer une classification des séquences interprétable en termes de présence de régularités significatives du point de vue de la quantité d'informations.

Identification de régularités dans des séquences particulières.

Grâce à la garantie de compression, on sait que si *Cfact* ne parvient pas à comprimer une séquence, cela est dû à une absence de répétitions exactes significatives. Les récapitulatifs montrent que certaines séquences sont particulièrement bien compressées par *Cfact* ou certains encodeurs arithmétiques. Elles recèlent donc une quantité anormalement élevée de sous-séquences régulières (pour *Cfact*) ou un fort biais statistique (pour les arithn). Par exemple, YSCMTCGG est la seule séquence où arith3 obtient le maximum. De même, ATEF1A23 est compressible de 20% par *Cfact*. On peut imaginer alors un test comparant la compression de divers schémas sur toutes les séquences introduites dans une banque. Cela aiderait à l'annotation des séquences et permettrait d'identifier des régularités significatives : répétitions d'une part, biais statistiques d'autre part. Reste à choisir les schémas selon des considérations biologiques portant sur les régularités intéressantes.

Caractérisation de la compressibilité des séquences d'une espèce.

On remarque en comparant les récapitulatifs que le comportement global des compresseurs varie. Par exemple, chez A. thaliana, le compresseur arith3 ne donne jamais que des taux négatifs. Ce n'est pas du tout le cas chez E. coli ou dans les génomes mitochondriaux. De même, avec *Cfact* le taux moyen reste inférieur à 1% pour E. coli, B. subtilis, S. cerevisiae et pour les mitochondries, tandis qu'il est supérieur chez toutes les autres espèces et

dépasse même 2% pour l'homme, X. laevis et A. thaliana. On voit par là, qu'en testant sur des séquences de chromosomes complets, on peut identifier des caractéristiques globales de l'ADN d'une espèce. Elles peuvent servir en tant que comparaison entre espèces mais aussi en donnant le gain moyen escompté pour la compression d'une séquence de cette espèce. Une base de données regroupant des informations globales sur l'ADN des espèces serait peut-être pertinente du point de vue biologique et phylogénétique.

Deuxième partie

Analyse des séquences répétées en tandem.

Chapitre 1

Introduction.

La présence de séquences répétées dans l'ADN est une caractéristique commune aux génomes de toutes les espèces. Un locus (segment d'une séquence génétique) peut être répété:

- soit par *duplication directe* dans l'ADN,
- soit de manière renversée et complémentaire selon les appariements de Watson-Crick (A se lie avec T et C se lie avec G), il s'agit alors d'un *palindrome génétique*,
- soit encore dupliqué plusieurs fois côte à côte: cela forme une *répétition en tandem.*

Ces trois cas sont illustrés dans la figure 1.1. Des segments incluant de telles répétitions possèdent des éléments de symétries dans leur séquence et sont aussi dénommés: segments de *DOS-DNA* (Defined Ordered Sequence-DNA voir [WS93]).

Ces séquences répétées ont été intensivement étudiées par les biologistes, en attestent la découverte des familles de répétitions dispersées dans les génomes des invertébrés et des mammifères (famille des séquences Alu chez l'Homo sapiens cf. [KR88, HHL+89]) ou les études sur le rôle des palindromes génétiques dans la conformation spatiale des ARN (le repliement en structure secondaire cf. [LMMH89, MDBB93]). Pour autant, la plupart de ces séquences ne sont associées à ce jour, à aucune fonction biologique et sont souvent considérées comme inutiles.

1. Répétition directe:

   acggtaAGGTCTTAggatcAGGTCTTAccgcta

2. Palindrome génétique:

   ttcgtacccGTAACTGCGAacatggTCGCAGTTACcattg

3. Répétitions en tandem:

   cgcctagGTAGTAGTAGTAGTAtctaccag

FIG. 1.1 - *Exemples de répétitions dans les séquences d'ADN (en majuscules).*

Depuis 1991, des découvertes génétiques ont généré un formidable intérêt chez les biologistes pour l'étude des séquences répétées. En effet, plusieurs recherches ont établi un lien entre 4, puis 7 maladies génétiques et la présence de répétitions en tandem de trinucléotides (pour les maladies voir [WS93], cet article présente complètement le problème génétique). Dès lors, un important courant de recherche se concentre sur la localisation et l'analyse de répétitions en tandem, et de DOS-DNA en général.

Les méthodes expérimentales basées sur la reconnaissance par hybridation d'oligonucléotides particuliers ([BKMD93, HBM+94]) permettent l'identification de répétitions en tandem d'un seul motif par manipulation. Elles sont donc inappropriées pour une étude plus générale de ce genre de locus. D'un côté, des méthodes statistiques ([GSP+91, HHZ+94, KBS+93]) ont étudié généralement la répartition des répétitions en tandem. D'autre part, [GT93a, MJ93a, RDDD94, Riv94] abordent l'emploi d'outils de compression pour l'identification et la comparaison de DOS-DNA de manière générale.

En utilisant le cadre de la compression de texte, nous apportons une approche computationnelle et opérationnelle à la problématique biologique de l'analyse de répétitions en tandem de mono-, di- et trinucléotides. Ces répétitions en tandem de courts motifs peuvent inclure des mutations ponctuelles (substitutions, insertions et délétions) et se placent ainsi hors de portée d'un algorithme qui ne traite que les répétitions parfaites.

Notre approche apporte la possibilité de **localiser précisément de telles séquences et de donner une mesure objective de leur importance quantitative**. Cette mesure est le gain ou le taux de compression et sert pour comparer les différents loci identifiés. Par ailleurs, le codage utilisé pour transcrire ces séquences en binaire permet **leur description détaillée** incluant : leur longueur, leur position, le motif répété et la liste des mutations. Ceci représente un atout vis à vis des méthodes statistiques.

Ces outils dédiés à une problématique précise ont permis la réalisation d'expérimentations portant sur les quatre premiers chromosomes séquencés de levure. Leurs résultats sont présentés dans le chapitre 4.

Chapitre 2

La problématique biologique des répétions en tandem.

Ce chapitre propose un survol biologique du concept de répétitions en tandem dans les séquences d'ADN, puis s'attache à détailler la problématique précise de notre étude. Celle-ci porte sur *le phénomène d'expansion des répétitions de triplets en tandem.* Sont explicitées alors, les raisons de cette étude sous la forme d'interrogations biologiques et pour chacune, l'apport de notre méthode. Nous présentons aussi les différents mécanismes hypothétiques pour ce phénomène d'expansion.

2.1 Panorama des répétitions en tandem (RT).

La présence de répétitions en tandem est une évidence dans tous les génomes étudiés à ce jour. Elles peuvent représenter une grande proportion de l'ADN (environ 5% chez l'homme, i.e. plus que de gènes, et jusqu'à plus de 20% chez Bovis domesticus (le bœuf)), ce qui pousse à s'interroger sur leur raison d'être, leur répartition et une éventuelle fonction. Le seul fait qu'elles ne soient liées à aucun phénotype précis, **invite à penser qu'elles ne sont associées à aucune fonction**. Nous donnons quelques informations éclairantes sur les deux grandes classes de répétitions: les gènes dupliqués en tandem et l'ADN satellite. **Dans la plupart de ces répétitions, la duplication du motif est imparfaite** dans la séquence, car elle a été altérée par mutations ponctuelles.

2.1.1 Les gènes dupliqués en tandem.

Cette sous-section montre que la «duplication» est un mécanisme d'évolution du génome. En effet, lorsque l'organisme a besoin d'une protéine, ce mécanisme permet au génome de créer le gène d'une protéine par duplication, puis modification d'un gène similaire existant, plutôt que de le créer ex-nihilo. Par exemple, chez la souris et chez l'homme, les gènes de l'alpha-fœtoprotéine et de l'albumine sont suffisamment semblables pour que l'on sache qu'ils proviennent de la duplication d'un gène ancestral. Chacune des copies a ensuite évolué séparément, par mutations ponctuelles notamment. À un niveau plus interne, chacune des protéines est composée de 3 sous-chaînes d'acides aminés, et la similarité de ces polypeptides prouve qu'ils furent eux aussi engendrés par duplication (voir figure 9.20 page 661 dans [SB92]).

2.1.2 L'ADN satellite.

Ainsi appelé parce qu'il apparaît lors de centrifugations, l'ADN satellite est une classe de répétitions plus large. Elle contient toutes les RT pour des motifs très divers de 2 pb à plusieurs centaines, dont la principale propriété est leur dispersion tout au long de l'ADN, y compris dans les séquences codantes pour des gènes. Dans les fonctionnalités évoquées pour ces RT, on distingue :

- La capacité à induire des conformations spatiales inhabituelles de l'ADN : l'empilement cyclique des molécules nucléotidiques déforme la double hélice d'ADN et lui impose des structures alternatives (nommées ADN-Z, triplex, ...). Cette caractéristique est commune à d'autres formes de DOS-DNA.

- Un mécanisme d'introduction de polymorphisme intraspécifique : la localisation dans le génome d'ADN satellite correspond souvent à des séquences qui varient (polymorphisme) selon les individus d'une même espèce (intraspécifique).

Deux sous-classes de RT ont été identifiées : *les mini- et les micro-satellites*. Les derniers sont des répétitions de motifs très courts inférieurs à 5 pb, et leur nombre de répétitions varie de quelques unités à quelques centaines. Notre approche leur est dédiée et les résultats du chapitre 4 en montrent des exemplaires. L'amplification anormale des RT de triplets, est retenue comme la cause principale des maladies génétiques évoquées en introduction (nous appelons cela phénomène d'expansion de triplets).

Les mini-satellites sont des RT de motifs plus grands, supérieurs à 5 pb (par opposition), dont le nombre peut atteindre plusieurs milliers. Les principaux représentants sont les ADN satellites présents dans les centromères et télomères des chromosomes, dont les pourcentages très importants sont évoqués dans le résumé au début du panorama. Un motif particulier par exemple $AAGAGAG$ chez la mouche du vinaigre est répété 16000 fois au centromère du chromosome 3. Un autre exemple est le satellite-α humain (170 pb) répliqué par paquet de 12 sur 2,1 kpb, auquel on ne connaît pas de fonction.

2.2 Modèles de mécanismes d'amplification des répétitions en tandem (de triplets).

Les processus biologiques qui entrainent l'amplification des répétitions en tandem de triplets et les maladies qui en découlent ne sont pas connus. Nous présentons deux modèles hypothétiques de mécanismes avancés par les biologistes. L'article de référence à consulter pour plus de détails est celui de Wells et Sinden : [WS93]. Ces deux modèles sont respectivement basés sur : le processus de *recombinaison homologue inégale ou «crossing-over» inégal* et sur le *dysfonctionnement de la réplication* de l'ADN.

2.2.1 Amplification par crossing-over inégal.

Les mécanismes de recombinaison permettent le réarrangement des segments d'ADN dans les chromosomes et créent ainsi de nouvelles associations de gènes sur ces chromosomes. La recombinaison homologue ou générale, aussi appelée «crossing-over» constitue le processus le plus connu et le plus courant de remaniement de l'information génétique.

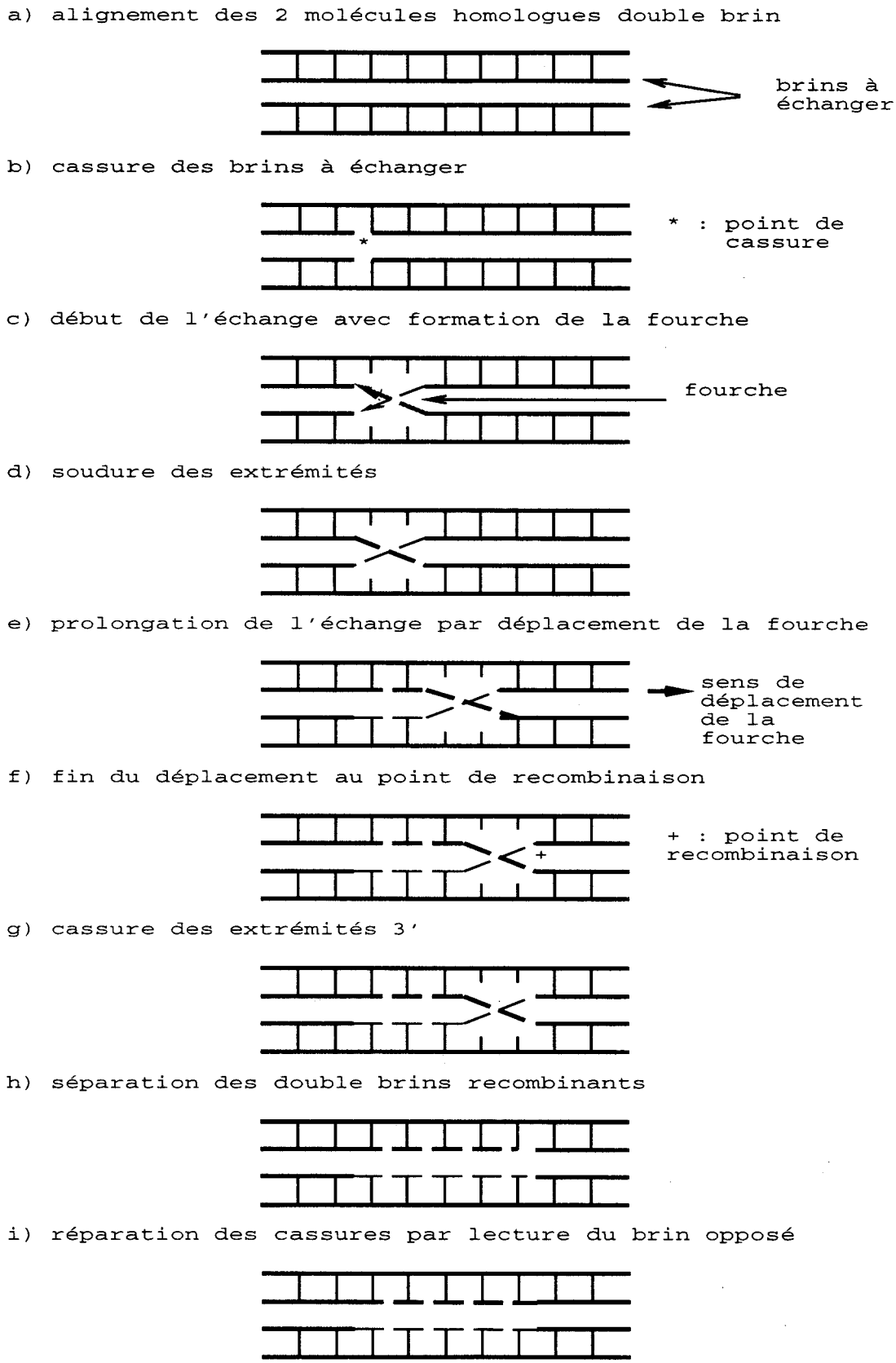


FIG. 2.1 - *Les étapes de la recombinaison homologue entre double brins d'ADN.*

Comme illustré sur la figure 2.1, ce processus consiste en l'échange de deux segments d'ADN presque identiques (i.e. homologues), contenant deux versions d'un même gène (appelé allèles) entre deux doubles chaînes d'ADN. Généralement ces deux molécules sont deux chromatides sœurs d'un même chromosome et le phénomène se déroule à la méiose. Le schéma représente 9 étapes de la recombinaison homologue entre deux double-brins d'ADN : les recombinants. La séquence des nucléotides est dessinée par un trait épais horizontal, tandis que les traits verticaux indiquent les appariements des bases complémentaires dans les 2 molécules. Les portions des brins échangés entre les recombinants ont des styles de traits différents. Au départ (étape c), le trait fin est sur la molécule du haut et à la fin (étape i), il est sur celle du bas.

La première étape (a) aligne les segments ressemblants d'ADN dans ces molécules (les met en face l'un de l'autre), puis dans les deux molécules, une chaîne se brise au point de cassure (étape b). Chacune des chaînes libres va s'apparier avec le brin complémentaire de la molécule d'en face (étape c). Ensuite la «fourche», point de croisement des brins échangés, se déplace le long des deux molécules jusqu'au point de recombinaison (étape d-e-f), où l'échange se termine, délimitant ainsi les segments échangés. Ceux-ci sont coupés à cette extrémité (étape g), ce qui résulte en la formation de deux nouvelles molécules indépendantes (étape h-i). Notons que dans ce processus de crossing-over «normal» les deux segments échangés sont de longueur identiques.

```
alignement   maximal des RT
             ccaTGCTGCTGCTGCTGCTGC...TGCTGCTGCTGCtgtt
             gtgTGCTGCTGCTGCTGCTGC...TGCTGCTGCTGCacc

alignement   partiel des RT
             ccatgctgctgcTGCTGCTGC...TGCTGCTGCTGCtgtt
                      gtgTGCTGCTGCTGCTGCTGC...TGCtgctgctgcacc
```

FIG. 2.2 - *Plusieurs possibilités d'alignement de deux répétitions en tandem lors d'une recombinaison homologue (segments alignés en majuscule). Sur l'alignement du haut, les débuts des deux RT sont en face l'un de l'autre.*

La condition déterminante de ce phénomène est la mise en correspondance ou alignement de deux segments quasiment identiques, mais pas forcément alléliques. Un alignement de RT met face à face deux segments capables de réaliser cette condition. Mais, de nombreuses mises en correspondance de ces segments sont possibles, comme l'indique la figure 2.2 ; parmi lesquelles certaines sont «décalées» : le début d'un segment de RT n'est pas mis en face du début de l'autre segment. Lors d'un processus similaire à celui du crossing-over normal, ceci entraîne un échange de segments de longueurs inégales entre les deux molécules : d'où le nom de crossing-over inégal. La figure 2.3 qui montre l'état initial et l'état final de ce crossing-over, met en évidence qu'une des molécules augmente le nombre de motifs dans sa répétition alors que la répétion en tandem de l'autre molécule a diminué en taille.

Deux remarques sur ce mécanisme de crossing-over inégal. Tout d'abord, les schémas des ouvrages de référence en biologie considèrent que le point de cassure ou celui de recombinaison du phénomène ne se produit pas au milieu d'un motif, produisant ainsi des molécules contenant des répétitions régulières du motif (notamment p. 440–442 du chapitre 21 partie 6

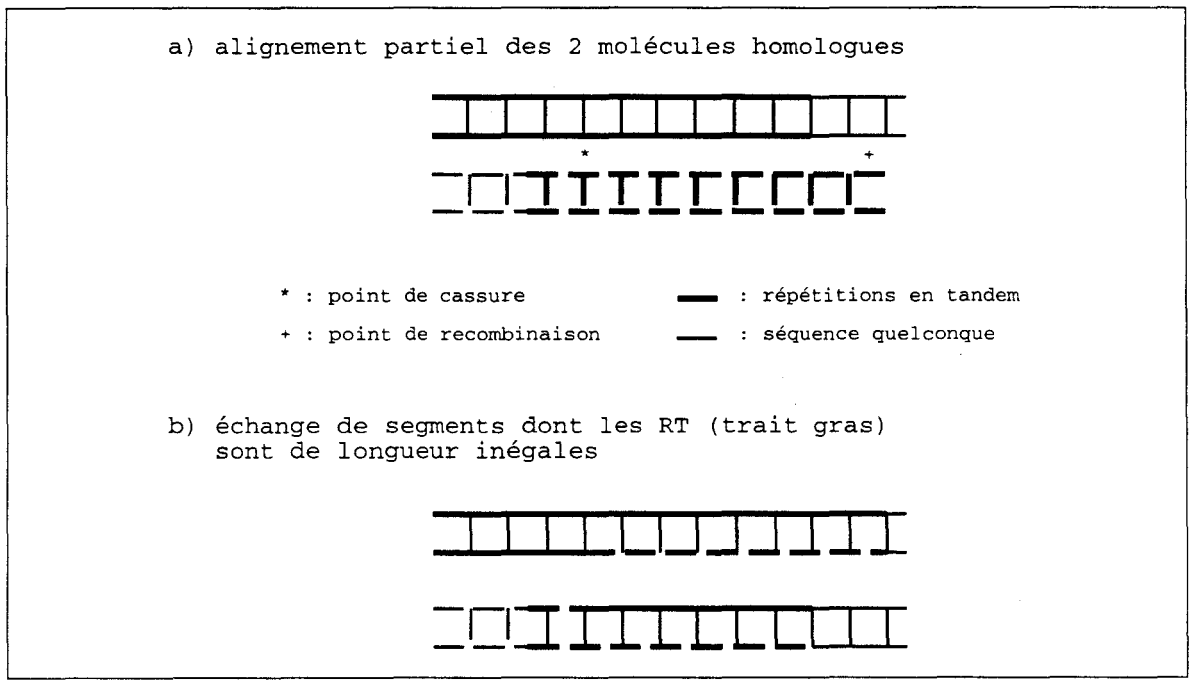


FIG. 2.3 - *Les étapes initiale et finale d'un crossing over inégal entre répétitions en tandem alignées partiellement (comme le deuxième alignement de la figure 2.2).*

de [Lew92] et p. 114–116 du chapitre 2 partie 1 de [SB92]). L'explication de cette hypothèse supplémentaire sur le processus n'est pas abordée.

Par ailleurs, l'hypothèse de l'utilisation de ce mécanisme est renforcée par le fait que les répétitions en tandem sont particulièrement sujettes à la recombinaison, fait dont il existe des évidences expérimentales.

2.2.2 Amplification par dysfonctionnement de la réplication.

Lorsqu'une cellule (dite mère) se divise en deux cellules filles, le stock complet de chromosomes est dupliqué et chacune des cellules filles en reçoit un exemplaire : ce phénomène s'appelle la *réplication de l'ADN*. Il est réalisé par un complexe protéique, dont l'élément principal est *l'ADN polymérase* : une molécule capable de générer une chaîne d'ADN.

La dupliquation porte sur un des deux brins d'une double hélice d'ADN, celui-ci est désolidarisé du brin complémentaire et mis sous forme linéaire. Ceci permet à l'ADN polymérase de s'intercaler dans la double hélice et de commencer la recopie du brin modèle, en appariant pour chaque base un nucléotide complémentaire et en se déplaçant vers l'extrémité 3′. Sur la figure 2.4, le point de séparation des deux brins côté 3′ forme la fourche de réplication. Ce point se déplace vers la droite poussé au fur et à mesure de la progression de la polymérase. La copie ainsi fabriquée est identique au brin complémentaire du brin pris comme modèle.

Tous les mécanismes d'amplification par réplication, présentent l'idée que le déplacement de la polymérase est entravé, provoquant ainsi :

- soit le glissement en arrière sur le brin modèle de la polymérase, ceci conduit à la nouvelle duplication d'un brin déjà répliqué, et donc à l'augmentation du nombre de répétitions ;

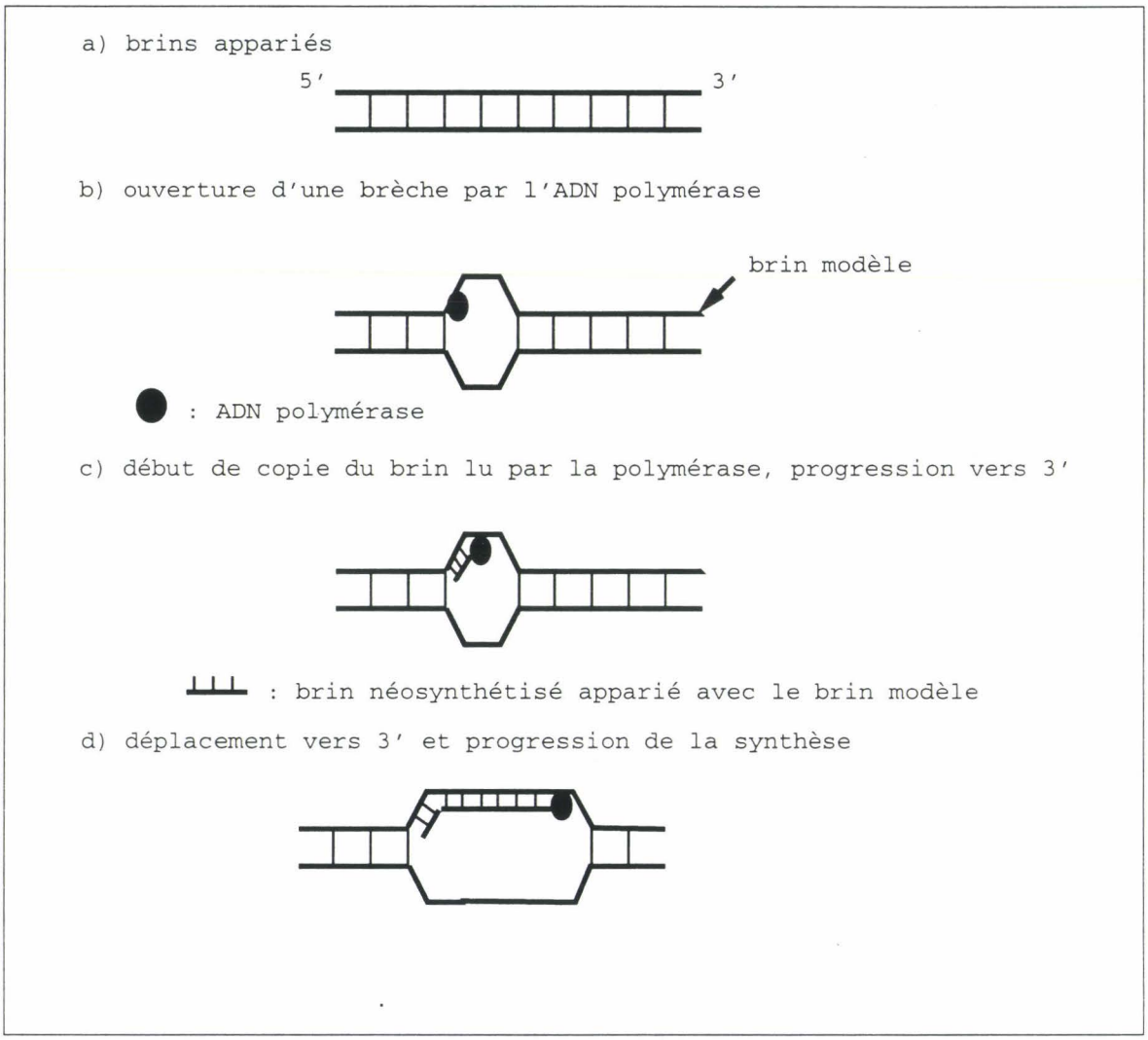


FIG. 2.4 - *Réplication d'un brin dans une molécule d'ADN double brin.*

- soit le saut en avant au dessus d'une partie non encore dupliquée dont le brin généré est ainsi privé, d'où une diminution du nombre de répétitions.

Deux hypothèses sont avancées pour ce dysfonctionnement. Dans le premier cas, une molécule vient s'attacher à l'ADN à la droite de la fourche, provoquant le blocage de la polymérase qui itère la duplication du même segment (voir figure 2.5). Le brin généré se replie au fur et à mesure, en une structure d'épingle à cheveux grâce à un «auto-palindrome» approximatif. Ceci impose des contraintes sur le motif de la répétition en tandem qui doit pouvoir s'apparier avec lui-même au moins partiellement. Par exemple le triplet CCG peut s'auto-apparier sur au moins deux bases sur trois. Cette remarque technique, indique l'importance de la question du choix préférentiel d'un motif pour ces amplifications (voir la section 2.3).

La deuxième hypothèse suggère que la polymérase changerait de brin modèle en sautant sur l'autre brin de la double hélice (en anglais phénomène de «strand-switching»). Ceci résulte en la duplication de la répétition en tandem complémentaire (qui peut-être identique dans sa séquence, par exemple AT complémentaire de TA et $(AT)^n \simeq (TA)^n$). Le phénomène est

illustré dans le schéma 2.6.

Une remarque générale sur les mécanismes : tous sauf la réplication itérative, ne permettent que de générer au maximum un doublement de la répétition en tandem. Or, la différence de taille des RT entre un individu atteint de la maladie et un individu sain peut varier d'un facteur 3 ou 4 à un facteur 10. Une telle évolution nécessite, selon ces modèles, soit plusieurs crossing-over inégaux, soit une répétition du phénomène de strand-switching. La réplication itérative permet seule l'avènement de très longues répétitions en tandem qui deviendraient ainsi pathogènes.

2.3 Interrogations sur l'expansion des répétitions en tandem de triplets.

La découverte à l'origine de l'intérêt pour le DOS-DNA, établit que certaines maladies génétiques proviennent du dysfonctionnement d'une protéine dont le gène inclut des RT de triplets. Ce dysfonctionnement est fortement corrélé à l'expansion dans ce gène, chez les individus malades, de la séquence de ces zones de RT, par augmentation du nombre de répétitions. Ceci est donc bien l'expression d'un polymorphisme intraspécifique humain. Ce nombre de répétitions augmente drastiquement avec la gravité de la maladie. Suite à cette présentation, nous listons les principales interrogations biologiques qui ont guidé cette étude et indiquons comment notre approche est adaptée pour y répondre.

1. L'expansion de RT en tant que mécanisme évolutif ne concerne-t-il que les triplets?
   En effet, aucune restriction n'exige que ce phénomène de mutagénèse ne s'applique qu'aux trinucléotides. Pour cela, nous avons construit trois algorithmes différents dédiés respectivement aux RT de mono-, di- et trinucléotides. Si l'on découvre par exemple, dans un chromosome la présence de RT de di- et de trinucléotides, ceux-ci ne portent pas forcément un même message : il doivent donc être étudiés séparément.

2. Certains motifs particuliers sont-ils préférentiellement soumis à ce mécanisme?
   Comme précisé dans la suite, notre méthode est indépendante du motif à rechercher (contrairement à certaines méthodes biologiques [BKMD93, HBM+94]). Elle peut localement trouver les RT de n'importe quel motif et donc extraire la liste des motifs répétés tout au long de la séquence.

3. Quelles caractéristiques peut-on attribuer aux mécanismes moléculaires responsables de ces expansions? Sont-ils conformes avec les modèles hypothétiques actuels?
   La compression fournit de nombreuses données soit sur l'ensemble des zones RT repérées (intervalle des gains), soit sur le détail de chacune (longueur, nombre de mutations, etc), qui donnent des indications sur ces mécanismes.

4. Quelle influence ces RT ont-elles sur l'organisation chromosomique?
   Ceci est une des questions les plus importantes car elle influencera une réponse à la question de la fonction de ces RT. Elle permet de savoir si l'apparition de ces RT est corrélée avec d'autres types de messages tels que les gènes, les promoteurs, etc. Pour cela, les expérimentations sont effectuées sur 4 séquences complètes de chromosomes de levure.

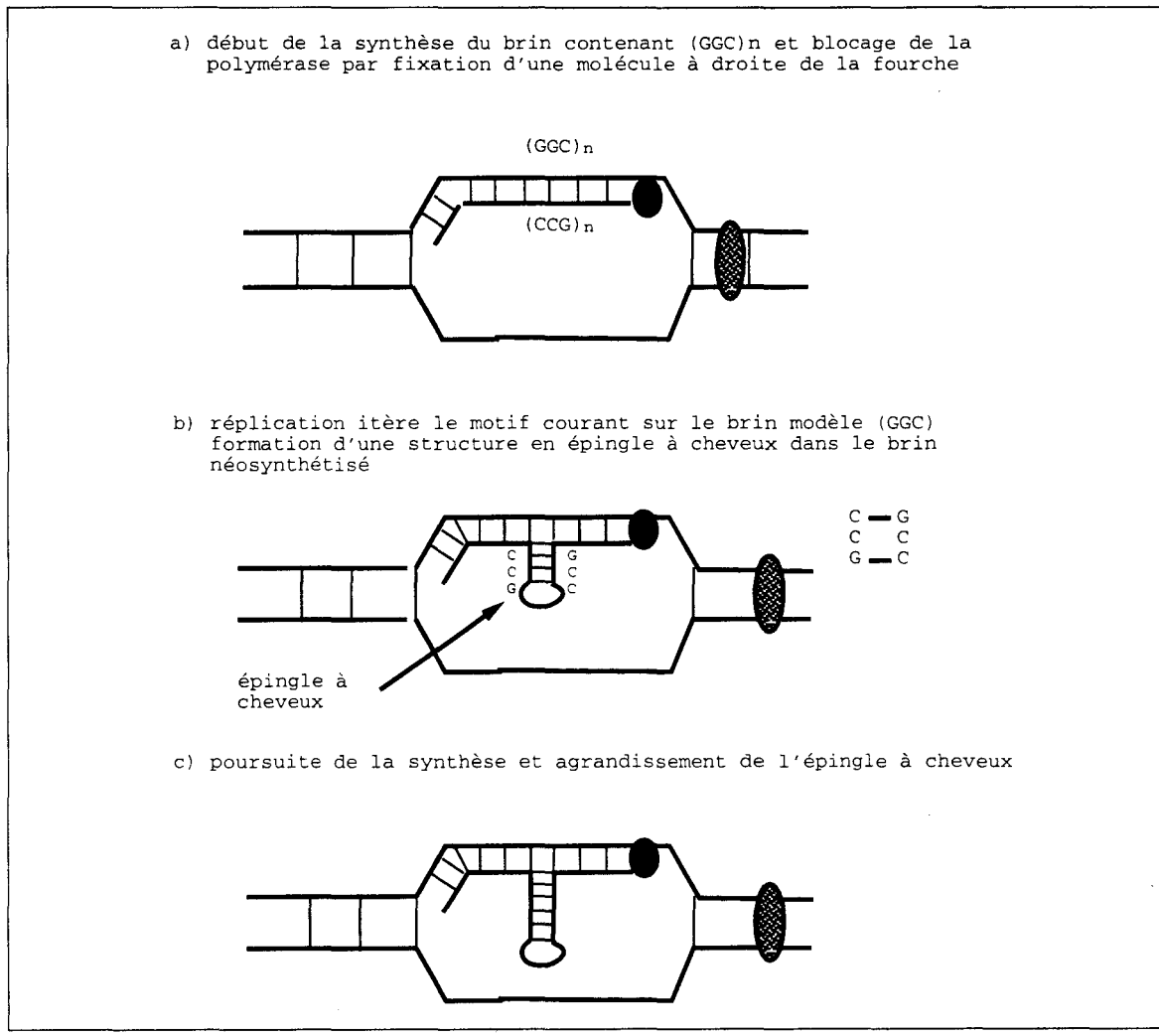


FIG. 2.5 - ***Réplication itérative*** *d'un brin contenant une répétition en tandem. La réplication est commencée quand une autre molécule vient bloquer la progression de la polymérase en se fixant à droite de la fourche sur l'ADN double brin. La polymérase continue à synthétiser le même motif lu sur le brin modèle.*

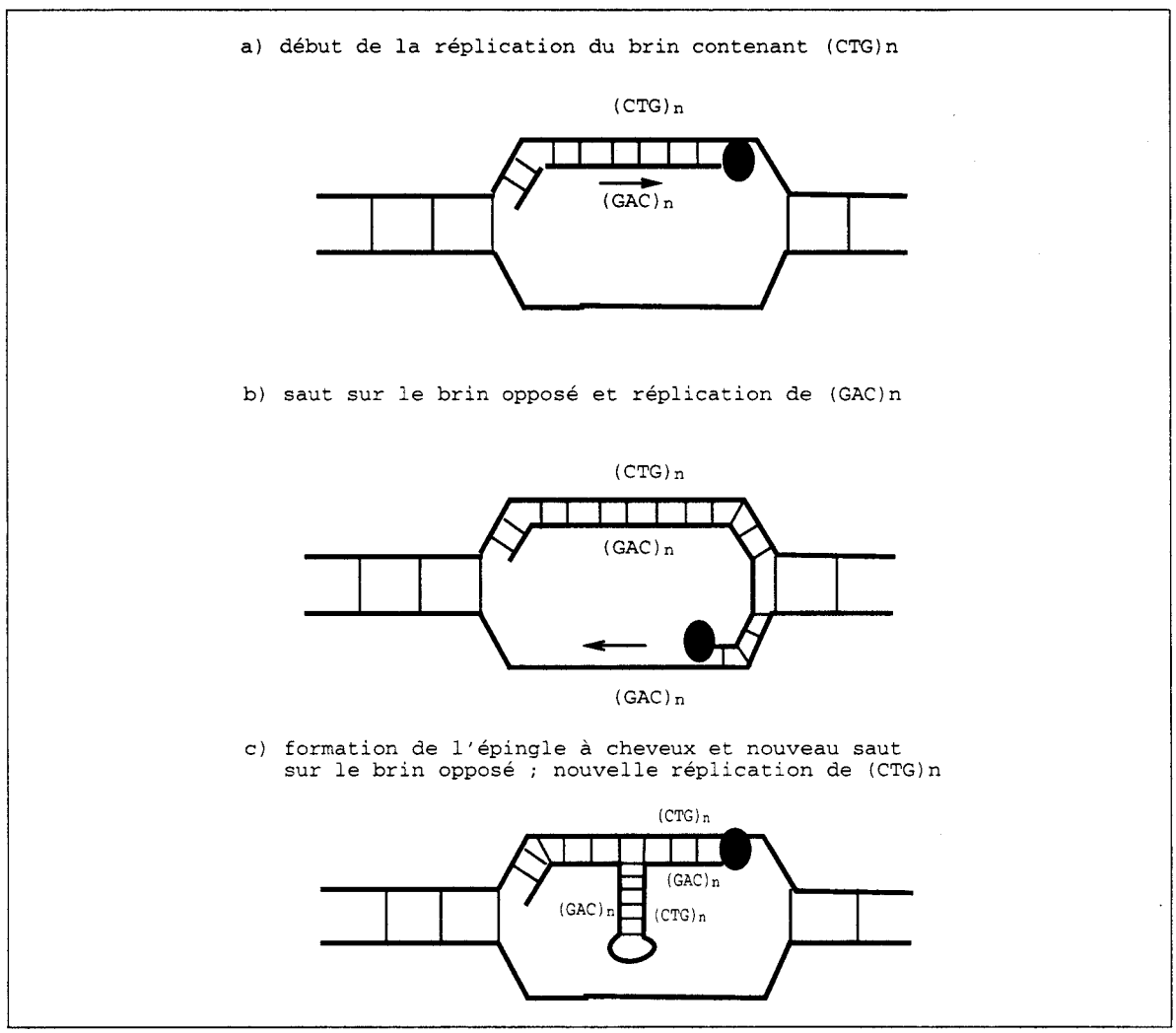


FIG. 2.6 - *Réplication avec saut sur l'autre brin. La polymérase commence à synthétiser la répétition en tandem $(CTG)^n$, puis saute sur l'autre brin, synthétise la répétition complémentaire $(GAC)^n$. Ces deux segments du brin néosynthétisé s'apparient en une épingle à cheveux, «obligeant» la polymérase à resauter sur le brin modèle original. Elle synthétise à nouveau la même répétition en tandem $(CTG)^n$. Le phénomène peut être itéré.*

Chapitre 3

Localisation des séquences répétées en tandem approximatives.

Nous présentons ici une méthode de compression de texte basée sur la localisation et le codage de répétitions en tandem. L'algorithme détecte des zones régulières composées de la concaténation d'un même facteur. Le motif (l'unité répétée) de ces répétitions est de petite taille. En outre, l'algorithme autorise la détection de répétitions approximatives car comme il est indiqué au début de la section 2.1, le génome subit en permanence des altérations parmi lesquelles des mutations ponctuelles. Les répétitions en tandem dans les séquences biologiques n'échappent pas à ce phénomène et sont donc souvent imparfaites. La capacité à détecter et encoder des zones qui incluent des erreurs ponctuelles est un impératif à une application biologique de notre méthode.

Dans la figure 3.1, nous montrons deux séquences génétiques: la première contient une répétition en tandem «exacte» du motif GAC, la seconde contient une répétition en tandem erronée ou approximative du motif TCA. Les erreurs ponctuelles autorisées comprennent la *délétion* d'un caractère au sein de la répétition, le «changement» d'un caractère par un autre, aussi appelé *substitution*, et *l'insertion* d'un caractère. Dans la deuxième séquence, un A a été supprimé, un C fut substitué en G et un G est inséré avant le dernier TCA.

L'algorithme commence donc par localiser des zones *similaires* à une répétition en tandem exacte d'un motif, selon une notion de similarité dont nous donnons une définition précise. Ensuite, il traduit ces zones dans un langage de description, et cette description est incluse dans la forme compressée de la séquence. Nous présentons ce codage qui est une manière efficace de décrire une répétition en tandem approximative. Le chapitre suivant précise le problème de localisation des répétitions en tandem et l'algorithme heuristique mis en œuvre. Nous terminons par la complexité de l'algorithme. D'autres approches, portant sur des problèmes très similaires sont présentées en premier lieu.

...GACGACGACGACGACGACGAC...

...TCATCTCATC**G**TCA**G**TCA...

FIG. 3.1 - *Exemples de répétitions en tandem exactes (RTE) et approximatives (RTA).*

3.1 Autres approches du problème.

Le problème de la compression de texte par le codage de répétitions en tandem approximatives n'a jamais été abordé à notre connaissance. Cependant divers travaux dans le domaine de la recherche de motifs ou «pattern matching», donnent une approche au problème de la détection de répétitions en tandem approximatives. Nous allons voir comment la notion d'approximation y est intégrée avant des les passer en revue.

3.1.1 Notion d'approximation dans les répétitions.

Les algorithmes présentés ci-dessous basent leur notion d'approximation dans une répétition de la même manière que les algorithmes de comparaison de séquence définissent la similarité entre deux loci : par le coût d'édition en nombre d'opérations élémentaires pour passer d'une chaîne à une autre. Rappelons que la possibilité de trouver des répétitions ayant des caractères erronés est primordiale pour les applications biologiques.

La notion d'opérations d'édition élémentaires pour «passer» d'une chaîne à une autre est connue de tous grâce au jeu qui consiste à *obtenir NUIT en partant de LUNE et en changeant une lettre à chaque étape*. Dans le cas de comparaison de chaînes, les opérations autorisées incluent la suppression et l'insertion de caractères en plus de l'opération de substitution.

À chaque opération portant sur un couple de lettres spécifiées est attribué un coût propre : par exemple une substitution de «A» en «C» peut coûter 2 alors que le changement de «T» en «G» ne coûte que 1, comme celui de «C» en «-» qui dénote la délétion de «C». L'ensemble des coûts est donné sous la forme d'une matrice carrée indicée par les lettres de l'alphabet et le symbole «-». La notion de ressemblance entre deux chaînes s'arrête au dessous d'un seuil fixé arbitrairement, qui représentre le coût maximal d'édition pour passer d'une chaîne à une autre.

La compression évite à l'utilisateur le problème du seuil de ressemblance, ce qui n'est pas un mince avantage. En fait, il utilise une matrice fixée qui calcule un coût se rapprochant d'une distance de Levenshtein [Lev66]. Toutes les opérations d'édition envisagées forment un sous-ensemble de la matrice complète et aucune n'est favorisée : elle ont toutes le même coût. Une mutation est vue comme une «erreur» dans une répétition parfaite, erreur qu'il faut encoder. La différence de taille entre la version codée de la séquence et la version originale est le gain de compression. Nous disons que «le seuil de ressemblance est atteint», si ce gain est positif.

3.1.2 Recherche des mots carrés approximatifs.

Le problème abordé dans [LS93, Sch94] consiste à identifier tous les couples de facteurs juxtaposés entre lesquels la distance d'édition est en dessous d'un seuil k pour une matrice de coûts fixée. Dans les deux cas, ces répétitions de deux unités similaires d'un même facteur, sont appelées *répétition en tandem*, alors que le sens que nous donnons à ce terme est la répétition approximative de n fois un même facteur. Par clarté, nous nommons les répétitions recherchées ici, des *mots carrés approximatifs*.

Méthode de Landau et Schmidt (93).

Le premier article résoud le problème avec un schéma d'algorithme similaire à celui de [ML84] dédié aux mots carrés sans approximation. Nous esquissons brièvement les idées de cet algo-

rithme.

Schéma de l'algorithme de Main et Lorentz pour les mots carrés exacts en $O(n\log(n))$.
[ML84] traite le problème récursivement car les mots carrés sont des régularités locales. Dans le texte $t[1..n]$, un carré est :

1. soit strictement inclus dans $t[1..\frac{n}{2}]$ ou symétriquement dans $t[\frac{n}{2}..n]$,
2. soit il chevauche les deux moitiés du texte.

Le deuxième cas ajoute une contrainte qui permet de ne rechercher que les carrés de la forme $u.v$ tels que $u = v$ et que u inclut le caractère $t[\frac{n}{2}]$ (le cas où v inclut $t[\frac{n}{2}]$ est similaire). Pour le premier cas, les deux moitiés de t sont traitées chacune par un appel récursif de l'algorithme.

Appelons $t[\frac{n}{2}]$ le point de cassure de u. La procédure consiste à vérifier pour tous les points de cassure possibles dans v, i.e. pour tous les indices j ($\frac{n}{2} < j \leq n$), si les conditions suivantes sont réalisées. La vérification est incrémentale, car on fait varier la longueur du suffixe de u qui débute après le point de cassure, notée l. On regarde :

- si le suffixe de u débutant en $\frac{n}{2}+1$ de longueur l correspond exactement à celui de v débutant en $j+1$ de longueur l, i.e. si $t[\frac{n}{2}+1..\frac{n}{2}+l] = t[j+1..j+l]$
- et si le préfixe de u se terminant en $\frac{n}{2}$ de longueur $j-\frac{n}{2}-l$ correspond au préfixe de v se terminant en j de même longueur, i.e. si $t[n+l-j..\frac{n}{2}] = t[\frac{n}{2}+l-1..j]$

Si les deux conditions sont vérifiées, alors $u.v$ est un carré où :

$$\begin{cases} u &= t[n+l-j..\frac{n}{2}] \,.\, t[\frac{n}{2}+1..\frac{n}{2}+l] \\ v &= t[\frac{n}{2}+l-1..j] \,.\, t[j+1..j+l] \end{cases}$$

Apport de [LS93] pour les carrés approximatifs.
[LS93] traite le problème où u et v ne sont pas identiques mais similaires, pour les notions de similarité suivantes :

1. le cas où $Hamming(u,v) \leq k$ en $O(nk\log(\frac{n}{k}))$ en temps
2. le cas où $Levenshtein(u,v) \leq k$ en $O(nk\log(k)\log(n))$ en temps

Pour trouver les préfixes et suffixes possibles pour un j donné, l'algorithme utilise une matrice D d'alignement de t contre lui-même, aussi appelée matrice de programmation dynamique ([SK83, Wat89] en donnent une définition). Les auteurs remarquent qu'il suffit de calculer le meilleur chemin entre 2 points de la diagonale représentant l'alignement de u contre v, i.e. celle qui passe en $D[j, \frac{n}{2}]$. Pour la distance de Hamming il suffit de calculer les coûts sur la diagonale car seules les opérations de substitutions sont permises. Pour la distance de Levenshtein, aussi appelée distance d'édition [Lev66], il est nécessaire de calculer plusieurs chemins pour un couple de points sur la diagonale.

Méthode de J.P. Schmidt (94).

Nous employons le terme poids avec le sens du coût d'édition entre deux facteurs u et v. [Sch94] sélectionne les carrés approximatifs pour ne donner que ceux qui sont localement optimaux. $u.v$ est *localement optimal* ssi :

qq. soit u' un préfixe de u $u'.v$ est un carré approximatif et $poids(u', v) \geq poids(u, v)$
qq. soit v' un suffixe de v $u.v'$ est un carré approximatif et $poids(u, v') \geq poids(u, v)$

Le problème est modélisé par une recherche de plus courts chemins, noté PCCs (i.e. de poids minimum), dans un graphe G pondéré et orienté qui prend la forme d'une grille : le graphe-grille. Ce graphe simule et peut être vu comme la matrice de programmation dynamique de t contre lui-même. Dans ce graphe, un carré de centre c (le centre de $u.v$ est l'indice du dernier car de u) est représenté par un plus court chemin entre un nœud de la colonne c et un nœud de la ligne c. Pour calculer le minimum des chemins entre des couples de nœuds (i, c) et (c, j) pour i et j variant, [Sch94] utilise les matrices $Dist_r$ (r variant de 1 à n) où $Dist_r[i, j] = cout(CPP[(i, 0), (r, l)])$, définies dans [AALM90, AP88].

Là aussi, le schéma algorithmique reprend celui de [ML84] : limitation du problème aux carrés qui chevauchent les deux moitiés du texte et appels récursifs pour ces moitiés. Dans le graphe-grille, on recherche pour les carrés de centre c, ceux dont le chemin associé croise la colonne $\frac{n}{2}$ à la ligne k. Le chemin complet est construit en associant deux sous-chemins, un d'un nœud de la colonne c vers $G[k, \frac{n}{2}]$, un second de $G[k, \frac{n}{2}]$ vers un nœud de la ligne c. La complexité de la méthode est en $O(n^2 \log(n))$ en temps et en $O(n^2)$ en espace.

3.1.3 Recherche de répétitions en tandem approximatives.

Le programme de [BW94] recherche les répétitions en tandem approximatives des motifs d'une longueur donnée le long d'une séquence. Il reçoit en paramètre :

- la longueur des motifs : m,
- la taille du facteur conservé dans le motif : c,
- la matrice des coûts d'édition,
- un seuil de similarité d'une RTA trouvée avec la RTE du motif choisi.

Étape 1.
L'algorithme effectue une passe sur la séquence en examinant les répétitions de c lettres qui apparaissent avec un décalage de m caractères (voir figure 3.2). Alors le mot u est considéré comme le motif d'une RTA, on entame dès lors l'étape 2.

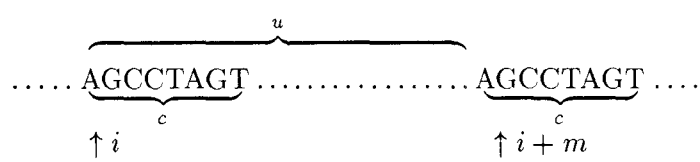


FIG. 3.2 - *Identification d'une répétition d'un bloc de c lettres, en i et i + m. Le motif u choisi est alors le mot de longueur m commençant en i.*

Étape 2.

Dans celle-ci, les limites de la zone RTA sont déterminées en alignant successivement les segments à droite et à gauche de la répétition, en observant le fléchissement du score de similarité. La procédure d'alignement est le «*wraparound dynamic programming*» *ou wdp* qui permet d'aligner un segment contre $\underbrace{u \ldots u}_{p \text{ fois}}$ (le wdp provient de [FLSS92]). Ensuite, un alignement complet de la zone par wdp est calculé et si son score dépasse le seuil, l'étape 3 de l'algorithme est déclenchée.

Étape 3.
On détermine un motif consensus dans lequel chaque base est majoritaire à sa position, pour recalculer un alignement qui peut améliorer le score de la RTA. On imprime en sortie, la zone et son alignement.

Par ailleurs, l'article propose une méthode statistique pour fixer le seuil de similarité, mais le calcul n'est pas inclus dans le programme. D'après les auteurs, «la méthode utilise une meilleure notion de similarité que dans [LS93]», mais elle est identique à celle de [Sch94], et «se veut moins restrictive que les méthodes de compression».

3.1.4 Conclusion

Le problème précis de la localisation des répétitions en tandem n'est abordée que dans [BW94]. L'implantation qui résulte de cette approche souffre de l'importance des paramètres, ce qui n'est pas le cas de notre approche via la compression. Par ailleurs, si une méthode de compression trouve moins de RTA qu'une autre méthode ce n'est pas dû à notre volonté de la régler pour qu'elle soit plus ou moins lâche. Ses résultats sont totalement contraints par la possibilité de coder telle ou telle zone, i.e. par le fait qu'une zone est ou n'est pas régulière.

Les méthodes apportées par Landau et Schmidt ne recherchent que les carrés approximatifs et non les répétitions en tandem telles que nous les définissons. De plus, la complexité nécessaire pour obtenir tous les carrés approximatifs par l'algorithme de [Sch94] est $O(n^2 \log(n))$ alors que notre algorithme heuristique de localisation des RTA est linéaire par rapport à la taille du texte.

3.2 Définition d'une répétition en tandem approximative ou RTA.

Cette section est consacrée à la définition formelle d'une zone de répétition en tandem dite approximative. Une telle zone notée RTA est un facteur d'un texte qui est similaire à une répétition en tandem exacte d'un motif, notée RTE. Nous définissons tout d'abord la notion de RTE puis celle de similitude à une RTE. Nous considérons des textes sur l'alphabet A.

Notation 4 Soient u un mot sur A et p un entier non nul. Le mot $\underbrace{u \ldots u}_{p \text{ fois}}$, formé de la concaténation p fois de u est noté u^p. La notation u^p représente plus un préfixe de u comme dans la partie I.

Définition 9 Soient u et t deux textes sur A. t est une répétition en tandem exacte ou u-RTE ssi :

$$\exists\, p > 1 :\ t = u^p$$

Nous définissons une *zone RTE maximale* dans un texte t.

Définition 10 Soient u, t deux textes sur A et w un facteur de t. w est une répétition en tandem exacte maximale ou RTE maximale de u ssi :

- w est une RTE de motif u
- il n'existe pas v, un facteur de t, tel que v soit une RTE de motif u et w soit un facteur de v.

Une répétition en tandem approximative d'un facteur u est un texte qui «ressemble» à une répétition en tandem exacte de u. Qu'un texte soit une RTA de u signifie intuitivement qu'il peut être lu comme une suite de mots, notés $(v_i)_{1 \leq i \leq p}$, tous similaires à u. La définition de *similarité à un motif* d'une RTE tient compte du fait que l'algorithme recherche des répétitions en tandem de petits motifs. L'utilisation principale concerne les motifs de longueur 3. Ainsi chaque v_i doit fortement ressembler au motif, sans quoi n'importe quel texte sur A^* peut être considéré comme une u-RTA. Chaque v_i doit donc inclure peu d'erreurs, en termes biologiques, il doit être faiblement muté. Nous définissons donc la similarité d'un mot à un motif, par son appartenance à un ensemble de mots obtenus en appliquant des opérations d'édition sur le motif.

Si u est un motif (i.e. un texte sur A), nous définissons les ensembles suivants de mots similaires à u :

- $del(u)$: ensemble de tous les mots obtenus en appliquant une délétion au motif u,
- $sub(u)$: ensemble de tous les mots obtenus en appliquant une substitution à u,
- $ins(u)$: l'ensemble des mots formés par u ou un mot de $del(u)$ ou de $sub(u)$, suivi d'une insertion.

Pour les quatre définitions suivantes, nous considérons un motif u qui est un mot de A^*.

Définition 11

$$del(u) = \{ \; l \in A^*, \quad |l| = |u| - 1, \; \exists \, j \in [1, |u|] : \; \forall \, i \in [1, j-1], \quad l[i] = u[i] \; \forall \, i \in [j, |u|-1], \quad l[i] = u[i+1] \; \}$$

Définition 12

$$sub(u) = \{ \; l \in A^*, \quad |l| = |u|, \; \exists \, j \in [1, |u|], \; l[j] \neq u[j], \; \forall \, i \neq j \quad l[i] = u[i] \; \}$$

Définition 13

$$ins(u) = \{ v = la : \; l \in \{u\} \cup sub(u) \cup del(u); \; a \in A \}$$

On définit l'ensemble des mots similaires à un motif u, il est noté $Sim(u)$.

Définition 14

$$Sim(u) = \{u\} \cup sub(u) \cup del(u) \cup ins(u)$$

Exemple 3 Pour le motif GTA, si on pose

$$\begin{aligned} L = \{ \; & GTA, \\ & ATA, CTA, TTA, \\ & GAA, GCA, GGA, \\ & GTC, GTG, GTT, \\ & GT, GA, TA \qquad \} \end{aligned}$$

on a : $Sim(GTA) = L \cup (L.a)$ où $a \in \{A, C, G, T\}$.

Formellement, le texte t est une RTA de motif u, s'il peut être factorisé avec des mots de $Sim(u)$. Nous notons $TailleMaxSim$ la taille maximale des mots de $Sim(u)$.

Définition 15 Soient u, t deux textes sur A. t est une répétition en tandem approximative ou u-RTA ssi :

$$\exists\, p > 0, \exists\, v_1, \ldots, v_p \in Sim(u) : \quad t = v_1 v_2 \cdots v_p$$

3.3 Codage des zones de répétitions en tandem.

Nous entendons par codage un langage de description avec lequel le compresseur produit le fichier en sortie, il traduit dans ce langage le texte passé en paramètre. La description comprimée du texte permet de distinguer les zones régulières (qui ressemblent à une répétition en tandem d'un motif donné) des zones non régulières. Seules les zones régulières sont codées dans une description avantageuse et différente de leur séquence. Ainsi, les zones non régulières sont décrites par la suite de leurs nucléotides, alors que les zones de répétitions en tandem sont excisées et remplacées par un code binaire permettant leur reconstruction. Dans le fichier en sortie, on distingue *la séquence «restante»* : constituée par l'ensemble des séquences des zones non régulières du texte, et *la partie «codée»* qui contient les codes de remplacement des zones régulières.

Pour une séquence en entrée, le compresseur renvoie une séquence codée. L'algorithme de décompression sait à partir de la version encodée reconstruire l'originale. Pour cela, la version codée doit décrire complétement la séquence telle qu'elle a été décomposée par le compresseur. Chacune de ces informations est codée indépendamment et la limite de la partie de code qui lui correspond, doit être détectable et détectée par le décompresseur. Après avoir précisé l'importance du codage, nous expliquons finement ce code binaire qui remplace les zones régulières et soulignons certaines propriétés qui influent sur l'algorithme de recherche.

3.3.1 Objectifs et contraintes pris en compte dans la conception du codage.

Nous récapitulons les idées qui guident la conception du codage :

1. Le codage doit décrire complètement le texte original pour que la compression soit sans perte d'informations.

2. Une zone régulière du texte est une zone dont on peut donner une description plus courte, i.e. qui se comprime. Comme *Cfact* l'algorithme dévéloppé en partie I, notre algorithme ne tente pas de compresser des zones qui lui semblent non régulières. Ces zones non régulières sont incorporées dans le fichier compressé, sous la forme de leur suite de nucléotides, où chaque base est codée sur 2 bits.

3. La longueur de la description d'une zone régulière doit refléter sa similarité avec une répétition en tandem parfaite du motif considéré. **Plus une zone régulière contient de mutations par rapport à la répétition en tandem parfaite, plus son codage binaire doit être long.** Nous verrons dans le détail du codage, que la description des mutations est une partie de longueur variable du code.

4. Le codage doit être «juste». Le langage de description qui code une zone RTA ne doit pas favoriser telle zone particulière par rapport à telle autre. Par exemple, le motif d'une zone, qu'il soit CGT ou ATT ne doit pas influer sur la longueur du code.

3.3.2 Signification du codage.

Le codage indique la manière dont l'algorithme «interprète» une zone régulière. Du point de vue de la sémantique, comme du point de vue pratique, le langage de description donne, pour une zone régulière, une description de la construction de cette zone. En effet, *du point de vue pratique*, l'algorithme de décompression doit reconstruire la séquence originale de la zone à partir de cette description. Tandis que *du point de vue de la sémantique*, ce langage révèle un modèle sous-jacent d'évolution des séquences génétiques. En effet, une zone régulière détectée est remplacée par un code ayant la signification suivante : cette zone «s'est construite» en dupliquant n fois le motif u, puis en appliquant les mutations ponctuelles listées ci-après.

On rejoint ici le point de vue de la complexité de Kolmogorov, où les zones sont les objets pour lesquels on recherche une description minimale. Une zone régulière est remplacée par un «programme» qui permet de la reconstruire ; si ce programme est plus court que sa séquence originale alors cette zone est compressible et donc effectivement régulière.

3.3.3 Description du codage des zones régulières.

Nous détaillons ici le codage des zones régulières qui, avec l'ensemble des séquences concaténées des zones non-régulières, forment la version compressée du texte.
À propos de la mise en œuvre de l'algorithme, présentée dans le chapitre 4, nous verrons que le texte à compresser est traité sous forme de fenêtres de 500 nucléotides, sur lesquelles on applique l'algorithme (la valeur de 500 nucléotides a peu d'influence sur notre algorithme et est expliquée dans la sous-section 4.1.2). Ainsi l'algorithme présuppose que les zones de répétitions en tandem approximatives qu'il peut détecter dans une fenêtre, sont toutes basées sur le même motif. Nous illustrons le codage pour les motifs de longueur 3 ou trinucléotides.

Le codage se comprend comme une suite d'éléments dont nous donnons la sémantique sans présenter le code binaire correspondant. Le codage spécifie d'abord le motif des répétitions et le nombre de zones régulières. Ensuite, vient pour chaque zone régulière, une description complète de la zone. On y trouve les items suivants :

1. Pour la description globale à la fenêtre :
   - le motif u, dont la longueur est spécifique à la version de l'algorihtme utilisée,
   - le nombre de zones régulières (zones RTA de motif u qui ont permis la compression),
2. Pour chacune des zones RTA :
   - la position relative du premier nucléotide de la zone,

- p le nombre de répétitions du motif pour lequel on a trouvé une similitude entre la zone courante et le facteur u^p
- le nombre de mutations trouvées par la factorisation,
- la liste de ces mutations qui comprend un item pour chaque mutation

3. Pour chacune des mutations de la zone :
   - le mot de la factorisation sur laquelle porte l'erreur, il est désigné par le numéro relatif du mot par rapport à l'erreur précédente,
   - le type de l'erreur, un code de taille fixe indique l'une des erreurs possibles.

La figure 3.3 illustre pour un cas de séquence compressée, les zones RTA du motif *TCG* identifiées dans la séquence, et la figure 3.4 montre le codage qui les remplace.

...tgctTCGTCCTCGTCCTCGTCATCATCGTCGTCGTCGTCGTCG
TCGTCGTCGTCGTCGTCGTCGTCATCGTCGTCATCGTCATCAT
CATCGTTgttgttttcgttcttatcttctgtatcttcttctTCGTCGTCGtct ...

FIG. 3.3 - *Exemple de zones RTA (en majuscule) dans une séquence.*

3; TCG; 1;
longueur du motif, le motif, nombre de zones moins 1

112; 29; 9;
zone 1: pos. relative dans la fenêtre, nb. d'occurrences du motif, nb. d'erreurs

(2,12);(2,12);(2,11);(2,11);(14,11);(3,11);(2,11);(1,11);(1,11);
sa liste d'erreurs: (numéro du motif de l'erreur 1, type de l'erreur 1); etc.

36; 3; 0;
zone 2: pos. relative (235 -112 - 29 * 3), nb. d'occurrences du motif, nb. d'erreurs (sans liste d'erreurs)

FIG. 3.4 - *Détail du codage des zones de la figure 3.3 (codes sous forme lisible en italique, explications du code en caractères gras ou normaux).*

3.3.4 Calcul du gain résultant du codage d'une zone.

Nous entendons par gain, la différence en nombre d'éléments binaires entre la représentation codée d'une zone et sa représentation originale. La version originale est considérée sous forme d'une séquence de lettres prises parmi 4, qui coûtent donc 2 bits chacune. On remplace la version originale par une version codée, si celle-ci est plus courte, le gain est positif, sinon il est négatif.

Les items concernant la fenêtre, comme ceux qui décrivent une zone précise hormis la liste d'erreurs, représentent un nombre fixe de bits. Par contre, comme voulu lors de la conception du codage, la liste des mutations d'une zone contient un nombre variable d'items indiquant chacun une erreur. Ainsi le code d'une zone est d'autant plus long que sa liste des erreurs l'est aussi.

Le gain d'une zone z, noté $GainZone(z)$, est une fonction linéaire de sa longueur en nombre de paires de bases (pb) notée n et de son nombre de mutations noté m. Pour l'algorithme dédié aux motifs trinucléotidiques, la fonction est :

$$GainZone(z) = 2n - 7m - 22$$

7 est la longueur en bits du code d'une mutation, que nous appelons aussi coût d'une mutation ; 2 est le nombre de bits nécessaires pour coder une base parmi 4. 22 est le coût fixe nécessaire au codage d'une zone.

Soient e la longueur du codage d'une erreur (e dépend de la longueur de motif considérée et vaut 7 pour les triplets) et C le coût fixe du codage d'une zone (il vaut 22 pour les triplets). On peut définir pour chaque mot appartenant à la factorisation d'une zone, un coût selon le nombre d'erreurs apportés par ce mot.

Définition 16 Pour tout $v \in Sim(u)$, nous définissons le coût imputé au gain d'une zone par le mot v :

- $Cout(v) = 0$ si $v = u$
- $Cout(v) = e.d(u, v)$ si $v \neq u$, où $d(u, v)$ représente le nombre de mutations élémentaires requises pour transformer u en v.

Pour les motifs de longueur 3 par exemple, $Cout(v) = 7$ si $v \in del(u) \cup sub(u)$ et $Cout(v) = Cout(l.a) = Cout(l) + 7$ si $v \in ins(u)$.

Définition 17 Soit z une zone RTA de motif u factorisée par la suite de mots $v_1.\cdots.v_p$, où tout $v_i \in Sim(u)$. On définit le gain de z par :

$$GainZone(z) = \sum_{i=1}^{p} (2|v_i| - Cout(v_i)) - C$$

3.4 Localisation des RTA.

Le but de l'algorithme présenté est la localisation dans un texte, de zones RTA compressibles pour un motif donné. L'objectif n'est pas de repérer tous les facteurs du texte qui sont des RTA, mais seulement ceux qui ressemblent significativement à une zone RTE du motif. La significativité de la ressemblance est mesurée par le gain de compression obtenu lors du codage du facteur. Si ce gain associé au facteur est positif, la ressemblance est considérée comme significative et la zone est incluse dans le codage. Dans le cas contraire, la zone n'est pas réellement ressemblante à une zone RTE, elle n'apparaît pas dans le résultat.

Une version codée du texte est produite en remplaçant les facteurs détectés par un code binaire (voir section 3.3). Il est donc inutile que les facteurs soient chevauchants. En effet, pour compresser une séquence, on remplace un facteur dont la séquence est régulière par une description codée plus courte que sa séquence elle-même. Des facteurs chevauchants conduisent à coder plusieurs fois une même portion du texte : ce qui est inutile et réduit le gain de compression.

3.4.1 Présentation de l'algorithme de recherche.

Dans tout le reste de la présentation de l'algorithme : u et t sont des textes sur A. L'algorithme recherche dans t des facteurs (nous employons aussi le terme «zones») RTA de motif u. Donc nous ne préciserons plus que toutes les zones RTE ou RTA sont donc de motif u. Nous introduisons la notation suivante pour nommer le résultat de l'algorithme de recherhe de zones RTA permettant la compression d'un texte.

Notation 5 Soient $1 \leq i < j \leq |t|$ les positions d'un facteur de t. On note Rech_RTA(t, u) le résultat de notre algorithme. $t[i, j] \in$ Rech_RTA(t, u) si :

- $t[i, j]$ est une RTA de motif u,
- $GainZone(t[i, j]) \geq 0$.
- il n'existe pas un facteur $t[k, l]$ inclus dans $t[i, j]$ tel que : $t[k, l]$ soit une RTA de motif u et $GainZone(t[k, l]) > GainZone(t[i, j])$.

3.4.2 Contrainte de gain pour une zone.

L'algorithme utilise cette contrainte de significativité pour limiter les facteurs à examiner. En effet, la suite de mots associée à un facteur détecté comme significatif, doit obligatoirement débuter et se terminer par un motif exact. Ceci est dû au coût de codage d'une erreur pour les petits motifs. Pour le motif GTA, l'exemple de la figure 3.5 illustre cette propriété qui découle du codage utilisé.

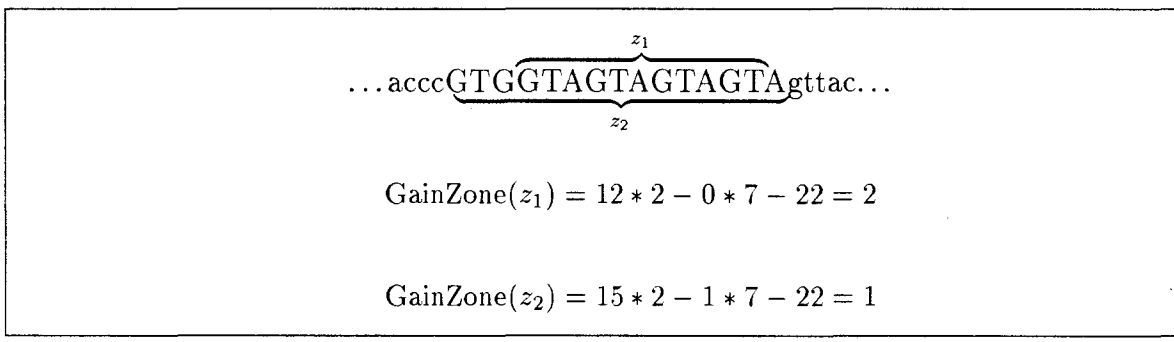


FIG. 3.5 - *$GainZone(z_1)$ est supérieur à $GainZone(z_2)$ car z_1 débute par un motif exact.*

Propriété 17 Soient t un texte sur A^*, u un mot sur A^* tel que $|u| \leq 3$. Soient $1 \leq i, j \leq |t|$ un couple de positions dans t. Si $t[i, j] \in$ Rech_RTA(t, u) alors :

$$t[i, i + |u| - 1] \quad = \quad t[j - |u| + 1] \quad = \quad u$$

Preuve Comme $t[i, j] \in$ Rech_RTA(t, u), on sait que $t[i, j]$ est une u-RTA, et donc que :

$$\exists\, p > 0, \exists\, v_1, \ldots, v_p \in A^* : \quad t[i, j] = v_1 \cdots v_p$$

et que :

$$\forall\, 0 < i \leq p : \quad v_i \in Sim(u)$$

Supposons que $v_1 \neq u$. Étant donné le coût d'une erreur par rapport au gain amené par un motif exact, nous savons que la factorisation de $t[i,j]$ doit contenir au moins une fois u, nous pouvons supposer dans un premier temps que $v_2 = u$. Dans ce cas $GainZone(t[i+|v_1|,j]) = GainZone(t[i,j]) - Cout(v_1)$, donc $t[i+|v_1|,j]$ est un meilleur facteur pour la compression de t. Le fait que celui-ci soit inclus dans $t[i,j]$ contredit l'hyposthèse que $t[i,j] \in$ Rech_RTA(t,u). Donc $v_1 = u$.

Si $v_2 \neq u$, on note k le plus petit indice tel que $v_k = u$. On peut alors appliquer le même raisonnement en remarquant que le facteur qui débute avec v_k est de meilleur gain que $t[i,j]$ et est inclus dans celui-ci.
La preuve est similaire pour prouver que $v_p = u$. □

En conséquence, seul un facteur de t compris entre deux occurrences de u dans t est susceptible d'appartenir à Rech_RTA(t,u). Plus généralement et toujours à cause du gain qui en résulte, un facteur qui appartient à Rech_RTA(t,u) est nécessairement compris entre 2 zones RTE maximales. Pour la suite de la présentation, nous englobons dans le terme RTE maximales, tous les facteurs de la forme u^p avec $p \geq 1$.

3.4.3 Explication de Rech_RTA(t,u).

Nous détaillons le schéma de l'algorithme.

Cas où t ne contient que 2 zones RTE maximales de motif u.
Considérons un texte t ne contenant que deux zones RTE maximales, notées respectivement r_1, r_2. Notons w le facteur qui les sépare (voir la figure 3.6). Si l'on arrive à factoriser w avec des mots de $Sim(u)$, il suffit de vérifier que $GainZone(r_1.w.r_2) > GainZone(r_1) + GainZone(r_2)$ et que $GainZone(r_1.w.r_2) > 0$ pour prouver que le facteur $r_1.w.r_2$ appartient à Rech_RTA(t,u). D'autre part, aucun facteur de $r_1.w.r_2$ est une zone RTA de gain supérieur à $GainZone(r_1.w.r_2)$, puisque seuls r_1 et r_2 sont envisageables et que l'on a vérifié que $GainZone(r_1.w.r_2) > GainZone(r_1) + GainZone(r_2)$. Pour simplifier la lecture, nous dirons alors que $r_1.w.r_2$ est le résultat de la «jonction» de r_1 et r_2.

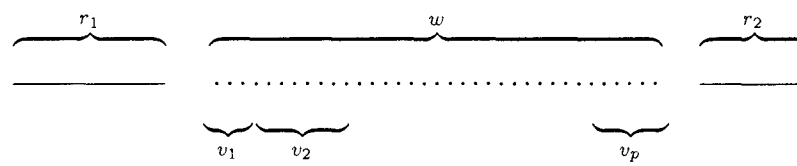


FIG. 3.6 - *Jonction de deux zones RTE, r_1 et r_2, par factorisation de w en $v_1, \ldots v_p$ (r_1, r_2 sont représentés par des traits continus et w par des pointillés).*

Cas général.
Supposons maintenant que t contiennent n zones RTE maximales, notées $r_1, r_2, \ldots, r_n$ dans l'ordre de leur apparition de gauche à droite dans t. Soit w un facteur de Rech_RTA(t,u). Par la propriété 17, nous savons que w commence et se termine par une zone RTE maximale, notons r_i et r_j avec $i < j$ (si $i = j$, le cas est simple puisque la zone RTA est réduite à une zone RTE maximale) les zones RTE maximales de ses extrémités. En outre, on sait que w est maximal, donc il n'existe pas dans Rech_RTA(t,u), un facteur commençant en r_{i-1} (respectivement se terminant au delà de r_j) et incluant w. On peut donc affirmer que r_i est une RTE maximale qui ne peut être jointe par la gauche.

Donc pour construire ce facteur w, il suffit d'exécuter l'algorithme suivant, qui effectue une **extension maximale de r_i par jonction à droite** (voir illustration par la figure 3.7) :

EXTENSION_MAX

Début
$w \leftarrow r_i$
suivante $\leftarrow$ i+1
TantQue (Jonction de w et $r_{suivante}$ possible
et $GainZone(w.w_{suivante-1}.r_{suivante}) > GainZone(w) + GainZone(r_{suivante})$) **Faire**
$w \leftarrow w.w_{suivante-1}.r_{suivante}$
suivante $\leftarrow$ suivante + 1
FinTantQue
Fin EXTENSION_MAX.

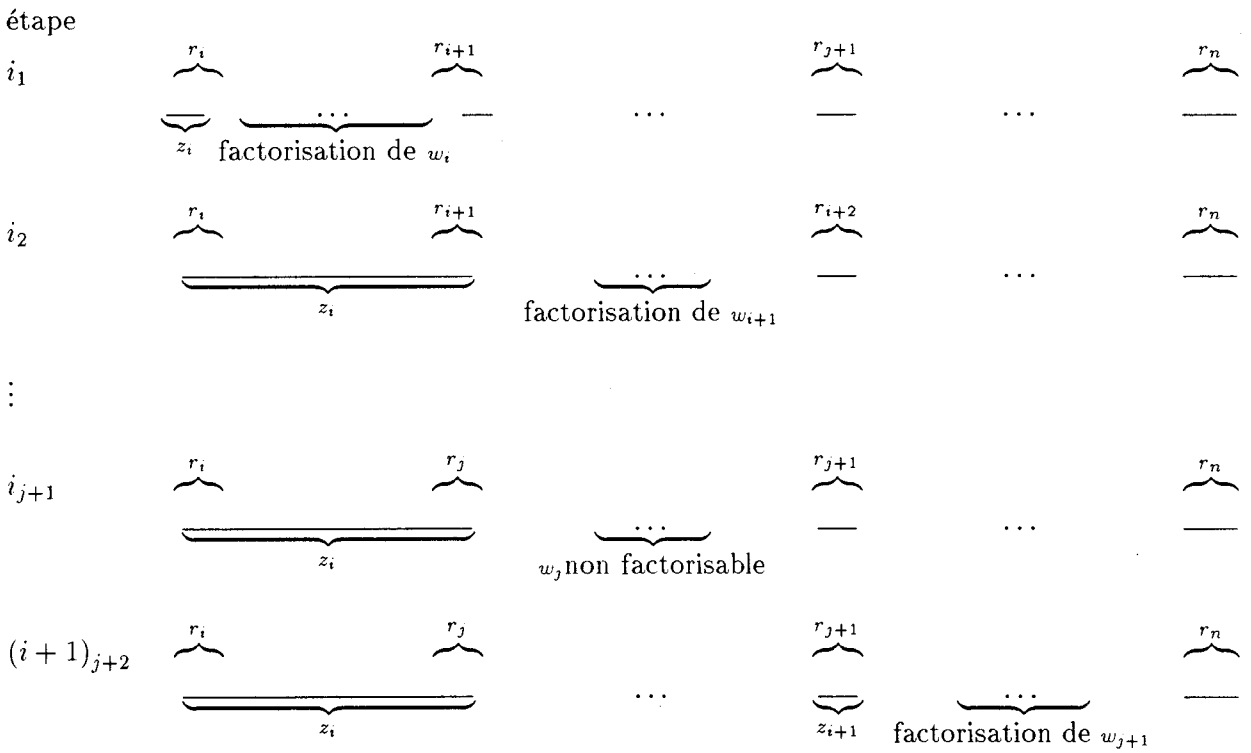


FIG. 3.7 - *Extension maximale de z_i (i.e. le w dans Extension_Max) par jonction à droite avec r_{i+1} (étape i_1), puis r_{i+2} (étape i), jusqu'à r_j (étape i_j). Ensuite la factorisation de w_j est impossible ce qui bloque l'extension de r_i. L'étape $(i+1)_{j+2}$ débute l'extension de z_{i+1} (appel de Extension_Max avec $w = z_{i+1}$).*

Ainsi pour calculer Rech_RTA(t, u), il suffit de procéder au même calcul pour toutes les zones RTE maximales susceptibles d'être un début de facteur de Rech_RTA(t, u), et ce dans l'ordre d'apparition dans le texte.

3.4.4 Procédure de Jonction.

Nous spécifions ici clairement la procédure de jonction de facteurs. On note $r_1, \ldots, r_n$ les différentes zones RTE maximales du texte t, dans leur ordre d'apparition. Soit dans le texte t, un facteur $w.w_i.r_{i+1}$ où w est une zone RTA (qui se termine par r_i), w_i est le facteur qui sépare r_i de r_{i+1}. On définit $Jonction(Res, w, r_{i+1})$ la procédure qui renvoie vrai si la jonction entre w et r_{i+1} est possible ou renvoie faux dans le cas contraire. Cette possibilité est déterminée en cherchant une factorisation de w_i avec des mots de $Sim(u)$. Si la factorisation est possible, la nouvelle zone RTA qui représente le facteur $w.w_i.r_{i+1}$, est mémorisée dans la variable résultat Res.

Le programme que nous utilisons définit un type de variable «ZoneRTA», capable de stocker la description complète d'une zone RTA : ses positions dans le texte, sa factorisation, son gain. Ici w et Res sont de ce type, de même dans le programme présenté ci-après, tout «facteur» ou toute «zone» en cours d'extension, et susceptible d'appartenir à Rech_RTA(t, u) est aussi de ce type.

3.4.5 Factorisation optimale et heuristique.

Pour que l'algorithme que nous avons détaillé construise effectivement Rech_RTA(t, u), il faut que la factorisation soit optimale vis à vis du gain de la zone. L'algorithme mis en œuvre utilise une procédure heuristique de factorisation, au lieu d'une procédure optimale de factorisation. Ceci est la seule heuristique que nous employons. Notons w_i le facteur de séparation, à factoriser. La factorisation s'opère comme suit :

- Si $|w_i| \leq TailleMaxSim$ alors l'algorithme recherche une factorisation optimale en temps constant,
- Sinon, il emploie un algorithme glouton de factorisation qui ne remet jamais en cause un mot de $Sim(u)$ déjà inclus dans la factorisation. Cet algorithme est donc linéaire par rapport à $|w_i|$. Il favorise les mots de $Sim(u)$ les moins coûteux et ceux qui n'apportent pas de décalage dans la répétition (i.e. il défavorise les insertions ou les délétions au profit des substitutions). Cependant, on peut faire un mauvais choix de mot qui bloque la factorisation, mais ceci est rare.

Dans tous les cas, la complexité de la factorisation de w_i est linéaire par rapport à $|w_i|$, donc la complexité de $Jonction$ appliquée à deux zones séparées par w_i, l'est aussi.

3.4.6 Présentation de l'algorithme Rech_RTA(t, u) en pseudo-code.

L'algorithme Rech_RTA(t, u) emploie une procédure de recherche des RTE maximales qui se déroule en une passe sur t. Elle n'est pas détaillée ici. Le tableau $Supprimee$ sert à noter quelles zones sont insérées dans une zone plus à gauche lors de l'extension.

3.4.7 Complexité de l'algorithme Rech_RTA(t, u).

Propriété 18 Rech_RTA(t, u) est de complexité linéaire par rapport à la taille de t.

Preuve Supposons qu'il y ait n zones RTE maximales dans t, notées $r_1, \ldots, r_n$ et notons $w_1, \ldots, w_{n-1}$ l'ensemble des facteurs qui séparent deux zones RTE maximales successives.

Premièrement, la procédure *Recherche_RTE_max* effectue un parcours du texte t pour associer une zone RTE maximale, à chaque bloc de répétitions contiguës de u. La taille du motif étant fixée, *Recherche_RTE_max* est linéaire. Il en est de même pour l'initialisation du tableau *Supprimee*.

La complexité de Rech_RTA(t, u) provient de celle de la boucle d'extension maximale par jonction à droite (i.e. le «tant que» le plus externe). La procédure complexe est le calcul d'une jonction entre deux zones toujours successives, qui est linéaire par rapport à la taille du facteur qui sépare ces deux zones (voir sous-section 3.4.5). Le processus d'extension maximale par jonction à droite d'une zone d'indice i est illustré par la figure 3.7. Pour une zone d'indice i donnée, la jonction entre $Zones[i]$ et $Zones[suiv]$ n'est calculée qu'une fois, puisque :

- si elle est possible, $Zones[i]$ est agrandie de manière à inclure $Zones[suiv]$ qui est supprimée du tableau des zones, et l'indice *suiv* est incrémenté de un ;
- sinon l'extension par jonction à droite de $Zones[i]$ s'arrête.

Lors de l'arrêt de l'extension par jonction de $Zones[i]$, i prend la valeur de *suiv* et donc $Zones[suiv]$ devient la prochaine zone pour laquelle on effectue une extension par jonction à droite. Ainsi, on va procéder à des calculs de jonction, uniquement à droite de $Zones[suiv]$ qui n'ont jamais été testés avant. Donc Rech_RTA(t, u) calcule dans le pire des cas, une jonction entre chaque paire de zones successives, nous avons alors :

$$\begin{aligned} Complexite(\text{Rech\_RTA}(t,u)) &\leq \sum_{i=1}^{NbZones-1} Complexite(Jonction(Zones[i], Zones[i+1])) \\ &\leq \sum_{i=1}^{NbZones-1} |w_i| \\ &< |t| \end{aligned}$$

□

```
RECH_RTA (t, u)
Var
   entiers : i, suiv, NbZones;
   booléens : JonctionPossible, Supprimee[];
   ZoneRTA : Zones[];

Début
   \\Initialisation du tableau de Zones :
   \\chaque élément représente une zone RTE maximale
   \\dans l'ordre d'apparition ; renvoie le NbZones initiales
   NbZones ← Recherche_RTE_max(Zones[], t, u);

   \\Initialisation de Supprimee
   Pour i allant de 1 à NbZones Faire
      Supprimee[i] ← faux;
   FinPour
   \\Parcours du tableau de Zones
   i ← 1;
   TantQue (i ≤ NbZones-1) Faire
      suiv ← i+1;
      JonctionPossible ← vrai;

      \\extension maximale par jonction à droite de Zones[i]
      TantQue ((suiv ≤ NbZones) et JonctionPossible) Faire
         JonctionPossible ← Jonction(Res, Zones[i], Zones[suiv]);
         Si (JonctionPossible et
            GainZone(Res) ≥ (GainZone(Zones[i]) + GainZone(Zones[suiv]))) Alors

            \\on mémorise la jonction et on passe a la zone suiv
            Zones[i] ← Res;
            Supprimee[suiv] ← vrai;
            suiv ← suiv+1;
         Sinon
            i ← suiv;
            JonctionPossible ← faux;
         FinSi
      FinTantQue
   FinTantQue
   renvoyer la liste des Zones non supprimées de gain > 0;
Fin RECH_RTA (t, u).
```

ALGO. 6 - *Algorithme Rech_RTA*(t, u) *: recherche des RTA pour le motif* u *dans le texte* t.

Chapitre 4

Identification de zones RTA dans les chromosomes de la levure.

L'expérimentation a permis d'obtenir des données biologiques intéressantes sur les répétitions en tandem, dont l'analyse approfondie est l'objet d'une collaboration avec une équipe de statisticiens-biologistes du CGM (Centre de Génétique Moléculaire de Gif-sur-Yvette). Ces données sont le centre de ce chapitre. Son contenu précise clairement le protocole expérimental et le matériel biologique employés, puis présente un aperçu des différents résultats. Nous situons notre travail par rapport à une étude statistique menée au CGM et à des tests sur des séquences pseudo-aléatoires. Une discussion biologique fruit de notre collaboration conclut cette étude.

Ce chapitre montre l'aboutissement d'une recherche informatique sur de réelles applications biologiques. Cette présentation des résultats montre combien un effort expérimental apporte des informations pertinentes pour l'analyse de séquences d'ADN. On y voit aussi que la compression est une approche analytique performante qui s'intègre au sein de la biologie computationnelle plus classique, car elle amène un regard informationnel neuf sur la notion de significativité.

4.1 Matériel biologique et protocole expérimental.

4.1.1 Matériel biologique.

Le matériel biologique présente les séquences sur lesquelles portent les expériences. Nos algorithmes de recherche de répétitions en tandem approximatives ont été appliqués aux séquences de quatre chromosomes complets de levure de boulanger, i.e. Saccharomyces cerevisiae [FAA+94, OdAAC+92, Ja94, DAAa94]. Il s'agit des chromosomes II (807188 pb), III (315357 pb), VIII (562638 pb) et XI (666448 pb) qui constituent un total de 2,351,631 pb, soit environ 17.42% de l'ADN de la levure. Ce choix se justifie par les réponses apportées aux deux questions suivantes.

Pourquoi l'organisme de la levure?

Plusieurs raisons importantes ont déterminé ce choix. Tout d'abord, la levure est un organisme modèle pour la génétique. En effet, certains «petits» génomes sont l'objet d'un programme de séquençage complet, pour permettre de mieux mener la tâche herculéenne

du séquençage et de l'analyse du génome humain. Par ailleurs, la taille et la complexité moindres de ces petits génomes ont autorisé des analyses très poussées d'où proviennent une part importante des connaissances génétiques actuelles (voir [Dan93]). Ceci implique que la génétique de ces organismes est mieux connue, donc l'apport d'informations sur la répartition du DOS-DNA dans leurs chromosomes est plus pertinent. En outre, la levure est l'organisme eucaryote dont le séquençage est le plus avancé et le seul pour lequel ont dispose de la séquence de quatre chromosomes.

Le fait que le chromosome soit entier est important, c'est une entité biologique réelle, alors qu'une longue séquence, même contenant plusieurs gènes, n'en est pas une : pour la cellule, elle est intégrée à un chromosome. En outre, le chromosome a une organisation propre : des extrémités appelées télomères, une partie centrale ou centromère et des séquences intermédiaires (voir schéma 4.1).

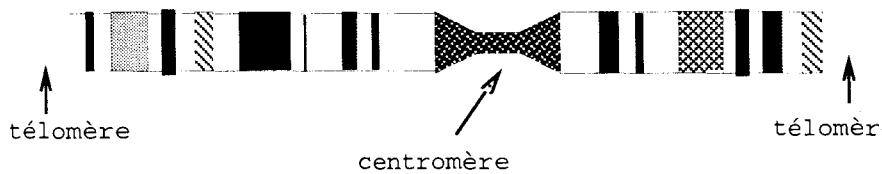


FIG. 4.1 - *Organisation du chromosome : schéma d'un chromosome coloré et arrêté en métaphase. Sont visibles les bandes de colorations différentes, ainsi que le centromère et les extrémités ou télomères.*

Pourquoi expérimenter sur la levure alors que le phénomène d'expansion de triplets a été découvert chez l'homme?

L'objectif plus général de notre étude est la répartition du DOS-DNA et son importance sur l'organisation du chromosome. Rien dans les mécanismes avancés jusque ici, ne laisse penser que ce phénomène est confiné à la génétique humaine. En plus, la présence universelle de répétitions en tandem, fait de la levure un bon sujet pour l'étude de leur répartition. Rappelons qu'en outre, nous ne disposons d'aucune séquence de chromosome humain. Par ailleurs, S. cerevisiae est un organisme eucaryote comme l'homme.

4.1.2 Protocole expérimental.

Nous détaillons ici la procédure d'expérimentation appliquée aux séquences des chromosomes de la levure.

Détail du protocole.

Avant de subir le test, la séquence chromosomique est découpée en fenêtres consécutives de 500 nucléotides. Ce chiffre se justifie par le fait que les répétitions en tandem sont des structures locales et par la volonté de comparer nos résultats à une étude statistique menée par M-O. Delorme, A. Hénaut et E. Ollivier, qui utilise cette taille de fenêtre sur les mêmes chromosomes (voir [ODH95]).

Pendant une expérience, l'algorithme choisi est appliqué à chaque fenêtre. Le chromosome subit trois expériences indépendantes : une par taille de motif (longueur 1, 2 ou 3). En effet,

l'algorithme Rech_RTA(t, u) est donné pour une longueur de motif fixée, nous avons donc implanté trois programmes, un par longueur de motif. En outre, pour ne pas étudier les fenêtres d'un chromosome dans une seule «phase», nous avons doublé chaque expérience avec le début de la première fenêtre du chromosome, positionné au nucléotide 250.

Pour une fenêtre et une longueur de motif fixés, le motif le plus courant est choisi après un comptage à toutes les positions, puis on applique Rech_RTA(t, u), où t est la séquence de la fenêtre et u le motif déterminé. Pour les longueurs de motif 2 et 3, les motifs constitués par un «run»[1] de mononucléotides, tels que AAA sont exclus. Le compresseur dédié aux répétitions de mononcléotides s'en occupe.

Lors de l'utilisation des algorithmes deux types de sorties sont disponibles en option. Une sortie «rudimentaire» sous forme de tableaux, où les informations sur les zones trouvées apparaissent codées par des entiers ; une sortie «lisible» reprenant toutes ces informations et visualisant la séquence dans laquelle les zones RTA et leurs erreurs sont mises en évidence. La figure 4.5 en donne un exemple.

Le gain d'une fenêtre contenant Z zones RTA est définit par la formule ci-dessous qui dépend du gain de chaque zone. Cette formule compare simplement la taille originale de la fenêtre avec sa taille compressée **en bits**. Celle-ci comprend le coût du codage de chaque zone, la séquence restante (notée *Seq_Restante*) où chaque base est codée sur 2 bits et un coût global de déclaration du format de la fenêtre (noté *Cout_Declaration*). La formule est :

$$Gain = 1000 + \sum_{i=1}^{Z} GainZone(z_i) - 2 * |Seq\_Restante| - Cout\_Declaration$$

4.2 Quantité, répartition et nature des zones identifiées.

Cette section présente les résultats bruts des expérimentations, puis les compare à ceux d'une expérimentation statistique.

4.2.1 Nature et importance des zones RTA.

Nous examinons les résultats globaux des expériences, vus au niveau des chromosomes complets sans entrer dans le détail d'une zone. Nous donnons une vision quantitative du nombre de fenêtres compressées, ainsi que leur répartition le long du chromosome. Nous montrons un exemple de tableau synoptique des zones trouvées dans un chromosome avec les zones codantes chevauchées, puis une liste des motifs utilisés. Nous terminons par les résultats d'autres expériences, dont les tests des algorithmes sur des séquences de procaryotes et sur des chaînes de Markov.

Nous appelons *expérience* le test d'un algorithme pour une longueur de motif sur un chromosome. Le *phénomène* étudié est la présence de répétitions en tandem de courts motifs. Du point de vue informationnel, le terme de *signal* est pris dans le sens physique. Un signal devient une *information* si l'entité qui le reçoit est capable de le décoder ou de le comprendre pour éventuellement le prendre en compte.

1. «run» est le terme biologique consacré pour un répétition en tandem d'un oligonucléotide court, souvent un mononucléotide.

Tableaux comparatifs du nombre de fenêtres dans les chromosomes.

Le tableau 4.1 récapitule le nombre de fenêtres examinées au total, le nombre de fenêtres compressées et le ratio des deux valeurs précédentes. Le tableau 4.2 est divisé en trois groupes de colonnes, une par expérience. Chaque ligne détaille les résultats pour un des chromosomes. La colonne «Comp» donne le nombre de fenêtres comprimées, «% comp» calcule le ratio de ce nombre par rapport au nombre total de fenêtres compressées toutes longueurs de motif confondues, tandis «% total» montre celui par rapport au total des fenêtres examinées. Le tableau 4.3 a la même structure et montre les gains moyens et maximaux par chromosome et par expérience.

Récapitulatif			
Chromosome	Total	Compressées	Ratio
2	1614	140	8.67
3	630	62	9.84
8	1124	109	9.69
11	1332	120	9.00

TAB. 4.1 - *Récapitulatif du nombre de fenêtres sur l'ensemble des chromosomes.*

Chr.	Motifs de longueur 1			Motifs de longueur 2			Motifs de longueur 3		
	Comp.	% comp	% total	Comp.	% comp	% total	Comp.	% comp	% total
2	107	76.4	6.29	11	7.8	0.68	22	15.6	1.36
3	42	67.7	6.66	11	16.6	1.74	9	14.5	1.42
8	76	69.7	6.76	16	14.6	1.42	17	15.6	1.51
11	84	70.0	6.30	14	11.6	1.05	22	18.3	1.65

TAB. 4.2 - *Nombre de fenêtres compressées dans les quatre chromosomes.*

Récapitulatif			Motifs de lg 1		Motifs de lg 2		Motifs de lg 3	
Chr.	Moy.	Max.	Moy.	Max.	Moy.	Max.	Moy.	Max.
2	11.5	121	7	37	15	50	32	121
3	14	298	9	65	37	298	9	30
8	12.2	144	8	46	26	144	18	65
11	14	292	7	46	19	89	38	292

TAB. 4.3 - *Gain moyen et maximal par chromosome.*

Tout d'abord ces tableaux contrastent fortement avec ceux résumant les expériences sur les chaînes de Markov (cf. la fin de cette sous-section). Ni la quantité de fenêtres, ni les gains de compression sont comparables. *En outre, ces tableaux démontrent une remarquable uniformité de la présence de répétitions en tandem indépendamment du chromosome étudié.* Le phénomène touche globalement entre 8.5 et 10% des fenêtres, quel que soit le chromosome. De même, le phénomène est plus fort pour les motifs de longueur 1 que pour les autres longueurs : autour de 70% des fenêtres compressées. Les répétitions d'autres motifs sont moins

courantes mais provoquent des gains plus forts que pour les motifs de longueur 1 : sauf dans un cas, le gain moyen pour les longueurs de motif 2 et 3, est le double de celui pour les motifs de longueur 1. En outre, la fenêtre de gain maximal a toujours été comprimée par les algorithmes pour les motifs de longueur 2 ou 3.

La présence de répétitions en tandem est une caractéristique globale de l'ADN de levure. Le processus de création de ces répétitions ignore l'entité chromosome. En termes informationnels, la création de DOS-DNA n'est pas concernée par ce signal : le processus de création s'applique de la même manière quel que soit le chromosome.

Une autre remarque est la disparité du phénomène selon la longueur du motif. En effet, la présence de répétitions en tandem de motif de longueur 1 est courante mais engendre de faibles gains. Les signaux sont quantitativement nombreux mais faibles en moyenne. Par contre, les RTA des motifs plus grands sont plus rares mais permettent de gagner plus de bits. Pour ces deux longueurs, les signaux sont donc plus rares et plus forts.

Répartition des fenêtres compressées dans les chromosomes.

Pour chaque chromosome et pour chaque longueur de motif étudiés, nous réalisons les graphiques représentant toutes les fenêtres compressées d'un chromosome. Une fenêtre y est représentée par un trait vertical dont la hauteur égale son gain. Ce trait est placé à l'abcisse qui vaut la position de début de la fenêtre. Nous montrons ici les 3 graphiques pour le chromosome 11 de la levure, voir figures 4.2, 4.3 et 4.4. La répartition a la même allure générale pour les autres chromosomes. L'échelle du gain en ordonnée n'est pas conservée entre les tableaux.

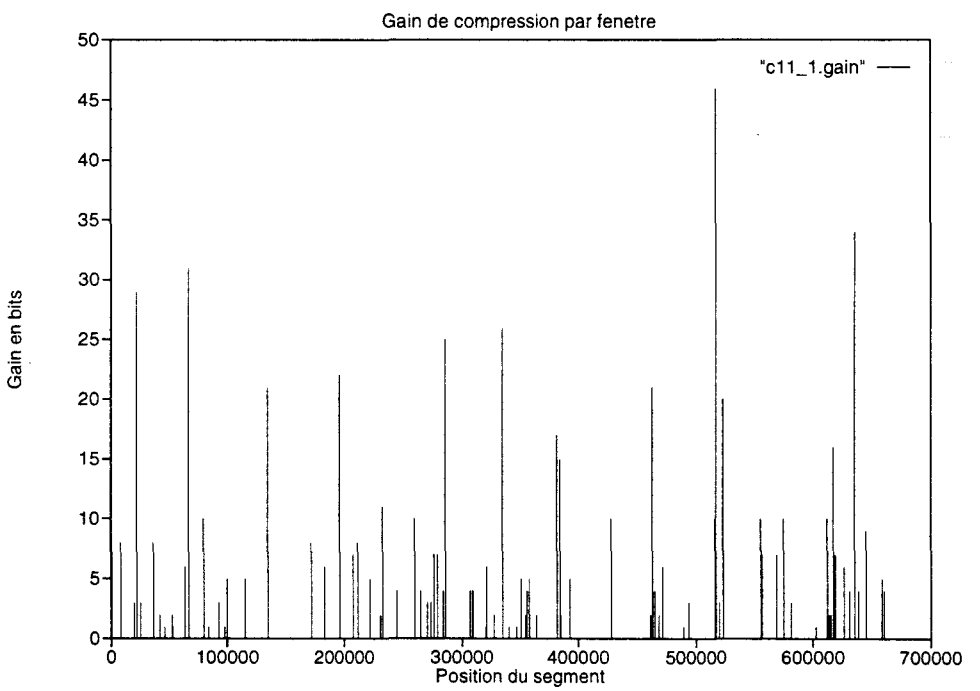


FIG. 4.2 - *Répartition des fenêtres compressées le long du chromosome 11 par l'algorithme pour les* **mononucléotides**.

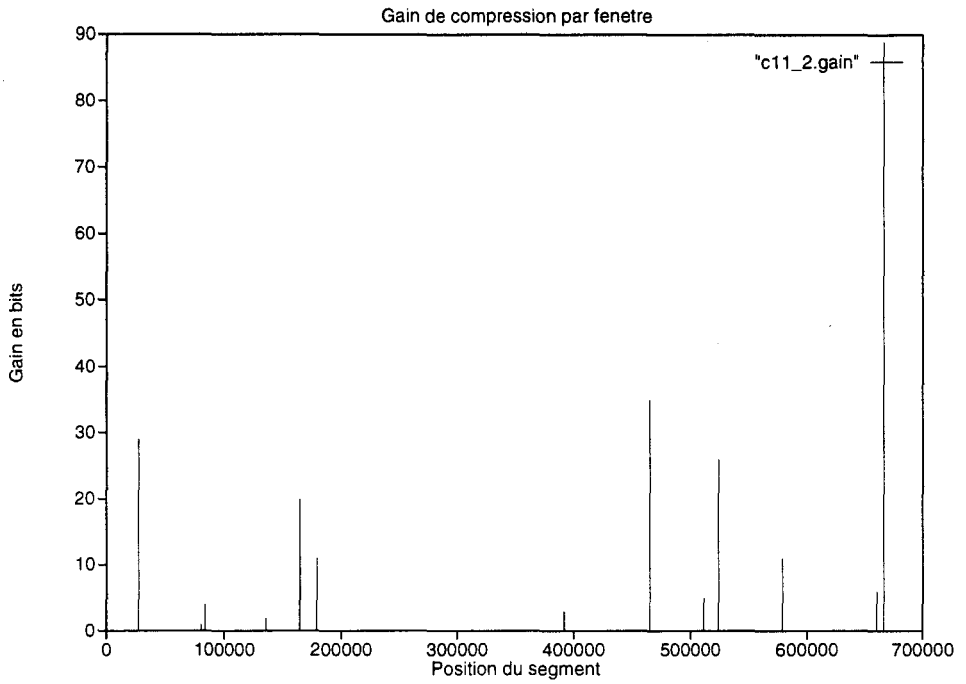


FIG. 4.3 - *Répartition des fenêtres compressées le long du chromosome 11 par l'algorithme pour les* **dinucléotides**.

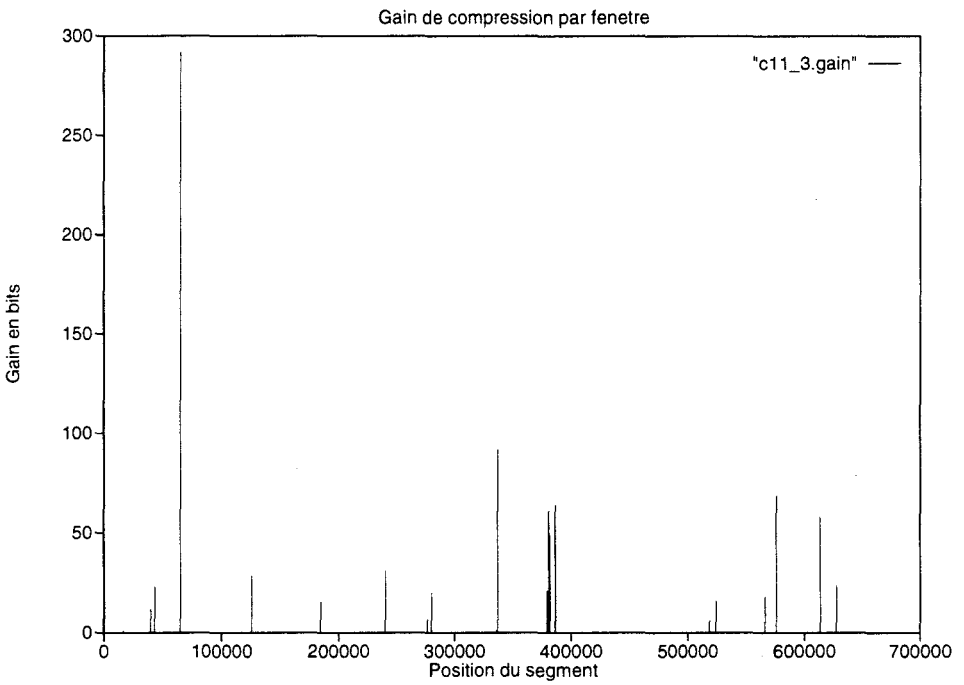


FIG. 4.4 - *Répartition des fenêtres compressées le long du chromosome 11 par l'algorithme pour les* **trinucléotides**.

Tout d'abord les graphiques montrent l'abondance quantitative de RTA de motif de longueur 1. Pour cette longueur, les fenêtres sont dispersées le long du chromosome, ne laissant pas de grands segments sans aucune répétition : l'écart maximal entre deux fenêtres consécutives ne dépasse pas 40000 pb et la moyenne se situe au dessous de 10000 pb. Pour les autres longueurs de motifs, la répartition des fenêtres ne montre aucune structure apparente.

Interaction avec les phases codantes.

Pour chaque chromosome et pour chaque longueur de motif, nous produisons un tableau récapitulatif des zones trouvées, car biologiquement une fois la fenêtre détectée, seules les zones RTA sont pertinentes. Ces tableaux permettent de localiser la zone dans le chromosome et d'en connaître toutes les caractéristiques, notamment un éventuel chevauchement avec les phases codantes ou «Open Reading Frames» (ORF). Nous montrons simplement le tableau correspondant au chromosome 2 pour les motifs de longueur 3 dans le tableau 4.4.

Contenu du tableau.

Parmi les colonnes du tableau, 3 sont consacrées à la fenêtre : position dans le chromosome, gain global et valeur du test du χ^2 (voir la sous-section 4.2.2). 4 autres décrivent une des zones de la fenêtre : position de début, longueur en pb, gain de la zone, pourcentage de motifs erronés (i.e. nombre de motifs contenant au moins une erreur sur le nombre de mots trouvés lors de la factorisation de la zone). Les 4 colonnes suivantes indiquent une éventuelle intersection de la zone avec une ORF, elles précisent : le brin qui contient l'ORF **W**atson ou **C**rick, le nom de l'ORF, la longueur de l'intersection (colonne $\cap$, le signe est négatif si la zone commence avant l'ORF sur le brin, positif sinon), la colonne φ indique le fait que la début de la répétition soit en phase avec le début de l'ORF :

- 1 signifie le respect de la phase,
- 0 le non-respect,
- et -1, que la question n'a pas de sens (longueurs de motifs 1 et 2).

La dernière colonne donne le motif de la fenêtre, son complémentaire est donné si la zone chevauche une ORF sur le brin Crick, de même dans ce cas, la position de la colonne 4 est la position du dernier nucléotide de la zone (i.e. la position du début de la zone complémentaire).

Ce tableau, outre qu'il constitue un récapitulatif complet, est intéressant parce qu'il met en évidence le chevauchement des zones RTA et des ORFs ou phases codantes. En fait, il renseigne sur l'interaction de deux phénomènes : la présence de répétitions en tandem qui sont des séquences d'ADN plutôt simples et les séquences codantes qui sont très contraintes. Dans ce tableau, nous voyons que 31 zones sur 35 chevauchent une phase codante ! C'est à dire plus de 90% des zones, alors qu'environ 70% de l'ADN de levure est codant. On peut donc dire que malgré de nombreuses contraintes imposées par la sélection sur les séquences codantes, celles-ci peuvent contenir des RTA de triplets.

Motifs utilisés dans les répétitions en tandem.

Cette sous-section montre le récapitulatif des motifs utilisés dans les fenêtres, chromosome par chromosome. Plusieurs faits sont observables sur la sélection des motifs répétés en tandem :

- Les mononucléotides A et T sont les seuls à être répétés, le biais de composition de l'ADN de levure (64.2% de $A + T$) semble être une explication suffisante, puisqu'on

Fenêtre			Zone				Interaction ORF				Motif
Pos.	Gain	χ^2	Début	Lg	Gain	% ME	Brin	Protéine	$\cap$	φ	
42500	8	45	42509	41	19	42	W	-	0	-1	TTC
62000	97	10	62386	69	108	4	C	YBL084c	-69	0	TAA
66000	84	17	66101	12	6	0	W	YBL081w	-12	0	CAA
66000	84	17	66383	63	89	9	W	YBL081w	-63	0	CAA
67000	1	7	67021	48	12	56	W	-	0	-1	ATT
113500	3	30	113593	21	2	42	W	YBL053w	-21	0	TTC
113500	3	30	113613	21	2	42	C	YBL052c	-21	1	GAA
113500	3	30	113999	15	12	0	C	YBL052c	-15	0	GAA
127000	13	53	127217	18	3	33	W	YBL046w	-18	1	GAT
127000	13	53	127268	24	15	25	W	YBL046w	-24	1	GAT
127000	13	53	127307	27	6	44	W	YBL046w	-27	1	GAT
160000	70	15	160366	69	81	21	W	YBL029w	-69	1	AAT
199500	48	26	199746	54	59	22	W	YBL011w	-54	1	GAA
206500	2	43	206757	18	10	16	C	YBL007c	-18	1	CAA
206500	2	43	206943	18	3	33	C	YBL007c	-18	1	CAA
264000	1	21	264389	15	12	0	W	YBR016w	-15	1	CAG
266500	28	43	266563	30	33	10	C	YBR017c	-30	0	ATG
266500	28	43	266674	12	6	0	C	YBR017c	-12	0	ATG
345000	1	36	345352	15	12	0	W	-	0	-1	TGT
449500	4	53	449667	21	9	28	W	YBR108w	-21	1	CAA
449500	4	53	449763	27	6	44	W	YBR108w	-27	1	CAA
458000	121	136	458106	93	132	12	C	YBR112c	-93	1	CAA
459500	29	49	459690	42	28	35	W	YBR113w	-42	1	GTT
459500	29	49	459731	42	28	35	C	YBR112c	-42	0	AAC
459500	29	49	459802	30	12	40	W	YBR113w	-30	0	GTT
499000	20	25	499209	54	31	44	W	YBR135w	-54	1	CAA
535000	52	42	535459	51	11	58	C	YBR150c	-51	0	TGA
535000	52	42	535498	36	52	0	C	YBR150c	-36	0	TGA
535500	33	12	535605	36	10	50	C	YBR150c	-36	1	AAT
535500	33	12	535641	27	34	0	C	YBR150c	-27	1	AAT
642500	5	26	642734	21	16	14	W	YBR212w	-21	1	CAA
754500	5	20	754690	21	16	14	W	YBR278w	-21	0	AGA
773500	23	26	773853	27	34	0	W	YBR289w	-27	1	CAA
774000	75	191	774342	135	86	51	W	YBR289w	-135	1	CAA

TAB. 4.4 - *Tableau descriptif de toutes les zones RTA pour le chromosome 2 et les motifs de longueur 3.*

Motifs		Nombre de fenêtres				
		Chr. 2	**Chr. 3**	**Chr. 8**	**Chr. 11**	**Total**
A	T	143	62	100	122	427
C	G	0	1	0	0	1
AT	TA	2	1	0	1	4
CT	AG	0	0	1	0	1
AT	TA	8	9	12	11	40
AC	GT	1	1	3	2	7
TTC	GAA	5	2	1	3	11
TTA	TAA	1	0	3	0	4
TTG	CAA	11	2	4	11	28
TCT	AGA	1	2	5	4	12
TCC	GAA	0	0	0	1	1
TCA	TGA	2	2	0	2	6
TAT	ATA	0	0	1	5	6
TGT	ACA	1	0	3	0	4
CTT	AAG	0	0	0	1	1
CTC	GAG	0	0	0	1	1
CTG	CAG	1	2	2	0	5
CAT	ATG	2	0	1	1	4
CGT	ACG	0	0	0	1	1
ATT	AAT	4	1	1	0	6
ATC	GAT	3	1	1	0	5
AAC	GTT	3	0	0	3	6
AGC	GCT	0	0	0	1	1

TAB. 4.5 - *Récapitulatif des motifs utilisés. Un motif et son complémentaire sont associés, pour chaque couple la ligne indique le nombre de fenêtres ayant ce motif pour chaque chromosome.*

observe la même sélection de motifs pour les chaînes de Markov. Les deux motifs se partagent équitablement les fenêtres.

- La sursélection de AT est aussi flagrante: 40 fenêtres sur 52, soit 77%. Cependant le motif complémentaire n'apparaît pas aussi souvent. Bien qu'il s'agisse de répétitions en tandem, cette remarque n'est pas anodine, puisque le test choisit le motif le plus courant à toutes les positions.

- Pour les trinucléotides, 17 couples sont utilisés sur 30 (30 parce que les triplets AAA, CCC, GGG et TTT ne sont pas recherchés).

Autres résultats.

Expérimentations avec des fenêtres décalées.
Les chromosomes ont subit les même expériences avec les fenêtres décalées de 250 pb par rapport au début de la séquence. Ce changement dans les expérimentations ne perturbe pas les résultats: ni dans la quantité de fenêtres observées, ni dans leur répartition le long du chromosome, ni dans les gains moyens observés. En effet, la plupart des zones sont retrouvées dans une fenêtre décalée, avec un gain sensiblement égal, parfois legèrement plus fort ou plus faible. Les résultats ne sont pas détaillés.

Expérimentations sur des chaînes de Markov.
Pour valider la significativité de nos résultats sur les séquences réelles, nous avons effectué les mêmes expériences sur des chaînes de Markov générées en tenant compte des caractéristiques de chaque chromosome grâce à un modèle d'ordre 1. Pour chaque chromosome, nous construisons une chaîne de Markov 10 fois plus longue, à partir de la matrice des transitions d'un nucléotide vers un autre. Cette matrice provient directement du décompte des dinucléotides à toutes les positions dans la séquence. Les résultats sont sans comparaison avec ceux des expériences commentées plus haut:

- aucune répétition en tandem de di- ou trinucléotides n'a été détecté,

- seul l'algorithme dédié au motif de longueur 1 détecte des zones RTA dans environ 0.35% des fenêtres, contre plus de 6% dans les vrais chromosomes,

- le gain moyen est de 2 bits et le gain maximal est de 9 bits, sauf dans le chromosome 11 où il atteint 12 bits,

- seul les motifs A et T sont sujets à répétitions.

Expérimentations sur des séquences de procaryotes.
En outre, le même protocole fut appliqué à de longs contigs provenant de deux organismes procaryotes modèles: Escherichia coli et Bacillus subtilis. Les résultats obtenus sont comparables à ceux des chaînes de Markov pour E. coli. Pour le bacille, le nombre de fenêtres est plus fort ainsi que le gain maximal, sans pour autant atteindre les résultats de la levure. Le détail est dans le tableau 4.6.

Organisme	Nombre de fenêtres			Gain		Motifs
	Total	Comprimées	Ratio	Moyen	Maximal	
B. subtilis	660	10	1.51%	1.5	11	A ou T
E. coli	1025	2	0.19%	2	2	A ou T

TAB. 4.6 - *Récapitulatif des résultats chez E. coli et B. subtilis.*

4.2.2 Comparaison méthodologique entre compression et méthode du χ^2.

Le DOS-DNA de l'ADN de levure fut aussi étudié au Centre de Génétique Moléculaire (C.G.M.) de Gif-sur-Yvette, par M-O. Delorme, A. Hénaut et E. Ollivier. La méthode mise en place est un test de χ^2 appliqué aux fenêtres consécutives de 500 pb. Un de nos objectifs fut de confronter nos résultats à ceux de cette méthode statistique.

Il existe un biais biologique connu de l'utilisation des dinucléotides dans les séquences. Leur fréquence ne correspond pas au produit des fréquences des nucléotides qui les composent. L'idée développée au CGM est d'analyser l'évolution locale de ce biais sur des fenêtres de 500 pb, dans le but de les classifier et d'identifier des disparités locales du biais. Dans ce cadre, un biais fort signifie une constitution de la fenêtre basée sur quelques dinucléotides et donc plutôt simple, tandis qu'un biais faible donne une répartition d'allure aléatoire des dinucléotides, qui est plus complexe à appréhender. Il s'agit comme pour une méthode basée sur la compression, de tester des hypothèses sur un processus de création du DOS-DNA.

Le test utilisé est un calcul de χ^2 que nous allons présenter, puis nous comparons les résultats et les méthodes. L'article de référence pour la méthode du χ^2 est [ODH95].

Présentation du test de χ^2.

Le test du χ^2 évalue *le biais compositionnel en dinucléotides* de la séquence de 500 pb sur laquelle il est appliqué. Toutes les fenêtres consécutives de cette taille subissent ce test. Il calcule pour chaque dinucléotide son biais d'apparition, par la différence entre la fréquence observée et la fréquence attendue s'il n'y avait pas de biais. N_i dénote le nombre d'occurrences observées du nucléotide i, et $N_{i,j}$ la fréquence observée du dinucléotide ij, i.e. le nombre d'occurrences de ij à toutes les positions, divisé par 499. De plus, on note N la taille de la séquence et on a :

$$N = \sum_{i=A}^{T} N_i \tag{4.1}$$

Sans biais compositionnel, ij apparaîtrait avec la même probabilité que l'événement : i est suivi de j dans la séquence. La fréquence attendue de ij s'il n'y avait pas de biais, notée $n_{i,j}$, est le produit du nombre d'apparitions de i par celui de j, sur le nombre total de nucléotides :

$$n_{i,j} = \frac{N_i * N_j}{N} \tag{4.2}$$

Le critère numérique du χ^2 pour la fenêtre, est la somme pour tous les dinucléotides de la différence entre fréquences observée et attendue, au carré, normalisée par la fréquence attendue :

$$\chi^2 = \sum_{i=A}^{T} \sum_{j=A}^{T} \frac{(N_{i,j} - n_{i,j})^2}{n_{i,j}} \tag{4.3}$$

Lorsque le calcul est effectué, il faut déterminer à partir de quelle valeur du critère le biais de la fenêtre est significatif. Dans l'expérience menée au CGM, *une fenêtre dont le χ^2 dépasse le seuil de 45 est considérée comme exceptionnelle* pour son biais compositionnel en dinucléotides.

Principaux résultats de la méthode du χ^2.
Les fenêtres exceptionnelles dont le χ^2 est supérieur à 45 sont appelées DOS-DNA et constituent 2% des fenêtres des quatre chromosomes de levure. Suite à l'observation minutieuse de ces fenêtres, on relève trois faits marquants:

1. La plupart de ces fenêtres contiennent de nombreuses répétitions de courts motifs tels que GT, TA et CAT déjà observés chez S. cerevisiae, mais aussi du motif CAG. Par contre le tandem $(CGG)^n$ n'apparaît pas.

2. Les fenêtres exceptionnelles sont espacées de manière plutôt regulière par une distance de 35 ± 7 kpb ou un de ses multiples.

3. **Le phénomène de présence de fenêtres à fort χ^2 est uniforme sur les 4 chromosomes.**

χ^2 contre compression.

Quantitativement les résultats des deux méthodes diffèrent fortement: 2% des fenêtres sont repérées par le χ^2 et presque 10% par la compression. Clairement, les deux méthodes ne voient pas le même phénomène. Certaines fenêtres sont visibles avec la compression mais indétectables par le χ^2: par exemple celle débutant en 66000 dans le chromosome 2 contient deux zones RTA du motif CAA pour un gain de 84 bits, mais le χ^2 vaut 17.92 (voir figure 4.5). A l'opposé, certains segments révélés par le χ^2, ont un biais dû à une répétition en tandem d'un long motif invisible pour nos algorithmes; par exemple le segment du chromosome 11 entre les bases 647000–647500 (voir figure 4 exemple *b* dans [ODH95]). Pourtant cette conclusion doit être tempérée. En effet, d'une part de nombreuses fenêtres sont doublement identifiées par le χ^2 et la compression. D'autre part, deux tiers des fenêtres à fort χ^2 contiennent des répétitions alternées de deux motifs élémentaires. En outre, l'examen détaillé de fenêtres à très fort χ^2 révèle la présence de RTA de motifs dont la longueur varie de 3 à plus de 100 pb. Bien que l'approche du χ^2 ne soit pas directement liée au concept de répétition en tandem, elle en détectent de nombreuses car les RTA peuvent engendrer un biais compositionnel des dinucléotides.

Par ailleurs, le phénomène mesuré et identifié par le χ^2 est lui aussi uniforme quel que soit le chromosome testé et invisible sur des séquences pseudo-aléatoires (voir la figure 1 dans [ODH95]).

Différences méthodologiques.
Une certaine *souplesse* caractérise la méthode du χ^2, par un calcul général elle peut distinguer des fenêtres très variées (parce que le biais peut être dû à des RTA ou être dispersé sur plusieurs dinucléotides). L'avantage de la compression réside dans son *objectivité et dans sa capacité à expliquer le phénomène.* Si le test du χ^2 dépend fortement d'un seuil fixé arbitrairement, le seul arbitraire des méthodes de compression est situé dans le choix du codage des zones RTA. Il n'est pas totalement arbitraire pour les raisons suivantes:

- Le codage est contraint: il doit décrire une zone RTA; les informations qu'il inclut

```
Fichier: c2.66000  Algo: 3 Motif: CAA NbOccur: 44 Gain:  84

      0         1         2         3         4           Zone
      01234567890123456789012345678901234567890123456789
-------------------------------------------------------------
  0   AGTAATTCCTTCCAGTCTCACAATGCGCCCTCCCACCAGTCGAACTACCA

 50   CCCCCATTACAATCACATGAAATACAACAACACTGGTAGCTATTACTATT

100   ACAACAACAACAATAACAGCAGTGTAAACCCACATAACCAAGCTGGTCTA
       ^^^^^^^^^^^^                                      1
150   CAATCCATTAACAGATCTATTCCATCGGCCCCGTACGGGGCTTACAACCA

200   GAACAGAGCTAATGACGTACCATATATGAATACCCAAAAGAAACACCACA

250   GATTTAGCGCTAACAATAATTTGAACCAGCAAAAATACAAGCAATATCCC

300   CAGTATACGTCCAATCCAATGGTTACTGCACATCTGAAGCAAACGTACCC

350   TCAACTGTACTACAATAGCAACGTCAATGCTCACAACAACAACAACAACA
                                        ^^^^^^^^^^^^^^^^^  2
400   GCAACAACAACAACAACAACAACAACAACAGCAACAACAACAACAATCTT
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  2
450   TACAACCAGACGCAGTTCTCCACGAGGTACTTCAACTCGAACTCCTCTCC


Zone  1 : 101-113 :   4 repet,   0 erreurs, Gain :   6
=======

Zone  2 : 383-446 :  21 repet,   2 erreurs, Gain :  89
=======
     Position      Type d'Erreur      Lettre        Num Erreur
        6             SUBS_3           G             15
       16             SUBS_3           G             15
```

FIG. 4.5 - *Compression du segment [66000-66500] du chromosome 2 pour les motifs de longueur 3.*

doivent permettre une compression sans perte d'informations. Les items tels que la position d'une zone et sa liste d'erreurs sont obligatoires et naturels.

- L'arbitraire est placé au niveau de l'hypothèse faite sur le mécanisme de construction des RTA. Tout code compressé d'un objet se comprend comme une suite d'instructions pour la construction de cet objet (puisqu'on peut décompresser). Fixer le codage revient à limiter les instructions utiles pour le mécanisme que l'on présuppose. Plus la compression est grande, meilleure est l'hypothèse et meilleure est l'explication. On voit par là, combien la capacité à expliquer le phénomène décelé est intrinsèque à la compression.

Pratiquement l'explication de la compressibilité d'une fenêtre, est la traduction en language lisible du code de chaque zone RTA détectée. La figure 4.5 en est un exemple.

Conclusion.
Les deux approches χ^2 et compression sont complémentaires : si l'intersection de leurs résultats est loin d'être nulle, elle n'est pas non plus totale. Elles constatent globalement deux phénomènes ressemblants par leur uniformité. L'examen conjoint et la confrontation de leurs résultats pointent sur des voies à suivre pour affiner la détection du DOS-DNA : il paraît maintenant opportun d'étendre les algorithmes de compression aux RTA de motifs plus longs ou aux RTA à plusieurs motifs imbriqués.

4.3 Discussion des résultats biologiques.

Nous récapitulons ici les conclusions biologiques de l'analyse des résultats directement obtenus des expérimentations. Cette discussion est surtout le fruit de M-O. Delorme, A. Hénaut et E. Ollivier du CGM, avec qui nous entretenons une collaboration. La section précédente montre les résultats des expériences d'identification de zones RTA dans les chromosomes de levure. Ce travail a pour objectif d'améliorer la compréhension biologique de la présence de RTA dans les séquences et de la structure de l'objet chromosome. Notre discussion se polarise sur deux questions majeures qui surgissent à la lecture de ces résultats :

1. Comment fonctionnent les mécanismes moléculaires capables d'engendrer ou d'amplifier une zone RTA ? Plus précisément : quelle taille de motifs sont concernées, certains motifs sont-ils préférentiellement choisis, les données confortent-elles les hypothèses de mécanismes avancés jusque là ? Ces questions correspondent aux interrogations 1, 2 et 3 de la section 2.3.

2. La présence de RTA est-elle uniforme dans les chromosomes de levure ou varie-t-elle avec le contenu du chromosome ? En termes plus profonds, est-elle liée à la sémantique génique du chromosome ou à sa structure propre ?

Nous développons les éléments de réponse qui proviennent de l'analyse.

4.3.1 Mécanismes de génération des zones RTA.

Les cas des maladies génétiques humaines qui ont initié l'intérêt pour le DOS-DNA et les zones RTA en particulier, sont uniquement liées à l'amplification de triplets dans ou à

proximité des gènes. Nos résultats décrivent un phénomène bien plus large, dont les principales caractéristiques sont :

1. de ne pas être limité à l'homme, mais d'apparaître aussi chez un des organismes eucaryotes les plus simples et les plus éloignés de l'homme en termes de parenté dans l'évolution des espèces ;

2. de s'appliquer, non seulement aux triplets, mais aussi à de nombreuses longueurs de motifs : les mono-, di- et trinucléotides dans nos expériences et des motifs plus longs vus avec les zones détectées par le test de χ^2 ;

3. de plutôt sélectionner les motifs amplifiés sans pour autant retrouver uniquement ceux impliqués dans les maladies génétiques humaines.

Nous développons plus finement cette dernière propriété et une conséquence sur une contrainte du mécanisme de génération par blocage de la réplication. Premièrement la sursélection des motifs A et T pour les mononucléotides est retrouvée dans les expériences sur les chaînes de Markov. Ceci laisse penser que la disproportion d'utilisation des bases en faveur de A, T est à l'origine de ce biais.

[HHZ+94] étudie les RTA de trinucléotides chez l'homme. Les motifs sont groupés par famille de 6, comprenant les 3 décalés d'un trinucléotide et leurs complémentaires. Dans nos expériences, la famille de TGG n'apparaît dans aucune fenêtre, et celle de CAG est présente dans 6 fenêtres au total, alors qu'elles sont statistiquement sursélectionnées dans [HHZ+94]. Il s'agit d'une différence notable de la présence de RTA chez l'Homo sapiens et chez Saccharomyces cerevisiae.

Invalidation de la contrainte d'auto-appariement du motif dans le mécanisme de génération des RTA par blocage de la réplication.

Ce mécanisme est présenté dans la sous-section 2.2.2, on voit sur le schéma 2.5 que le brin néosynthétisé doit adopter une structure en épingle à cheveux. Or ce brin contient la répétition en tandem d'un seul motif de longueur 3, dans l'exemple il s'agit de CCG. Cela contraint à n'amplifier que des motifs capables de s'auto-apparier avec eux-mêmes au moins deux bases sur trois (plus que deux est impossible pour les triplets). La mise en correspondance de CCG en face de lui-même ou en face de CGC, dans la RTA, ne modifie en rien la contrainte. Nous présentons ci-dessous le tableau des familles de motifs identifiées dans les zones RTA des chromosomes. La notion de famille est expliquée au paragraphe précédent. Les familles des motifs dégénérés AAA et CCC sont exclues puisque non recherchées par nos algorithmes. Une ligne donne : un motif de la famille, le nombre de zones RTA dans lesquelles la famille a été identifiée, le pourcentage par rapport au nombre total de zones, le pourcentage d'apparition à toutes les positions du chromosome. Enfin, la dernière colonne indique le classement de la famille par un calcul de χ^2 qui teste l'hypothèse que les motifs des zones ne sont pas choisis, mais respecte la probabilité de leur apparition dans la séquence du chromosome.

On voit dans ce tableau que les familles de AAG et de ACC sont très fortement surreprésentées (classement 1+ et 4+) et totalisent à elles-seules plus de 50% des zones RTA. Or les motifs de ces familles ne peuvent pas s'auto-apparier deux bases sur trois. De même les familles ACC et AGG sont sous-représentées. Il semble donc que le mécanisme de génèse

Famille	Nb Zones	% Zones	% chrom.	Classement χ^2
AAT	16	15	17	9-
AAG	24	23	15	4+
AAC	38	37	13	1+
ATG	15	15	13	8+
AGT	0	0	10	2-
AGC	6	6	8	7-
ACG	1	1	6	6-
ACC	0	0	8	3-
AGG	2	2	8	5-
CGC	0	0	2	10-

TAB. 4.7 - *Représentation des familles de triplets dans les zones RTA et dans les chromosomes.*

des RTA soit indifférent à la contrainte d'auto-appariement. L'hypothèse de l'emploi du mécanisme par blocage de la réplication et celle de son existence sont contredites, au moins chez la levure.

4.3.2 Caractérisation de la structure d'un chromosome.

Aujourd'hui, on ne sait pas caractériser un chromosome, ni définir un modèle de la structure d'un chromosome. Les chromosomes se distinguent entre eux principalement par leur contenu en gènes. Une propriété uniforme sur plusieurs chromosomes et qui ne dépend pas du contenu génique, représente une caractéristique de l'objet chromosome. Elle fait partie d'une structure imposée à chaque chromosome.

La disponibilité récente des séquences a prohibé les expériences qui visent à caractériser les chromosomes. Par exemple, le pourcentage de bases appartenant à un gène était évalué globalement sans regarder précisément la séquence des chromosomes, il était ainsi évalué à 70% environ. Aujourd'hui, la détection de zones RTA et le calcul du pourcentage de séquence codant pour un gène, exhibent apparement de telles propriétés uniformes sur les 4 chromosomes envisagés. Une conclusion plus définitive passe par un examen fin des résultats pour savoir si la présence de zones RTA est réellement indépendante de la présence de gènes.

Troisième partie

Représentation optimale et complexité de Kolmogorov.

Chapitre 1

Introduction

La théorie de la complexité de Kolmogorov considère que parmi toutes les représentations d'un objet (représentations que l'on appelle programmes), la longueur de la plus courte d'entre elles, mesure la complexité informationnelle de cet objet.

Nous considérons qu'un objet peut toujours être représenté sous la forme d'un texte[1]. Sur les différentes représentations d'un objet, la théorie de la complexité de Kolmogorov, nous apprend deux choses. Premièrement, pour un texte quelconque y de $\{0,1\}^*$, la complexité de Kolmogorov de y, notée $K(y)$, nous donne la taille minimale des représentations de ce texte. Mais cette fonction K qui a y associe $K(y)$, n'est pas calculable, donc elle ne nous permet pas effectivement de savoir de combien de bits une représentation que l'on utilise dépasse cette taille minimale $K(y)$. Deuxièmement, une faible proportion de textes sont compressibles, i.e. ont une représentation plus courte qu'eux mêmes.

Cependant dans la pratique, les objets que nous stockons sous une forme codée dans les mémoires d'ordinateurs sont souvent compressibles. On peut même les regrouper en classes : textes en langage naturel, programmes, images, fichiers comptables, etc, pour lesquelles des méthodes adaptées obtiennent de bons taux de compression en tirant profit des régularités ou de la structure commune aux objets de la classe (cf. [BCW90, Sto88]).

Un concepteur de codage travaille entre ces deux cas extrêmes. En construisant un codage pour une classe d'objets, il est dans l'incapacité de prouver que son codage est optimal, sinon en le justifiant par des arguments de bon sens. Il ne peut répondre à la question : la représentation utilisée est-elle optimale? D'autre part, il essaye de coder des objets qui lui semblent compressibles, parce qu'il s'agit d'une classe particulière d'objets ayant une structure commune. Comment savoir si dans les codes employés, ne se cachent pas des redondances exploitables par un algorithme de compression. Apporter les moyens formels pour répondre à ces questions est un des objectifs de ce travail.

Examinons le concept de complexité de Kolmogorov. Elle est une limite de la taille des représentations possibles d'un objet. Cependant, la notion de représentation optimale n'a pas de sens général, car la complexité de Kolmogorov dépend de la machine de Turing universelle fixée, avec laquelle elle est calculée. Pour tout objet, on peut définir une machine de Turing universelle qui associe à cet objet le programme minimal de longueur nulle, alors qu'une autre machine de Turing universelle lui associe un programme minimal de longueur 100 par exemple.

1. Par conséquent, nous employons indifféremment ces deux termes. Chaîne et chaîne de caractères sont aussi des synonymes de texte.

Lorsque l'on change de machine de Turing universelle de référence, la mesure de la complexité de Kolmogorov ne varie au plus que d'une constante: la complexité de Kolmogorov est robuste mais pas unique. Pour un objet particulier et une machine de Turing universelle U fixée, elle ne peut servir de limite générale de représentation, mais elle indique une borne de compression relativement à U. **Elle ne permet donc pas de répondre à la question de l'optimalité d'une représentation pour un texte fixé pris isolément.** En outre, la non-calculabilité de la complexité de Kolmogorov empêche d'utiliser cette borne. Il est facile de majorer la complexité de Kolmogorov d'une suite, mais très difficile, voire impossible de la calculer exactement, sauf pour un nombre fini de suites de caractères. Cependant, il existe un lien naturel entre la complexité de Kolmogorov et la compression, lien théorique mais non utilisable en pratique.

Puisqu'on ne peut généralement pas savoir quel est la plus petite représentation pour un texte donné, il est intéressant de définir une notion d'optimalité pour une famille de textes, qui soit moins contraignante et utilisable effectivement : c'est l'objet de ce travail. Il est basé sur l'idée suivante. La structure d'un texte nous dicte souvent une représentation intuitivement optimale. Prenons en exemple un texte constitué par un nombre entier de concaténations d'un facteur.

Exemple sur $\{a, b\}$: $(abaab)^3 = abaababaababaab$
Une représentation intuitivement appropriée pour ce mot d'une structure spéciale, est donnée par le facteur suivi de son nombre de répétitions :

$abaab; 3$
En effet, il est naturel de penser que pour les mots de la forme x^m, il n'existe pas en général, de forme plus courte que d'indiquer le facteur constitutif et le nombre de fois où il est répété. Cette représentation est valable pour tout mot adoptant cette forme avec un facteur quelconque et un nombre de répétitions quelconque. Dans une famille de textes présentant cette structure, nous savons que la plupart des facteurs constitutifs sont sans particularités (quelconques), puisque ce sont des textes de $\{0, 1\}^*$ et que la théorie de la complexité de Kolmogorov affirme qu'ils sont en général incompressibles. Il en va de même pour la donnée : nombre de répétitions du facteur qui est codable par un texte sur $\{0, 1\}$. Donc la plupart de ces textes sont optimalement représentés sous la forme (x, m) i.e. facteur et nombre de répétitions. Ceci permet d'affirmer que cette forme de représentation est optimale en moyenne pour une famille de textes de cette structure. Nous proposons une formalisation de cette idée intuitive.

Ce formalisme permet de montrer l'optimalité en moyenne d'une représentation, comme dans le cas précédent, mais aussi la non-optimalité en moyenne. Par exemple, pour la famille des mots de la forme ;

$(y.\overline{y})^m$
où $\overline{y}$ est le symétrique de y (sur l'alphabet $\{a, b\}$, a est le symétrique de b et inversement); la représentation préconisée pour le premier exemple n'est plus optimale. L'optimalité est perdue puisque par hypothèse, tous les textes de cette famille ont un facteur constitutif formé d'un sous-mot et de son symétrique, un facteur qui est donc régulier et pas incompressible comme la moyenne des textes. On pourra utiliser la représentation :

$$y; m \quad \text{au lieu de} \quad y\overline{y}; m$$

Comparaison avec les autres approches.

Plusieurs essais pour cerner le concept de représentation optimale ont déjà été tentés. Les notions proposées sont toujours des notions d'optimalité relative à une hypothèse[2].

a) Théorie probabiliste de l'information [CT91, Huf52, Sha48].

On suppose que les lettres des messages respectent une **loi de probabilité fixée** (par exemple, probabilité de «0» : $\frac{1}{3}$ et probabilité de «1» $\frac{2}{3}$). Il est prouvé qu'on ne peut compresser mieux que d'un taux lié à l'entropie de la mesure de probabilité. Cette approche est évidemment très restrictive car aucune régularité autre que statistique n'est exploitée. En compression de textes, les algorithmes les plus utilisés aujourd'hui ne se satisfont pas de l'exploitation des régularités statistiques (voir [BCW90, CR94, Sto88]).

b) Théorie algorithmique de l'information [Cal94, CT91, Del94, LV93, Wat92].

Mis à part le choix de la **machine de Turing universelle**, l'approche est parfaitement générale. Comme précisé plus haut, elle est irréaliste à cause de la non-calculabilité de K.

c) Approche de Goldberg et Sipser [GS91].

On suppose que les mots à compresser appartiennent à un **langage particulier**. Une notion d'optimalité (voir la section 2.6) s'impose alors, mais elle est de peu d'intérêt car les algorithmes de compression *par rangement et numérotation* sont optimaux mais très contraignants et de grande complexité ([GS91] montre que l'algorithme est de complexité $\#P$ en temps).

d) Approche de représentation optimale en moyenne.

Notre approche ne se réduit à aucune des trois précédentes et propose de compresser relativement à une **hypothèse structurale sur les mots**. L'idée avancée est que l'origine d'un texte permet souvent de savoir qu'il a une certaine structure algébrique bien précise. Un texte de chanson par exemple, est souvent constitué d'une suite de couplet-refrain répété n fois. n est à priori indéterminé, éventuellement égal à 1 et donc tout texte peut être un texte de chanson : une hypothèse structurale n'est donc pas équivalente à une hypothèse d'appartenance à un langage. Notre théorie ne se ramène donc pas à celle de Goldberg et Sipser, elle généralise d'une certaine façon leur approche. La notion de forme que nous introduisons permet d'exprimer de telles hypothèses structurales et conduit naturellement à une définition de représentation optimale que nous détaillons ci-après. Cette approche qui se fonde sur la complexité de Kolmogorov n'en a pas les inconvénients concernant la non-calculabilité : peut-être peut-on la voir comme la version praticable de la théorie algorithmique de l'information?

2. L'objet sur lequel porte l'hypothèse est mis en gras pour chaque approche.

Chapitre 2

Définitions et résultats de bases.

Nous présentons tout d'abord, quelques notations, puis la définition de forme. Ensuite, le concept central d'optimalité est minutieusement introduit avant d'aboutir à la définition. Nous exhibons plusieurs versions équivalentes de notre définition d'optimalité originale, qui lui confèrent une certaine robustesse. Enfin un théorème central, prouve que la contrainte d'utilisation exclusive de programmes auto-délimités n'affecte pas notre théorie de la représentation optimale. Elle est ainsi rendue compatible avec deux versions importantes de la théorie de la complexité algorithmique de l'information : l'une exigeant des programmes auto-délimités et l'autre non. Puis nous introduisons deux lemmes qui donnent des conditions suffisantes pour l'optimalité et la non-optimalité et permettent de simplifier des preuves ultérieures. Enfin, la dernière section établit que l'injectivité d'une forme entraîne son optimalité. Nous comparons alors les définitions de fonction de compression optimale de [GS91] avec notre définition de forme optimale.

2.1 Notations préliminaires et notion de forme.

Cette section introduit le concept de forme, qui formalise une représentation possible pour des objets. À cette notion s'applique la propriété d'optimalité définie par la suite.

Notation 6 L'alphabet de référence est $\{0,1\}$ et nous notons $\{0,1\}^*$ l'ensemble de toutes les chaînes finies sur $\{0,1\}$. Soit E un sous-ensemble de $\{0,1\}^*$, $E^{<n}$ (respectivement $E^{\leq n}$, E^n) correspond à l'ensemble de tous les textes de E de taille inférieure à n (resp. inférieure ou égale à n, resp. égale à n).

Notation 7 Soient :

- E un ensemble de cardinal fini
- f une fonction de E dans $\mathbf{R}$ (l'ensemble des nombres réels)

on note :

$$\underset{t\in E}{Moy}\, f(t) = \frac{\sum_{t\in E} f(t)}{Card(E)}$$

Notation 8 Nous utilisons les symboles d'ordre de grandeur o et O avec les significations suivantes. Soient f et g des fonctions réelles. La notation $f(x) = o(x)$ signifie que :

$$\lim_{x\to\infty} \frac{f(x)}{x} = 0$$

et $f(x) = O(g(x))$ veut dire qu'il existe deux constantes positives c, x_0 telles que :

$$\forall x \geq x_0 : |f(x)| \leq c|g(x)|$$

Définition 18 Soit $Types = \{\mathbf{N}, \{0,1\}^*\}$. Une *forme* f est une application de T sur $\{0,1\}^*$ telle que :

$$\begin{array}{rcl} f : T & \rightarrow & \{0,1\}^* \\ t & \mapsto & f(t) \end{array}$$

où T est un produit d'éléments de $Types$. Si on pose $T = A_1 \times \cdots \times A_k$ où pour tout i, A_i est un élément de $Types$ on a :

$$\begin{array}{rcl} f : A_1 \times \cdots \times A_k & \rightarrow & \{0,1\}^* \\ t_1, \cdots, t_k & \mapsto & f(t_1, \cdots, t_k) \end{array}$$

Une **forme** définit une représentation possible d'un texte par un k-uplet d'objets respectivement de type $A_1, \ldots A_k$. Elle indique la fonction qui à partir de ce k-uplet d'objets retrouve le texte original, i.e. la suite des caractères qui le composent, que nous appelons sa **représentation naturelle.**

Exemple : la forme suivante

$$\begin{array}{rcl} f : \{0,1\}^* \times \mathbf{N} & \rightarrow & \{0,1\}^* \\ (x, m) & \mapsto & x^m \end{array}$$

donne un codage possible pour les mots de $\{0,1\}^*$ qui sont fait de la répétition d'un facteur un nombre entier de fois. Cette représentation contient deux objets : le facteur (un texte sur $\{0,1\}^*$) et le nombre fois où il est répété (un entier).

En d'autres termes, une forme donne un codage possible pour un ensemble de textes et le décodeur approprié. T donne le type de chaque objet de la représentation : t_1 de type $A_1, \cdots, t_i$ de type $A_i, \cdots, t_k$ de type A_k. Contrairement à [GS91], il peut y avoir plusieurs codages possibles pour un même texte.

Remarque : pour l'instant, nous n'exigeons pas que le décodeur soit une fonction calculable. Lorsque f est calculable, on dit que la forme est *calculable*. Il existe une procédure uniforme pour concevoir un algorithme de compression «brute-force», mais en pratique pour les cas étudiés ci-après, nous pouvons toujours trouver un algorithme de compression polynômial.

Notation 9 Si y est un objet et U une machine de Turing universelle de référence, nous notons $K_U(y)$ la complexité de Kolmogorov de y, i.e. la taille du plus petit programme capable d'engendrer y sur U. Étant donnée la robustesse de la complexité de Kolmogorov, nous supposons une machine de Turing universelle fixée et nous notons simplement $K(y)$ la complexité de Kolmogorov de y. Nous utilisons aussi la complexité de Kolmogorov à programmes auto-délimités de y et la notons $H(y)$. La complexité de Kolmogorov dite auto-délimitée prend en compte dans la complexité de y l'auto-délimitation des programmes capables d'engendrer y. On sait que les mesures de complexité par H et K ne diffèrent que d'un terme additif en $O(\log(|y|))$. Pour plus de détail, nous renvoyons le lecteur à [LV93].

Un programme auto-délimité signifie que la lecture de ce programme indique où il se termine. La différence provient de la manière dont sont écrits les programmes sur le ruban de la machine de Turing universelle : avec l'alphabet $\{0, 1, blanc\}$ où le blanc sert de délimiteur sur une machine à programmes non-auto-délimités, ou bien avec l'alphabet $\{0,1\}$ sur une

machine à programmes auto-délimités. Sur les premières, le ruban débute par le programme qui est suivi par des blancs, on sait donc où il se termine. Tandis que sur les deuxièmes, le programme code sa propre longueur de façon à ce qu'on trouve sa fin sur un ruban qui ne contient que des caractères 0 et 1.

Par ailleurs, nous utilisons une forme de la complexité à plusieurs variables que nous notons : $K(x_1, \cdots, x_m)$. Elle correspond à la longueur du plus petit programme capable d'imprimer x_1, puis x_2, ..., jusqu'à x_m et un code de séparation permettant de les distinguer les uns des autres. Remarquons que ce plus petit programme est forcément plus long que le plus petit programme qui engendre simplement la concaténation de tous les x_i, i.e. $K(x_1.\cdots.x_m)$.

Fait 1 En outre, nous savons d'après [LV90, LV93] que :

$$K(x_1, \cdots, x_m) \leq K(x_1) + \cdots + K(x_m) + O\left(\log\left(\sum_{i=1}^{m} K(x_i)\right)\right)$$

Nous utilisons la formule suivante plusieurs fois dans notre travail.

Fait 2

$$\forall n \in \mathbf{N} : \sum_{i=0}^{n} i * 2^i = (n-1) * 2^{n+1} + 2$$

Preuve Soit $n \in \mathbf{N}$.

$$\begin{aligned}
\sum_{i=0}^{n} i * 2^i &= 2\sum_{i=0}^{n} i * 2^{i-1} \\
&= 2\left[\left(\sum_{i=0}^{n} x^i\right)'\right]_{x=2} \\
&= 2\left[\left(\frac{1-x^{n+1}}{1-x}\right)'\right]_{x=2} \\
&= 2\left[\left(\frac{-(n+1) * x^n * (1-x) + (1-x^{n+1})}{(1-x)^2}\right)\right]_{x=2} \\
&= 2\left[\left(\frac{-(n+1) * x^n + n * x^{n+1} + 1}{(1-x)^2}\right)\right]_{x=2} \\
&= n * 2^{n+2} - (n+1) * 2^{n+1} + 2 \\
&= (n-1) * 2^{n+1} + 2
\end{aligned}$$

□

2.2 Définition de l'optimalité d'une forme.

La définition centrale de ce travail concerne l'optimalité d'une forme, optimalité du point de vue de la taille de la représentation en nombre de bits. Notons y un texte sur $\{0,1\}^*$.

2.2.1 Justification et définition.

Intuitivement, une représentation est une liste d'informations $(t_1, \ldots, t_k)$ qui permet de retrouver y. Elle sous-entend la notion d'algorithme de reconstruction de y à partir de cette liste d'informations. Remarquons qu'il existe plusieurs représentations possibles pour un texte y, et qu'elles peuvent être plus ou moins longues que sa représentation naturelle. Pour le mot *abaababaab* et la forme précédemment vue en exemple, $(abaababaab, 1)$ indique que le mot *abaababaab* est constitué d'une fois le facteur *abaababaab*, tandis que $(abaab, 2)$ indique qu'il se décompose en deux fois le mot *abaab*. Les deux représentations sont possibles avec cette forme, mais $(abaababaab, 1)$ est plus longue que la représentation naturelle, tandis que $(abaab, 2)$ est plus courte.

Parmi toutes les listes d'informations qui représentent y, certaines contiennent juste les informations nécessaires à la construction de y, tandis que d'autres incluent des informations superflues puisqu'elles sont plus longues que les premières. Par définition de la complexité de Kolmogorov, les meilleures représentations de y, du point de vue de la quantité d'information, sont celles de longueur $K(y)$.

La fonction qui à y associe $K(y)$ n'est pas récursive, donc d'un point de vue pratique, on ne peut que très rarement savoir si un texte donné y est compressé au maximum. Prise directement, la complexité de Kolmogorov ne permet pas de parler de représentation optimale et ne rend donc pas compte de notre intuition qu'il existe des représentations optimales. Une autre raison rend impossible l'utilisation directe de la complexité de Kolmogorov pour parler de représentation optimale d'un texte y. Elle est qu'une machine universelle de référence bien choisie, peut associer à un texte donné y long, un plus petit programme très court (et donc permettre une représentation très courte de ce texte particulier). Ceci nous conduit à n'envisager le concept de représentation optimale que pour une famille de textes. Notre idée est qu'en associant une hypothèse structurelle aux textes que nous cherchons à représenter et en ne visant qu'une optimalité en moyenne, il devient possible d'utiliser de manière effective la notion d'optimalité d'une représentation. On peut alors exprimer l'intuition que par exemple, les textes de la forme x^m sont représentés au mieux en moyenne par les données x, m.

Si f est une forme, la représentation de $f(t)$ par t est optimale, si elle contient la même quantité d'information que $f(t)$, i.e. si :

$$|t| \quad = \quad K(f(t))$$

En exigeant que cette optimalité soit vraie en moyenne pour $t \in E$ (où E est un ensemble quelconque), nous obtenons que cette représentation est optimale en moyenne pour E ssi :

$$\underset{t \in E}{Moy}\, |t| \quad = \quad \underset{t \in E}{Moy}\, K(f(t))$$

Lorsque E est fini, la non-calculabilité de K interdit de prouver une telle égalité. Nous ne nous intéressons donc qu'aux cas où E est infini. Lorsque E est infini, la version asymptotique la plus naturelle de cette égalité est (d'autres versions sont envisagées ci-après) :

$$\frac{Moy_{|t|=n}\, K(f(t))}{Moy_{|t|=n}\, |t|} \quad \rightarrow \quad 1$$

puisque la moyenne de taille de t sur l'ensemble $\{t : |t| = n\}$ est égale à n, ceci est équivalent à :

$$\underset{|t|=n}{Moy}\, K(f(t)) \quad = \quad n + o(n)$$

D'où notre définition :

Définition 19 Une forme $f : T \to \{0,1\}^*$ est *K-optimale* ssi :

$$\underset{|t|=n}{Moy}\, K(f(t)) = n + o(n) \tag{2.1}$$

i.e. si :

$$\lim_{n\to\infty} \frac{\left| Moy_{|t|=n}\, K(f(t)) - n \right|}{n} = 0$$

De même, f est *H-optimale* ssi :

$$\underset{|t|=n}{Moy}\, H(f(t)) = n + o(n) \tag{2.2}$$

2.2.2 Une deuxième version naturelle de l'optimalité.

La définition d'optimalité conserve le même sens si la moyenne est faite sur l'ensemble $\{t : |t| \leq n\}$ i.e sur un «anneau» de représentations plutôt que sur un «disque» de représentations. Les deux versions de la définition semblent aussi naturelles l'une que l'autre. Nous montrons ici qu'elles coïncident.

Propriété 19 f est K-optimale est équivalent à :

$$\underset{|t|\leq n}{Moy}\, K(f(t)) = n + o(n) \tag{2.3}$$

Preuve Nous devons démontrer que la définition implique l'équation 2.3, puis que cette équation implique l'équation 2.1.

Démonstration de $\Rightarrow$.
Nous supposons que :

$$\underset{|t|=n}{Moy}\, K(f(t)) \;=\; n + o(n)$$

et nous montrons que :

$$\underset{|t|\leq n}{Moy}\, K(f(t)) \;=\; n + o(n)$$

Soit $a > 0$. Posons :

$$g(n) \;=\; \frac{|n - Moy_{|t|=m}\, K(f(t))|}{n}$$

On sait que $\lim_{n\to\infty} g(n) = 0$. Par hypothèse il existe $n_0 > 0$ tel que :

$$\forall n > n_0 \frac{|n - Moy_{|t|=n}\, K(f(t))|}{n} \;<\; \frac{a}{6}$$

Quel que soit $n > n_0$ nous avons :

$$\begin{aligned}\frac{Moy_{|t|\leq n} K(f(t))}{n} &= \frac{\sum_{m=0}^{n} \left[2^m \, Moy_{|t|=m} K(f(t))\right]}{n(2^{n+1}-1)} \\ &= \frac{\sum_{m=0}^{n} [2^m (m+g(m))]}{n(2^{n+1}-1)} \\ &= \frac{\sum_{m=0}^{n_0-1} [2^m (m+g(m))]}{n(2^{n+1}-1)} + \frac{\sum_{m=n_0}^{n} [2^m (m+g(m))]}{n(2^{n+1}-1)}\end{aligned}$$

Soit $n_1 > n_0$ tel que :

$$\forall n > n_1 \quad \frac{\sum_{m=0}^{n_0-1} [2^m (m+g(m))]}{n(2^{n+1}-1)} < \frac{a}{3}$$

On a :

$$\begin{aligned}\sum_{m=n_0}^{n} \frac{2^m |g(m)|}{n(2^{n+1}-1)} &= \frac{|g(n)|2^n}{n(2^{n+1}-1)} + \frac{|g(n-1)|2^{n-1}}{n(2^{n+1}-1)} + \cdots + \frac{|g(n_0)|2^{n_0}}{n(2^{n+1}-1)} \\ &< \frac{a}{6} + \frac{1}{2}\frac{a}{6} + \cdots + \frac{1}{2^{n-n_0}}\frac{a}{6} \\ &< \frac{a}{3}\end{aligned}$$

et :

$$\begin{aligned}\frac{\sum_{m=n_0}^{n} m2^m}{n(2^{n+1}-1)} &= \frac{(n-1)2^{n+1} - (n_0-1)2^{n_0+1}}{n(2^{n+1}-1)} \\ &= \frac{1}{1-\frac{1}{2^{n+1}}} + \frac{-2^{n+1} - (n_0-1)2^{n_0+1}}{n(2^{n+1}-1)} \\ &= 1 - b(n)\end{aligned}$$

où $\lim_{n\to\infty} b(n) = 0$. Donc il existe $n_2 > 0$ tel que :

$$\forall n > n_2 : \; |b(n)| < \frac{a}{3}$$

Soit $n > max(n_0, n_1, n_2)$, nous avons alors :

$$\begin{aligned}\left|\frac{Moy_{|t|\leq n} K(f(t)) - n}{n}\right| &= \left|\frac{Moy_{|t|\leq n} K(f(t))}{n} - 1\right| \\ &< \frac{a}{3} + \frac{a}{3} + \frac{a}{3} \\ &= a\end{aligned}$$

De ceci découle l'équation 2.3.

Démonstration de $\Leftarrow$.
Nous supposons que :

$$\underset{|t|\leq n}{Moy} K(f(t)) = n + o(n)$$

et nous montrons que :

$$\underset{|t|=n}{Moy}\, K(f(t)) \;=\; n + o(n)$$

Quel que soit $n > n_0$ nous avons :

$$\begin{aligned}
\underset{|t|\leq n}{Moy}\, K(f(t)) &= \frac{\sum_{m=0}^{n}\left[2^m\, Moy_{|t|=m}\, K(f(t))\right]}{2^{n+1}-1} \\
&= \frac{2^n\, Moy_{|t|=n}\, K(f(t))}{2^{n+1}-1} + \frac{\sum_{m=0}^{n-1}\left[2^m\, Moy_{|t|=m}\, K(f(t))\right]}{2^{n+1}-1} \\
&= \frac{2^n\, Moy_{|t|=n}\, K(f(t))}{2^{n+1}-1} + \frac{(2^n-1)\, Moy_{|t|\leq n-1}\, K(f(t))}{2^{n+1}-1}
\end{aligned}$$

et donc que :

$$\underset{|t|\leq n}{Moy}\, K(f(t)) - \frac{(2^n-1)\, Moy_{|t|\leq n-1}\, K(f(t))}{2^{n+1}-1} \;=\; \frac{2^n\, Moy_{|t|=n}\, K(f(t))}{2^{n+1}-1}$$

finalement :

$$\begin{aligned}
\underset{|t|=n}{Moy}\, K(f(t)) &= \frac{2^{n+1}-1}{2^n}\left[\underset{|t|\leq n}{Moy}\, K(f(t)) - \frac{(2^n-1)}{2^{n+1}-1}\underset{|t|\leq n-1}{Moy}\, K(f(t))\right] \\
&= (2+a(n))\left[n + g(n) - (\frac{1}{2} + b(n))(n-1+g(n-1))\right] \\
&= n + o(n)
\end{aligned}$$

où $\lim_{n\to\infty} a(n) = 0$ et $\lim_{n\to\infty} b(n) = 0$. □

2.3 Incompressibilité des textes sans hypothèse de forme.

Le résultat suivant exprime en termes d'optimalité, un résultat de base de la théorie de la complexité de Kolmogorov. Lorsqu'on ne connaît rien d'un texte, qu'aucune hypothèse n'est fournie sur sa structure, sa plus courte représentation en moyenne est le texte lui-même. En d'autres termes, la plupart des textes sont incompressibles. Cette section est dédiée à plusieurs versions de ce résultat, qui permettent une troisième définition de l'optimalité. Nous montrons cette optimalité pour K (et de la même manière pour H) sur un anneau et un disque de représentations, avec la complexité K à une variable et à plusieurs variables.

2.3.1 Incompressibilité pour la complexité K à une variable.

Propriété 20

$$\underset{|t|\leq n}{Moy}\, K(t) = n + O(1)$$

Preuve Nous savons qu'il existe $c > 0$ tel que :

$$\forall t \in \{0,1\}^* :\; K(t) \;\leq\; |t| + c$$

où c est une constante qui ne dépend que de la machine de Turing universelle choisie comme référence, est un résultat fondamental de la théorie de la complexité de Kolmogorov. On le démontre en considérant M la machine de Turing qui imprime son argument. Une machine de Turing universelle peut simuler M en ayant comme programme le numéro de M auto-délimité suivi de son argument t : $0^{n(M)}1t$. Ce programme est de taille : $n(M) + |t|$ et ne dépasse t que d'une constante, puisque la numérotation des machines de Turing est fixée. En moyennant la complexité de Kolmogorov des chaînes de l'ensemble $E = \{t : |t| \leq n\}$, nous sommes sûrs que :

$$\underset{|t|\leq n}{Moy}\, K(t) \leq n + c \tag{2.4}$$

Soit n un entier. Fixons nous une machine de Turing universelle U et notons E l'ensemble des chaînes de longueur inférieure ou égale à n. La machine de Turing de référence étant fixée, pour tout texte t de E, $K(t)$ est fixé. Nous devons minorer :

$$\frac{\sum_{|t|\leq n} K(t)}{Card(E)}$$

Au mieux, parmi tous ces programmes minimaux, un est de longueur zéro, deux sont de longueur un, et ainsi de suite jusqu'à n, où les 2^n programmes minimaux sont au mieux tous de longueur n, i.e. au mieux, ce sont toutes les chaînes sur $\{0, 1\}$ de longueur n. Ce qui se traduit par l'inégalité suivante :

$$\sum_{|t|\leq n} K(t) \quad \geq \quad \sum_{i=0}^{n} i * 2^i$$

d'après le fait 2 on obtient :

$$\begin{aligned} \frac{\sum_{|t|\leq n} K(t)}{Card(E)} &\geq \frac{(n-1) * 2^{n+1} + 2}{2^{n+1}} \\ &= n - 1 + 2^{-n} \end{aligned}$$

Ainsi, grâce à cette dernière inégalité et à l'inégalité 2.4, nous avons :

$$\underset{|t|\leq n}{Moy}\, K(t) = n + O(1)$$

□

Propriété 21

$$\underset{|t|=n}{Moy}\, K(t) = n + O(1)$$

Preuve Même preuve que la propriété précédente.

2.3.2 Incompressibilité pour la complexité K à plusieurs variables.

Nous généralisons cette propriété à la forme de K à plusieurs variables.

Propriété 22

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1, \cdots, x_m) = n + O(\log(n))$$

Preuve Nous allons démontrer qu'il existe $c > 0$ tel que :

$$\left| \underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1,\cdots,x_m) - n \right| \leq c\log(n)$$

en prouvant les deux inégalités suivantes :

- il existe $c_1 > 0$ tel que :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1,\cdots,x_m) \leq n + c_1 log(n) \tag{2.5}$$

- il existe $c_2 > 0$ tel que :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1,\cdots,x_m) + c_2 log(n) \geq n \tag{2.6}$$

Preuve de l'inégalité 2.5.
Nous savons, suite à la remarque faite après la définition de $K(x_1,\cdots,x_m)$, que :

$$K(x_1,\cdots,x_m) \leq K(x_1) + \cdots + K(x_m) + O(\log(\sum_{i_j} K(x_{i_j}))$$

et donc qu'il existe $c > 0$ tel que :

$$K(x_1,\cdots,x_m) \leq \sum_i K(x_i) + c\log\left(\sum_i K(x_i)\right)$$

Si $|x_1| + \cdots + |x_m| \leq n$, d'après la propriété 20 on a :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} (\sum_i K(x_i)) = n + O(1)$$

donc en calculant la moyenne, on peut trouver un $c_1 > 0$ tel que :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1,\cdots,x_m) \leq n + c_1\log(n)$$

Preuve de l'inégalité 2.6.
Comme précisé dans la définition : $K(x_1,\cdots,x_m)$ est supérieur à $\sum_i K(x_i)$ et grâce à la propriété 20, nous savons que :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} (\sum_i K(x_i)) = n + O(1)$$

Nous en déduisons qu'il existe $c_2 > 0$ tel que :

$$\underset{|x_1|+\cdots+|x_m|\leq n}{Moy} K(x_1,\cdots,x_m) + c_2\log(n) \geq n$$

□

Propriété 23

$$\underset{|x_1|+\cdots+|x_m|=n}{Moy} K(x_1,\cdots,x_m) = n + O(\log(n))$$

Preuve Même preuve que pour la propriété précédente.

2.3.3 Une formulation équivalente de l'optimalité.

Nous savons donc que la moyenne des tailles des représentations sur l'ensemble des représentations $\{t : |t| = n\}$ est égale la moyenne de leur complexité de Kolmogorov sur ce même ensemble, nous formulons deux versions équivalentes de K-optimalité. Les mêmes équivalences sont valables avec la version auto-délimitée de la complexité, H.

Théorème 2 f est K-optimale ssi :

$$\underset{|t|=n}{Moy}\, K(f(t)) = \underset{|t|=n}{Moy}\, K(t) + o(n)$$

ou ssi :

$$\underset{|t|\leq n}{Moy}\, K(f(t)) = \underset{|t|\leq n}{Moy}\, K(t) + o(n)$$

Preuve À partir des propriétés 20 et 22. □

2.4 Compatibilité avec la théorie de la complexité à programmes auto-délimités.

La théorie de la complexité algorithmique de l'information avec K est naturelle, alors que la théorie avec H est plus complexe mais aussi plus harmonieuse. Cette dernière permet de définir la mesure de probabilité de Levin (cf. [LV93]). Les deux théories coexistent et évoluent en parallèle car elles ont toutes les deux leur importance et leur utilité. Pour que notre théorie de la représentation optimale puisse s'accorder avec ces deux cadres de pensée, nous prouvons que les définitions de K-optimalité et H-optimalité sont équivalentes. Pour cela, nous introduisons un lemme.

Lemme 2

$$\underset{|t|=n}{Moy}\, H(t) = \underset{|t|=n}{Moy}\, K(t) + O(\log(n))$$

Preuve Nous savons que :

$$\forall t : H(t) > K(t) + c$$

et que :

$$H(t) = K(t) + O(\log(|t|))$$

Soient n un entier. Nous savons qu'il existe un réel c tel que pour tout t de taille n :

$$\begin{aligned} & H(t) \leq K(t) + c\log(|t|) \\ \Rightarrow\ & Moy_{|t|=n} H(t) \leq Moy_{|t|=n} K(t) + c\, Moy_{|t|=n} \log(|t|) \\ & = Moy_{|t|=n} K(t) + c\log(n) \\ \Rightarrow\ & Moy_{|t|=n} H(t) = Moy_{|t|=n} K(t) + O(\log(n)) \end{aligned}$$

□

Théorème 3 Soit f une forme, f est K-optimale ssi elle est H-optimale.

Preuve Nous savons que :

$$\underset{|t|=n}{Moy}\, H(t) \;=\; \underset{|t|=n}{Moy}\, K(t) + O(\log(n))$$

et donc à fortiori :

$$\underset{|t|=n}{Moy}\, H(t) \;=\; \underset{|t|=n}{Moy}\, K(t) + o(n)$$

Donc si :

$$\underset{|t|=n}{Moy}\, K(f(t)) \;=\; n + o(n)$$

on a :

$$\underset{|t|=n}{Moy}\, H(f(t)) \;=\; n + o(n)$$

et réciproquement. □

Remarque : Sachant que les deux définitions sont équivalentes, nous dirons d'une suite K-optimale ou H-optimale, qu'elle est optimale en moyenne ou plus simplement optimale.

2.5 Deux lemmes utiles.

Nous donnons maintenant deux lemmes qui simplifient les preuves d'optimalité et de non-optimalité. Le premier stipule que la moyenne des complexités des $f(t)$ est inférieure ou égale à n plus un terme en $O(\log(n))$, si f est calculable. En effet, si la forme est calculable, on peut trouver un programme qui quel que soit t calcule $f(t)$ à partir de t. Donc la complexité de $f(t)$ est inférieure ou égale à celle de t plus un terme en $O(\log(|t|))$, l'inégalité qui se vérifie aussi sur la moyenne.

Lemme 3 Soit $f : T \longrightarrow \{0,1\}^*$ une forme calculable, alors les deux inéquations suivantes sont vraies :

$$\underset{|t|=n}{Moy}\, K(f(t)) \;\leq\; n + O(log(n))$$
$$\underset{|t|=n}{Moy}\, H(f(t)) \;\leq\; n + O(log(n))$$

Preuve Soit U une machine universelle. Si f est calculable, nous savons grâce à la thèse de Church qu'il existe un programme uniforme, capable de simuler sur U la machine qui calcule f en lui fournissant la donnée t. Ce programme inclut au mieux, un des plus petits programmes auto-délimités de t, qui lui permet de produire t. Le reste du programme est de taille constante et cette constante varie selon la machine universelle choisie U (en effet, le programme écrit en langage C n'aura pas la même taille qu'écrit en langage Pascal).

Nous avons donc :

$$H(f(t)) \;\leq\; H(t) + c$$

donc pour tout $n \in \mathbf{N}$, grâce au lemme 2 :

$$\underset{|t|=n}{Moy}\, H(f(t)) \leq n + O(\log(n))$$

Comme pour tout t, la complexité non-auto-délimitée $K(t)$ ne diffère de la complexité auto-délimitée $H(t)$ que d'un terme en $\log(|t|)$, on a aussi :

$$\underset{|t|=n}{Moy}\, K(f(t)) \leq n + O(\log(n))$$

Si T est un produit d'ensemble de *Types*, alors le programme ne fournit pas t seul, mais t_1, puis t_2, jusqu'à t_k, et doit donc les auto-délimiter. Cela coûte au plus un terme en $O(\log(n))$. □

Le deuxième lemme donne une condition suffisante pour qu'une forme soit non-optimale.

Lemme 4 Si

$$\left| n - \underset{|t|=n}{Moy}\, |f(t)| \right| \neq o(n)$$

alors f est non-optimale.

Preuve D'après le théorème d'invariance, on sait qu'il existe $c > 0$ tel quel que soit t :

$$K(f(t)) \leq |f(t)| + c$$

donc :

$$\underset{|t|=n}{Moy}\, K(f(t)) \quad \leq \quad \underset{|t|=n}{Moy}\, |f(t)| + c$$

dont on déduit que :

$$\left| n - \underset{|t|=n}{Moy}\, K(f(t)) \right| \quad \geq \quad \left| n - \underset{|t|=n}{Moy}\, |f(t)| \right|$$

Avec l'hypothèse on en déduit que :

$$\left| n - \underset{|t|=n}{Moy}\, K(f(t)) \right| \quad \neq \quad o(n)$$

On utilise la propriété suivante : si $f > g > 0$ et $g(n) \neq o(n)$ alors $f \neq o(n)$. □

2.6 Injectivité et optimalité.

Le résultat principal de cette section démontre qu'une forme injective est obligatoirement optimale en moyenne. Informellement, l'injectivité se traduit par l'association d'un code et d'un seul à chaque mot du langage. On ne peut faire moins qu'associer un code à chaque mot, si l'on désire compresser potentiellement n'importe quel mot de ce langage. Si l'on associe plus d'un code à un mot du langage, on gaspille la ressource des codes disponibles. La forme qui au couple (x, m) composé d'un texte x et d'un entier m, associe le mot x^m se comporte ainsi, pourtant elle est optimale en moyenne (cf. la propriété 28). Un codage peut donc se permettre

un certain gaspillage des codes, tout en restant une bonne représentation en moyenne vis à vis d'une certaine hypothèse de structurale. La propriété liant l'injectivité à l'optimalité en moyenne, met en valeur cette marge de manœuvre dans la conception d'un codage.

Après la preuve de ce résultat, nous rappelons les définitions de *fonction de compression* et *d'optimalité de compression* données dans [GS91]. Nous étudions aussi l'optimalité de la compression par *rangement et numérotation*, dit «*ranking*» en anglais.

Théorème 4 Soit f une forme. Si f est injective et calculable alors f est optimale en moyenne.

Ici, nous examinons le cas d'une forme $f : T \to \{0,1\}^*$ où T est un type simple (i.e. pas de la forme $T_1 \times \cdots \times T_k$), pour des raisons de clarté. Le théorème reste valide dans le cas général.

Preuve Soit f une forme calculable et injective. Montrons que :

$$\underset{|t|=n}{Moy}\, K(f(t)) = n + O(1)$$

Pour cela, nous prouvons les deux inégalités suivantes, i.e. il existe $c > 0$ et $c' > 0$ tels que :

$$\underset{|t|=n}{Moy}\, K(f(t)) \leq n + c \tag{2.7}$$

$$\underset{|t|=n}{Moy}\, K(f(t)) + c' \geq n \tag{2.8}$$

Preuve de l'inégalité 2.7.
De la calculabilité de f et du lemme 3 découle l'inégalité 2.7.

Preuve de l'inégalité 2.8.
Soit n un entier. Comme f est injective, nous avons :

$$Card\,\{f(t) : |t| = n\} = Card\,\{t : |t| = n\} = 2^n$$

La machine de Turing universelle de référence étant fixée, les plus petits programmes au sens de Kolmogorov des 2^n $f(t)$ sont au mieux les 2^n plus courtes chaînes de $\{0,1\}^*$. Donc :

$$\sum_{|t|=n} K(f(t)) \geq \sum_{i=0}^{n-1} i2^i$$

d'où :

$$\begin{aligned}
\underset{|t|=n}{Moy}\, K(f(t)) &= \frac{\sum_{|t|=n} K(f(t))}{2^n} \\
&\geq \frac{\sum_{i=0}^{n-1} i2^i}{2^n} \\
&= \frac{(n-2)2^n + 2}{2^n} \\
&= n - 2 + 2^{-n}
\end{aligned}$$

On peut donc trouver c' pour que l'inégalité 2.8 soit vérifiée. □

Définition 20 D'après [GS91], une fonction de compression d'un langage L, est une bijection sur L, qui à un mot x de longueur n, associe un code $f(x)$ de taille inférieure à n, excepté pour un nombre fini de mots.

Définition 21 De plus, on dit que f compresse L de manière optimale ssi pour tout x appartenant à L de longueur n :

$$|f(x)| \leq \left\lceil \log \left(\sum_{i=0}^{n} Card(L^i) \right) \right\rceil$$

Trois différences séparent une forme d'une fonction de compression. Tout d'abord, une fonction de compression dépend d'un langage. A chaque mot de ce langage, elle associe un code. Tandis qu'une forme est une fonction de décompression, à partir d'un code elle reconstruit le mot encodé à l'origine ; elle n'est pas relative à un langage, mais elle est basée sur une hypothèse de structure. En outre, une fonction de compression est bijective alors qu'une forme n'est que surjective.

La compression par rangement et numérotation ou ranking est une méthode de compression utilisable quel que soit le langage à compresser. Soit L un langage. En effet, r_L (pour ranking sur L) est une fonction de compression qui à un mot x quelconque, associe le nombre de mots qui dans L sont :

- soit de taille inférieure à $|x|$
- soit de taille égale et de rang inférieur ou égal selon l'ordre lexicographique.

Ranking s'adapte à tous les langages et est optimal au sens de [GS91].

Fait 3 Quel que soit le langage L, r_L compresse L de manière optimale.

Quel que soit le langage sur lequel il est appliqué, ranking est bijectif sur ce langage, mais il est dépendant de celui-ci. Sur $\{0,1\}^*$, r_L n'est plus injectif. Donc sa fonction inverse ne peut être une forme puisqu'elle n'est pas surjective sur $\{0,1\}^*$.

Fait 4 Si on restreint l'ensemble de départ à L, r_L^{-1} est injective et donc optimale en moyenne.

Preuve Résulte du théorème 4. □

Chapitre 3

Exemples de formes optimales et non optimales.

Dans ce chapitre, nous démontrons l'applicabilité de la théorie de la représentation optimale en montrant l'optimalité ou la non-optimalité pour certaines formes de bases. Les hypothèses de structure qu'elles introduisent forment un ensemble d'opérations naturelles sur des textes ou sur des k-uplets d'objets. Ces opérations sont : la concaténation, la sélection, les opérations logiques, la mise à la puissance et la suppression de digits. La dernière opération fait l'objet d'une attention particulière parce qu'elle traduit le gaspillage des codes possibles dans un codage.

3.1 Formes «concaténation», «sélection», «opérations logiques».

Tout d'abord, nous examinons une forme qui pour coder un texte z, garde un couple quelconque de sous-mots (x, y) tels que la concaténation de x et de y donne z.

Propriété 24 Soit

$$\begin{array}{rccc} f_1 : & \{0,1\}^* \times \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & (x,y) & \longmapsto & xy \end{array}$$

f_1 est une forme optimale.

Preuve Nous démontrons que :

$$\underset{|x|+|y|=n}{Moy} K(xy) = n + O(\log(n))$$

Pour cela, nous prouvons les deux inégalités suivantes, i.e. il existe $c > 0$ et $c' > 0$ tels que :

$$\underset{|x|+|y|=n}{Moy} K(xy) \leq n + c\log(n) \tag{3.1}$$

$$\underset{|x|+|y|=n}{Moy} K(xy) + c'\log(n) \geq n \tag{3.2}$$

qui avec le lemme 2 permettent de conclure.

Preuve de l'inégalité 3.1.

De la calculabilité de f_1 et du lemme 3 découle l'inégalité 2.7.

Preuve de l'inégalité 3.2.
Le fait 1 dit que :

$$K(x,y) \leq K(xy) + O(\log(K(x) + K(y)))$$

or la propriété 23 nous dit que :

$$\underset{|x|+|y|=n}{Moy} K(x,y) = n + O(\log(n))$$

donc nous savons qu'il existe $c' > 0$ et $n_0 > 0$ tels que $\forall n > n_0$:

$$\underset{|x|+|y|=n}{Moy} K(xy) + c' \log(n) \geq n$$

□

Vient la forme qui à (x,y) associe x et est non optimale.

Propriété 25 Soit

$$\begin{array}{rccc} f_2 : & \{0,1\}^* \times \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & (x,y) & \longmapsto & x \end{array}$$

f_2 n'est pas une forme optimale.

Preuve Il faut montrer que :

$$\underset{|x|+|y|=n}{Moy} K(f_2(x,y)) \neq n + o(n) \tag{3.3}$$

On partitionne l'ensemble $\{(x,y) : |x| + |y| = n\}$ en $n+1$ sous-ensembles $E_{n,i}$ (i variant de 0 à n) de 2^n couples chacun. En effet, si $E_{n,i} = \{(x,y) : |y| = i, |x| = n - i\}$, $Card(E_{n,i}) = 2^n$. Nous avons :

$$\begin{aligned} \underset{|x|+|y|=n}{Moy} |x| &= \frac{\sum_{i=0}^{n} i}{n+1} \\ &= \frac{n(n+1)}{2(n+1)} \\ &= \frac{n}{2} \\ &\neq n + o(n) \end{aligned}$$

De ce résultat et du lemme 4 découle l'inéquation 3.3. □

Nous examinons maintenant une forme qui permet de stocker toute l'information nécessaire à la construction de textes qui résultent d'un xor entre deux autres textes. Par exemple :

$$01010101 \oplus 101110 = 11101101$$

le texte 11101101 peut être stocké sous la forme du couple $(010101, 10111010)$. Si r est un texte et (s,t) un couple de textes tels que $r = s \oplus t$, cette forme code r par le couple (t,s). Elle

conserve plus d'informations qu'il n'est vraiment nécessaire de garder. Nous allons montrer qu'elle n'est pas optimale.

Propriété 26 Soit

$$\begin{array}{rccc} f_3 : & \{0,1\}^* \times \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & (x,y) & \longmapsto & x \oplus y \end{array}$$

f_3 n'est pas une forme optimale.

Preuve Il nous faut montrer que :

$$\underset{|x|+|y|=n}{Moy} K(f_3(x,y)) \neq n + o(n) \tag{3.4}$$

Pour cela nous allons d'abord calculer $Moy_{|x|+|y|=n} K(x \oplus y)$, en partitionnant l'ensemble $E_n = \{(x,y) \in (\{0,1\}^*)^2 : |x| + |y| = n\}$ en $n+1$ sous-ensembles $E_{n,i} = \{(x,y) \in (\{0,1\}^*)^2 : |x| = i, |y| = n-i\}$ (i variant de 0 à n) de 2^n éléments chacun. Remarquons tout d'abord que le résultat du xor entre deux textes a la même longueur que le plus grand des deux textes :

$$\forall (x,y) \in (\{0,1\}^*)^2, |x \oplus y| = \max(|x|, |y|)$$

Les rôles de x et y étant symétriques, on peut calculer la moyenne en ne s'intéressant qu'à une moitié des $n+1$ ensembles $E_{n,i}$. Nous traitons ici le cas n impair, le cas n pair se traite de la même manière. On a :

$$\begin{aligned} \underset{|x|+|y|=n}{Moy} |x \oplus y| &= \frac{\sum_{i=0}^{n} \left[Card(E_{n,i})\, Moy_{E_{n,i}} |x \oplus y| \right]}{\sum_{i=0}^{n} Card(E_{n,i})} \\ &= \frac{\sum_{i=0}^{n} 2^n\, Moy_{E_{n,i}} |x \oplus y|}{(n+1)2^n} \\ &= \frac{2\sum_{i=0}^{n/2} Moy_{E_{n,i}} |x \oplus y|}{n+1} \\ &= \frac{2\sum_{i=0}^{n/2} |n-i|}{n+1} \\ &= \frac{2}{n+1}\left[(\frac{n}{2}+1)n - \frac{1}{2}(\frac{n}{2}+1)\frac{n}{2} \right] \\ &= \frac{3(n+2)n}{4(n+1)} \\ &= \frac{3(n+2)}{4(1+\frac{1}{n})} \end{aligned}$$

donc :

$$\begin{aligned} \underset{|x|+|y|=n}{Moy} |x \oplus y| &< \frac{3n}{4} + \frac{3}{2} \\ &\neq n + o(n) \end{aligned}$$

De ce résultat et du lemme 4 découle l'inéquation 3.4. □

Remarquons que la propriété et la preuve restent valides si la forme réalise un «et» ou un «ou» logiques entre x et y.

3.2 Formes «puissance d'un mot».

Nous examinons l'optimalité des formes qui mettent un mot à la puissance (crée le carré, le cube, ..., d'un mot), pour une puissance fixée et pour une puissance variable. Soit un entier k, la forme qui stocke le motif x pour tout texte de la forme x^k, est injective et donc optimale.

Propriété 27 Soient k un entier et

$$\begin{array}{rccc} f_{4,k}: & \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & x & \longmapsto & x^k \end{array}$$

$f_{4,k}$ est une forme optimale.

Preuve La forme est injective, donc d'après le théorème 4, elle est optimale en moyenne. □

Nous considérons tous les textes construits par répétition d'un motif, tels que nous les avons cités en exemple dans l'introduction. Pour ces textes là, la forme suivante est optimale. L'utilisation de cette forme de représentation trouve des applications en compression de séquences génétiques [RDDD95, RDDD94], qui contiennent «répétitions en tandem», i.e. des segments de faible complexité [Yoc92].

Propriété 28 Soit

$$\begin{array}{rccc} f_5: & \{0,1\}^* \times \mathbf{N} & \longrightarrow & \{0,1\}^* \\ & (x,m) & \longmapsto & x^m \end{array}$$

f_5 est une forme optimale.

Preuve Nous démontrons que :

$$\underset{|m|+|x|=n}{Moy} K(x^m) = n + O(\log(n))$$

Pour cela, nous prouvons les deux inégalités suivantes, i.e. il existe $c > 0$ et $c' > 0$ tels que :

$$\underset{|m|+|x|=n}{Moy} K(x^m) \leq n + c\log(n) \tag{3.5}$$

$$\underset{|m|+|x|=n}{Moy} K(x^m) + c'\log(n) \geq n \tag{3.6}$$

qui avec le lemme 2 permettent de conclure.

Preuve de l'inégalité 3.5.
De la calculabilité de f_5 et du lemme 3 découle l'inégalité 2.7.

Preuve de l'inégalité 3.6.
Soient n et $i \in \mathbf{N}$ tels que $i < n$. On peut partitionner l'ensemble des couples (x,m) tels que $|x| + |m| = n, |x| \neq 0, |m| \neq 0$ en $n-1$ sous-ensembles de 2^n couples chacun. En effet, si $E_{n,i} = \{(x,m) : |m| = i, |x| = n-i\}$, $Card(E_{n,i}) = 2^n$. On peut donc former par f_5, 2^n textes différents deux à deux, à partir des 2^n couples (x,m) de $E_{n,i}$. Par le même raisonnement que celui employé dans la propriété 20, nous affirmons qu'au mieux, les plus petits programmes des ces 2^n textes ont les longueurs des 2^n plus courtes chaînes de $\{0,1\}^*$, i.e. la totalité des

chaînes de longueur inférieure ou égale à $n-1$ plus une de longueur n. Nous en déduisons l'inégalité suivante :

$$\begin{aligned} \underset{|m|=i,|x|=n-i}{Moy} K(x^m) &\geq \frac{n+\sum_{j=0}^{n-1} j2^j}{Card(E_{n,i})} \\ &= \frac{(n-2)2^n+2+n}{2^n} \\ &= n-2+\frac{n+2}{2^n} \end{aligned}$$

Par associativité de la moyenne, nous en déduisons :

$$\underset{|m|+|x|=n}{Moy} K(x^m) \geq n-2$$

Donc il existe $n_0 > 0$ et $c' > 0$ tels que :

$$\forall n \geq n_0 : \quad \underset{|m|+|x|=n}{Moy} K(x^m) + c' \log(n) \quad \geq \quad n$$

□

3.3 Formes «suppression de bits» selon un sous-ensemble de N.

Les formes envisagées ici suppriment des digits au texte en entrée. Nous définissons une forme générique $f_{6,A}$ qui dépend d'un paramètre : l'ensemble A. Nous montrons ensuite l'optimalité ou la non-optimalité en moyenne de $f_{6,A}$ pour quelques cas particuliers de A. Pour décoder un texte x de $\{0,1\}^*$, la forme lui supprime les digits dont les rangs ne sont pas des entiers de $A^{\leq |x|}$. Selon la densité de A, $f_{6,A}$ est ou n'est pas optimale en moyenne. Nous examinons divers cas selon les propriétés de l'ensemble A et commentons le phénomène de gaspillage des codes associé à ce type de formes.

Définition 22 Soient A un sous-ensemble de $\mathbb{N}$,

$$\begin{array}{rccc} f_{6,A} : & \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & (x)_{0\leq i\leq n-1} & \longmapsto & (x)_{i\in A^{\leq n}} \end{array}$$

où $(x)_{i\in A^{\leq n}}$ est la sous-suite des lettres dont le rang appartient à $A^{\leq n}$.

Remarque : $f_{6,A}$ est calculable ssi A est récursif.

3.3.1 Divers cas de densité du sous-ensemble.

Premier exemple : soit la forme qui a un texte, (i.e. une suite finie de lettres) associe le sous-mot formé par les lettres de rang pairs (i.e. la sous-suite de rang pairs). Elle n'utilise pas une partie fixée du code, précisément la moitié du code, pour décoder le texte original. Cette forme n'est pas optimale en moyenne. Si on nomme P l'ensemble des entiers pairs, cette forme est $f_{6,P}$.

Propriété 29 Soient P l'ensemble des entiers pairs et la forme :

$$\begin{array}{rccc} f_{6,P}: & \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & (x)_{0\leq i\leq n-1} & \longmapsto & (x)_{2i} \end{array}$$

$f_{6,P}$ n'est pas une forme optimale.

Preuve Il nous faut montrer que :

$$\underset{|x|=n}{Moy}\, K(f_{6,P}(x)) \neq n + o(n) \tag{3.7}$$

Soient n un entier et x un texte tel que $|x| = n$. On peut supposer sans perte de généralités que n est pair. Remarquons alors que $|f_{6,P}(x)|$ vaut la moitié de la taille de x. Nous avons :

$$\underset{|x|=n}{Moy}\, |f_{6,P}(x)| \;=\; \frac{n}{2}$$

D'après le lemme 4, cette condition est suffisante pour obtenir l'inéquation 3.7. □

Nous allons examiner tous d'abord le cas A fréquent.

Définition 23 Soit A un sous-ensemble de $\mathbf{N}$. A est fréquent ssi :

$$\forall a > 0, \exists n_a, \forall n > n_a : \frac{Card(A^{\leq n})}{n} > 1 - a$$

Théorème 5 Soit A un sous-ensemble de $\mathbf{N}$ fréquent, alors $f_{6,A}$ est une forme optimale.

Preuve Nous démontrons que :

$$\underset{|x|=n}{Moy}\, K(f_{6,A}(x)) = n + o(n)$$

Par définition, si A est fréquent on a :

$$\forall a > 0, \exists n_a, \forall n > n_a : \frac{Card(A^{\leq n})}{n} > 1 - a$$

Soit $a > 0$, par hypothèse il existe un rang n_a tel que :

- pour tout $n > n_a$: $Card(A^{\leq n}) > n(1 - \frac{a}{2})$
- $\frac{2}{n_a} \leq \frac{a}{2}$

Soit $n > n_a$. Pour calculer l'image d'un texte de taille n, $f_{6,A}$ supprime les bits dont les rangs ne sont pas dans $A^{\leq n}$. Les rangs de ces bits sont fixés et donc indépendants du texte dont on calcule l'image. L'image de $\{x, |x| = n\}$ est donc l'ensemble des textes de taille $Card(A^{\leq n})$. Cet ensemble image contient $2^{Card(A^{\leq n})}$ éléments. Par le même raisonnement que celui employé dans la propriété 20, nous affirmons qu'au mieux, les plus petits programmes des textes de $f_{6,A}(\{x, |x| = n\})$ ont au moins les longueurs des $2^{Card(A^{\leq n})}$ plus courtes chaînes de $\{0,1\}^*$, i.e. la totalité des chaînes de longueur inférieure ou égale à $Card(A^{\leq n}) - 1$ plus une

de longueur $Card(A^{\leq n})$. Notons m la valeur $n - Card(A^{\leq n})$. Nous en déduisons l'inégalité suivante :

$$\begin{aligned}\sum_{y \in f_{6,A}(\{x,|x|=n\})} K(y) &\geq n - m + \sum_{j=0}^{n-m-1} j2^j \\ &= n - m + (n - m - 2)2^{n-m} + 2 \\ &\geq (n - m - 2)2^{n-m}\end{aligned}$$

En outre, puisque l'on supprime $n - Card(A^{\leq n})$ bits à un code pour obtenir son image, il existe pour chaque image $2^{n-Card(A^{\leq n})} = 2^m$ antécédents (qui ne diffèrent entre eux que par les positions dont le rang est dans le complémentaire de $A^{\leq n}$ et inférieur à n). Or la moyenne des complexités de $f_{6,A}(x)$ pour tous les x de taille n, tient compte de ces duplications de code[1]. Dans ce calcul de moyenne la complexité de chaque texte image est comptée 2^m fois. Donc nous avons :

$$\begin{aligned}\underset{|x|=n}{Moy}\, K(f_{6,A}(x)) &= \frac{2^m \sum_{y \in f_{6,A}(\{x,|x|=n\})} K(y)}{2^n} \\ &\geq \frac{2^m (n - m - 2)2^{n-m}}{2^n} \\ &= n - m - 2\end{aligned}$$

Or d'après l'hypothèse, nous savons que $Card(A^{\leq n}) > n(1-\frac{a}{2})$, or comme $Card(A^{\leq n}) = n-m$ on a :

$$\begin{array}{rccc} & Card(A^{\leq n}) & > & n(1-\frac{a}{2}) \\ \Rightarrow & n-m & > & n(1-\frac{a}{2}) \\ \Rightarrow & n-m & > & n-\frac{na}{2} \\ \Rightarrow & m & < & \frac{na}{2} \end{array}$$

donc :

$$\begin{aligned}\frac{\left|n - Moy_{|x|=n} K(f_{6,A}(x))\right|}{n} &= \frac{n - Moy_{|x|=n} K(f_{6,A}(x))}{n} \\ &\leq \frac{n - n + m + 2}{n} \\ &\leq \frac{m+2}{n} \\ &< \frac{a}{2} + \frac{a}{2} \\ &< a\end{aligned}$$

□

Ensuite vient le cas non-fréquent, i.e. le contraire de fréquent.

Théorème 6 Soit A un sous-ensemble de $\mathbf{N}$ non fréquent, alors $f_{6,A}$ n'est pas une forme optimale.

1. Nous parlons de duplication de codes pour indiquer que plusieurs codes peuvent coder le même texte, i.e. qu'une image d'une forme peut avoir plusieurs antécédents.

Preuve A n'est pas fréquent veut dire que :

$$\exists a \in]0,1[,\ \forall n_a > 0, \exists n > n_a : \frac{Card(A^{\leq n})}{n} < a$$

et donc si x est tel que $|x| = n$ on a $|f_{6,A}(x)| < an$. D'où $Moy_{|x|=n} |f_{6,A}(x)| < an$. Grâce au lemme 4, on obtient la non-optimalité de $f_{6,A}$. □

Nous introduisons d'autres catégories d'ensembles, nous examinons le comportement de $f_{6,A}$.

Notation 10 Soit A un sous-ensemble de $\mathbf{N}$. On note (P_n) la suite de terme :

$$\forall n \in \mathbf{N}^* :\ P_n = \frac{Card(A^{\leq n})}{n}$$

(P_n) prend ses valeurs dans l'intervalle réel $[0,1]$.

Pour toute la suite de la sous-section nous notons A un sous-ensemble de $\mathbf{N}$. Nous donnons tout d'abord une équivalence de la notion de fréquent en termes de lim inf P_n.

Propriété 30 A est fréquent ssi lim inf $P_n = 1$.

Preuve En effet, si A est fréquent alors la limite de (P_n) existe et est égale à 1. Donc lim inf $P_n = 1$. D'autre part, si lim inf $P_n = 1$ alors la limite sup de (P_n) aussi et par conséquent lim $P_n = 1$ car lim P_n est comprise entre sa limite sup et sa limite inf. □

On déduit la propriété suivante.

Propriété 31 A est non fréquent ssi lim inf $P_n < 1$.

Définition 24 Soit A un sous-ensemble de $\mathbf{N}$. A est *éparpillé* (en anglais sparse) i.e. le complémentaire d'un ensemble fréquent ssi :

$$\forall a > 0, \exists n_a, \forall n > n_a : \frac{Card(A^{\leq n})}{n} < a$$

i.e. ssi lim sup $P_n = 0$

Propriété 32 Si A est éparpillé, alors A est non fréquent.

Preuve Par définition, si A est éparpillé on a :

$$\text{lim sup } P_n = 0 \quad \Rightarrow \quad \text{lim inf } P_n = 0$$

et donc A est non fréquent. □
Par conséquent, si A est éparpillé, $f_{6,A}$ n'est pas optimale en moyenne.

Définition 25 Soit A un sous-ensemble de $\mathbf{N}$. A n'est pas éparpillé ssi :

$$\exists a > 0, \forall n_a, \exists n > n_a : \frac{Card(A^{\leq n})}{n} \geq a$$

i.e. ssi lim sup $P_n > 0$

On ne peut rien conclure sur $f_{6,A}$, car si A n'est pas éparpillé, il peut être fréquent ou non fréquent. Un autre cas à considérer est celui des ensembles dits de *densité non nulle.*

Définition 26 Soit A un sous-ensemble de $\mathbf{N}$. A est de densité non nulle (il ne s'agit pas du sens topologique de cette qualité) ssi :

$$\exists a > 0, \exists n_a, \forall n > n_a : \frac{Card(A^{\leq n})}{n} > a$$

i.e. ssi $\liminf P_n > 0$

Comme pour le cas non éparpillé, on ne peut rien conclure sur $f_{6,A}$. On examine le cas des ensembles qui ne sont pas de densité non nulle.

Définition 27 Soit A un sous-ensemble de $\mathbf{N}$. A n'est pas de densité non nulle ssi :

$$\forall a > 0, \forall n_a, \exists n > n_a : \frac{Card(A^{\leq n})}{n} \leq a$$

i.e. ssi $\liminf P_n = 0$

Propriété 33 Si A n'est pas de densité non nulle, alors A est non fréquent.

Preuve Par définition, si A n'est pas de densité non nulle on a :

$$\liminf P_n = 0 \quad \Rightarrow \quad \liminf P_n < 1$$

et donc A est non fréquent. □

Par conséquent, si A n'est pas de densité non nulle, $f_{6,A}$ n'est pas optimale en moyenne.

On peut ensuite regarder les ensembles dont le complémentaire est ou n'est pas de densité non nulle.

Définition 28 Soit A un sous-ensemble de $\mathbf{N}$. A a un complémentaire de densité non nulle ssi :

$$\exists a > 0, \exists n_a, \forall n > n_a : \frac{Card(A^{\leq n})}{n} < a$$

i.e. ssi $\limsup P_n < 1$

Propriété 34 Si A a un complémentaire de densité non nulle, alors A est non fréquent.

Preuve Par définition, si A a un complémentaire de densité non nulle, on a :

$$\limsup P_n < 1 \quad \Rightarrow \quad \liminf P_n < 1$$

et donc A est non fréquent. □

Par conséquent, si A a un complémentaire de densité non nulle, $f_{6,A}$ n'est pas optimale en moyenne.

Définition 29 Soit A un sous-ensemble de $\mathbf{N}$. A n'a pas un complémentaire de densité non nulle ssi :

$$\forall a > 0, \forall n_a, \exists n > n_a : \frac{Card(A^{\leq n})}{n} \geq a$$

i.e. ssi $\limsup P_n = 1$

Comme pour le cas non éparpillé, on ne peut rien conclure sur $f_{6,A}$.

3.3.2 Commentaire sur la duplication de codes.

A propos de l'injectivité, nous avons souligné qu'une forme peut associer plusieurs codes pour représenter un seul mot. Nous appelons cela duplication ou gaspillage des codes. Par définition une forme est injective si elle associe un seul code à chaque mot. Nous avons vu que si une forme injective est calculable alors elle est optimale. Cependant certaines formes ont des codes dupliqués et sont optimales.

Une forme peut associer plusieurs codes à un mot si lors du décodage elle ne tient pas compte d'une partie des informations pour reconstruire le texte original. Le cas le plus simple d'une telle situation est lorsqu'on supprime certains bits du code pour produire un texte plus court que le code: la forme $f_{6,A}$ fait cela. Jusqu'à quel point une forme peut accepter de dupliquer certains codes tout en restant optimale? Peut-on décider de l'optimalité ou de la non-optimalité d'une forme en connaissant simplement combien de codes sont dupliqués et combien de fois?

Pour cela, l'étude des formes $f_{6,A}$ donne des éléments de réponse, car ces formes sont des exemples typiques de duplication des codes. En effet, lors du décodage $f_{6,A}$ supprime les bits dont les rangs sont fixés par l'ensemble paramètre A (ce sont les rangs qui n'appartiennent pas à A). La forme générique $f_{6,A}$ est la plus simple procédure qui ne fait «rien d'autre que gaspiller des codes» ; son comportement est intrinsèquement lié au phénomène de gaspillage des codes. Chaque bit enlevé indique que le décodage ne tient pas compte d'un caractère qui peut prendre deux valeurs 0 ou 1. Cette élimination implique que 2 antécédents qui ne diffèrent qu'à cette position auront la même image. Cela entraîne deux conséquences intéressantes:

- la suppression systématique par $f_{6,A}$ de m bits sur un code de taille n, donne à chaque image 2^m antécédents;

- la duplication des codes est répartie uniformément entre toutes les images de l'ensemble $f_{6,A}(\{x : |x| = n\})$. Cette uniformité est essentielle pour démontrer le théorème 5. Si le gaspillage des codes est inéquitablement réparti, sans loi apparente, on ne peut qu'approximer très largement la complexité moyenne de l'ensemble suscité, ce qui en général empêche de conclure à l'optimalité.

Remarque : Que la forme $f_{6,A}$ supprime des digits ou les remplace par des digits constants (toujours à 1 par exemple), ne changerait pas la complexité moyenne des textes décodés.

Ainsi l'étude de ces formes permet en premier lieu, de voir à quel point le gaspillage de codes influe sur l'optimalité. La principale indication est donnée par le théorème 5 : $f_{6,A}$ reste optimale même si 2^m codes de taille n peuvent engendrer la même image (chaque texte a 2^m codes possibles) à la condition que $\frac{m}{n}$ converge vers 0 lorsque n tend vers l'infini (pour un cas où la duplication des codes est équitablement répartie entre tous les textes). Avec d'autres fonctions pour m, $f_{6,A}$ n'est plus optimale. C'est le cas de la forme $f_{6,P}$ par exemple (cf. la propriété 29).

Par ailleurs, la forme f_1, montre un cas où le nombre de codes pour un texte est linéaire par rapport à la taille du code: si on associe (x, y) à $x.y$ et que $|x| + |y| = n$, on peut trouver $n + 1$ couples (x, y) qui donnent la même image (un couple par position de $x.y$). f_1 est aussi optimale (voir la propriété 24). Bien d'autres cas se révèlent plus difficiles à statuer car la duplication n'est pas également répartie, on peut donc pas en tenir compte dans le calcul de la moyenne (par exemple, il n'en est pas tenu compte dans la preuve d'optimalité de f_5).

Chapitre 4

Composition et optimalité.

Dans ce chapitre, nous examinons les possibilités de composer les formes. Bien que notre premier résultat prouve que la composition ne conserve pas l'optimalité, deux autres théorèmes permettent d'utiliser la composition pour prouver, dans certaines conditions, l'optimalité d'une forme en la décomposant en formes plus simples. La propriété 36 exhibe un exemple d'utilisation de la décomposition.

4.1 Résultats de composition de formes.

Théorème 7 L'optimalité n'est pas conservée par la composition.

Preuve Voici un contre-exemple. Soit NC l'ensemble des entiers non carrés et non nuls. NC est fréquent. On considère la forme $f_{6,NC}$ prouvée optimale en moyenne dans le théorème 5. Soit

$$\begin{array}{rccc} g: & \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & x & \longmapsto & w \end{array}$$

où $w_i = 0$ si $i \notin NC$ et $w_{i^2} = x_i$ sinon. g est injective et donc optimale en moyenne.

Notons j la forme composée $f_{6,NC} \circ g$. Pour tout n et pour tout x tels que $|x| = n$ nous avons :

$$\begin{aligned} j(x) &= f_{6,NC} \circ g(x) \\ &= f_{6,NC}(0x_1 00 x_2 000 x_3 0 \cdots 0 x_n) \\ &= 0^{n^2-n+1} \end{aligned}$$

Un programme qui écrit $j(x)$ se déroule de la manière suivante : il calcule $|x|$ et récupère le résultat dans une variable l, puis boucle $l^2 - l + 1$ fois en écrivant 0. Ce programme est de la longueur de $K(|x|)$ majorée d'une constante qui formalise la description ci-dessus. Or à une constante près, $K(|x|) \leq \log(|x|)$ et donc $K(j(x)) \leq \log(|x|) + c$. Nous avons alors :

$$\begin{aligned} \underset{|x|=n}{Moy}\, K(j(x)) &\leq \log(n) + c \\ &\neq n + o(n) \end{aligned}$$

□

Théorème 8 La composition de formes injectives et calculables conserve l'optimalité en moyenne.

Preuve La composition préserve la calculabilité et l'injectivité. □

Théorème 9 Soient deux formes calculables f et g avec f injective. Alors les deux propositions suivantes sont équivalentes :

1. g est K-optimale en moyenne,
2. $f \circ g$ est K-optimale en moyenne.

Preuve Notons h la forme composée $f \circ g$. Nous devons montrer l'implication dans les deux sens.

Démonstration de $\Rightarrow$
Nous supposons que g est K-optimale et nous démontrons que :

$$\underset{|x|=n}{Moy}\, K(h(x)) \quad = \quad n + o(n)$$

Soit n un entier. Comme f est injective elle associe un texte différent à chaque texte de $\{g(x);\ |x| = n\}$. Donc la moyenne des complexités des textes des ensembles $\{g(x);\ |x| = n\}$ et $\{f(g(x));\ |x| = n\}$ sont équivalentes à une constante près, grâce à la calculabilité de f. Donc :

$$\left| \underset{|x|=n}{Moy}\, K(f(g(x))) - \underset{|x|=n}{Moy}\, K(g(x)) \right| = o(n) \tag{4.1}$$

De plus on sait que g est optimale donc :

$$\left| \underset{|x|=n}{Moy}\, K(g(x)) - n \right| = o(n) \tag{4.2}$$

De 4.1 et 4.2 et grâce à l'inégalité triangulaire, on obtient :

$$\left| \underset{|x|=n}{Moy}\, K(f(g(x))) - n \right| \quad = \quad o(n)$$

Démonstration de $\Leftarrow$.
On suppose que $f \circ g$ est K-optimale et nous montrons que g l'est aussi, i.e. que :

$$\underset{|x|=n}{Moy}\, K(g(x)) \quad = \quad n + o(n)$$

Par hypothèse, nous savons que :

$$\underset{|x|=n}{Moy}\, K(h(x)) = n + o(n) \tag{4.3}$$

Soit n un entier. Comme f est injective elle associe un texte différent à chaque texte de $\{g(x);\ |x| = n\}$. Donc la moyenne des complexités des textes des ensembles $\{g(x);\ |x| = n\}$

et $\{f(g(x));\ |x| = n\}$ sont équivalentes à une constante près, grâce à la calculabilité de f. Donc :

$$\left| \underset{|x|=n}{Moy}\, K(f(g(x))) - \underset{|x|=n}{Moy}\, K(g(x)) \right| = o(n) \tag{4.4}$$

De 4.3 et 4.4, grâce à l'inégalité triangulaire, on obtient :

$$\left| \underset{|x|=n}{Moy}\, K(g(x)) - n \right| \quad = \quad o(n)$$

□

4.2 Exemples de décomposition.

Nous montrons des exemples de formes dont l'optimalité se ramène à l'optimalité d'une autre forme par composition. Supposons que nous ayons un texte y infini sur l'alphabet $\{0,1\}$. Si l'on considère que les bits de y sont numérotés à partir de 0, alors y représente un sous-ensemble de $\mathbb{N}$ que nous notons Y, selon le code : $i \in Y$ ssi $y_i = 1$. Nous définissons les formes qui réalisent une opération logique entre y et le texte qu'elles ont en argument.

Définition 30 Soient y un texte de $\{0,1\}^\infty$, $\mho$ un symbole représentant une opération logique parmi $\vee, \wedge, \oplus$. Soit $f_{\mho,y}$, une forme telle que :

$$\begin{array}{rccc} f_{\mho,y} : & \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ & x & \longmapsto & x \mho y \end{array}$$

Dans l'opération logique $\mho$, x et y sont cadrés à gauche, i.e. $x \mho y = (z_i)_{1 \leq i \leq |x|}$ où $\forall i :\ z_i = x_i \mho y_i$.

Nous examinons les trois cas des opérations logiques $\oplus, \vee$ et $\wedge$.

Propriété 35 Quel que soit y un texte de $\{0,1\}^\infty$, $f_{\oplus,y}$ est une forme optimale.

Preuve Cette forme est injective, donc optimale. □

Propriété 36 Soit y un texte de $\{0,1\}^\infty$, $f_{\vee,y}$ (respectivement $f_{\wedge,y}$) est une forme optimale ssi $f_{6,Y}$ est optimale.

Preuve Soit x un texte de $\{0,1\}^*$. A toutes les positions $i \leq |x|$ où y_i est à 1 (resp. à 0), $x_i \vee y_i$ (resp. $x_i \wedge y_i$) vaut 1 (resp. 0). Donc, calculer $f_{\vee,y}(x)$ (resp. $f_{\wedge,y}(x)$) revient à remplacer tous les bits de x dont le rang appartient à Y par des 1 (resp. par des 0). Or un changement peut toujours se faire par une suppression puis une insertion.

Soit $f_{Ins_1,Y}$ (resp. $f_{Ins_0,Y}$), la forme qui à un texte x insère des bits à 1 (resp. à 0) aux rangs de Y, en décalant vers la droite les bits de x. Cette forme est injective (elle est similaire à la forme g utilisée dans la preuve de le théorème 7, qui est $f_{Ins_0,NC}$). D'après le calcul explicité ci-dessus, il est évident que : $f_{\vee,y} = f_{Ins_1,Y} \circ f_{6,Y}$ (resp. que $f_{\wedge,y} = f_{Ins_0,Y} \circ f_{6,Y}$). Donc, d'après le théorème 9, $f_{\vee,y}$ (respectivement $f_{\wedge,y}$) est une forme optimale ssi $f_{6,Y}$ est optimale, i.e. ssi Y est un ensemble fréquent. □

Bibliographie

[AALM90] A. Apostolico, M. Attalah, L. Larmore, and S. McFaddin. Efficient Parallel Algorihtms for String Editing Problems. *SIAM Journal of Computing*, 19(5):968–988, 1990.

[AF85] A. Apostolico and A.S. Fraenkel. Robust transmissions of unbounded strings using Fibonacci representations. Technical Report CS85-14, Dpt of Applied Mathematics, The Weizmann Institute of Science, Rehovot Israel, 1985.

[AP88] A. Aggarwal and J. Park. Notes on Searching in Multidimensional Monoton Arrays. In *29th Symposium on Foundations of Computer Science*, pages 497–512, 1988.

[BBE+85] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[BCW90] Bell, Cleary, and Witten. *Text Compression.* Prentice Hall, 1990.

[Bel86] Thimoty C. Bell. Better OPM/L text compression. *IEEE Trans. Communications*, COM-34(12):1176–1182, December 1986.

[Bel87] Thimoty C. Bell. *A unifying theory and improvements for existing approaches to text compression.* PhD thesis, Dpt. of Computer Science, University of Canterbury, Christchurch, New Zealand, 1987.

[Ber89] G. Bernardi. The isochore organization of the human genome. *Annual Review Genetics*, 23:637–661, 1989.

[BKMD93] A. Behn-Krappa, J. Mollenhauer, and W. Doerfler. Triplet repeat sequences in human DNA can be detected by hybridization to a synthetic (5'-cgg-3')17 oligodeoxyribonucleotide. *FEBS Letters*, 333:248–250, 1993.

[BM77] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20:762–772, 1977.

[BMGB89] G. Bernardi, D. Mouchiroud, C. Gautier, and G. Bernardi. Compositional patterns in vertebrate genomes: conservation and change in evolution. *Journal of Molecular Evolution*, 28:7–18, 1989.

[BW94] G. Benson and M.S. Waterman. A Method for Fast Database Search for All k-nucleotide Repeats. In *Symposium on Multiple Alignment*, Seattle, USA, 1994. IEEE Computer Society Press.

[Cal94] C. Calude. *Information and Randomness: an Algorithmic Perspective.* Springer-Verlag, 1994.

[CR94] Maxime Crochemore and Wojciech Rytter. *Text Algorithms.* Oxford University Press, 1994.

[Cre76] E. Mac Creight. A space-economical suffix tree construction algorithm. *Journal of the Association of Computing Machinery*, 23(2):262–272, April 1976.

[CT91] Tom M. Cover and Joy A. Thomas. *Information Theory.* Wiley Series in Telecommunications. A. Wiley-Interscience, 1991.

[DAAa94] B. Dujon, D. Alexandraki, B. André, and al. Complete DNA sequence of yeast chromosome XI. *Nature*, 369:371–378, 1994.

[Dan93] Antoine Danchin. Le séquençage des petits génomes. *La Recherche*, 1993.

[Del94] Jean-Paul Delahaye. *Information, complexité et hasard.* Hermès, Paris, 1994.

[FAA+94] H. Feldmann, M. Aigle, G. Aljinovic, B. André, M.C. Baclet, and al. Complete DNA sequence of yeast chromosome II. *EMBO Journal*, 13(5):795–809, 1994.

[FLSS92] V. Fischetti, G. Landau, J. Schmidt, and P. Sellers. Identifying Periodic Occurrences of a Template with Applications to Protein Structure. In *3rd Annual Symposium of Combinatorial Pattern Matching*, pages 111–120, 1992.

[GMM+84] M. Gouy, F. Milleret, C. Mugnier, M. Jacobzone, and C. Gautier. ACNUC: a nucleic acid sequence data base and analysis system. *Nucleic Acid Research*, 12:121–127, 1984.

[GS91] Andrew V. Goldberg and Michael Sipser. Compression and Ranking. *SIAM Journal of Computing*, 1991.

[GSP+91] E. Gilson, W. Saurin, D. Perrin, S. Bachelier, and M. Hofnung. *Nucleic Acids Research*, 19:1375–1383, 91.

[GT93a] Stéphane Grumbach and Fariza Tahi. Compression of DNA Sequences. In *Data Compression Conference*, 1993.

[GT93b] Stéphane Grumbach and Fariza Tahi. A New Challenge for Compression Algorithms: Genetic Sequences. *Journal of Information Processing and Management*, 1993.

[GT95] Stéphane Grumbach and Fariza Tahi. Compression et compréhension de séquences nucléotidiques. *Technique et science informatique*, 14(217-233), 1995.

[HBM+94] H. Hummerich, S. Baxendale, R. Mott, S.F. Kirby, M.E. MacDonald, J. Gusella, H. Lehrach, and G.P. Bates. Distribution of trinucleotide repeat sequences across a 2mbp region containing the huntington's disease gen. *Human Molecular Genetics*, 3:73–78, 1994.

[HD94] Alain Hénaut and Antoine Danchin. Escherichia coli in silico. In F.C. Neidhart, editor, *Escherichia coli and Salmonella typhimurium cellular and molecular biology.* American Society for Microbiology, Washington DC, 1994.

[Hel91] Gilbert Held. *Data Compression.* John Wiley and Sons, Ltd, 3rd edition, 1991.

[HHL+89] C.A. Hutchison, S.C. Hardies, D.D. Loeb, W.R. Shehee, and M.H. Edgell. LINES and Related Transposons: Long Interspersed Repeated Sequences in the Eucaryotic Genome. In D.E. Berg and M.M. Howe, editors, *Mobile DNA*, pages 593–618. American Society of Microbiology, Whashington DC, 1989.

[HHZ+94] J. Han, C. Hsu, Z. Zhu, J.W. Longshore, and W.H. Finley. Over-representation of the disease associated (CAG) and (CGG) repeats in the human genome. *Nucleic Acids Research*, 22:1735–1740, 1994.

[Huf52] D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. In *Proceedings of the Institure of Electrical and Radio Engineers*, volume 40, pages 1098–1101, 1952.

[Ja94] M. Johnston and al. Complete nucleotide sequence of Saccharomyces cerevisiae chromosome VIII. *Science*, 265(20):77–82, 1994.

[KBS+93] S. Karlin, B.E. Blaisdell, R. Sapolsky, L. Cardon, and C. Burge. Assessments of DNA inhomogeneities in yeast chromosome III. *Nucleic Acids Research*, 21:703–711, 1993.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6:323–350, 1977.

[KR88] J.R. Korenberg and M.C. Rykowski. Human Genome Organisation : Alu, LINES and the Molecular Structure of Metaphase Chromosome Bands. *Cell*, 53:391–400, 1988.

[Lev66] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reverseals. *Soviet Phys. Dokl*, 10:707–710, 1966.

[Lew92] Benjamin Lewin. *Gènes*, volume III. Médecine et Sciences Flammarion, 1992.

[LH87] Debra A. Lelewer and Daniel S. Hirschberg. Data Compression. *ACM Computing Surveys*, 19(3):261–296, 1987.

[LMMH89] G. Lenaers, L. Maroteaux, B. Michot, and M. Herzog. Dinoflagellates in Evolution. A Molecular Phylogenetic Analysis of Large Subunit Ribosomal RNA. *Journal of Molecular Evolution*, 29:40–51, 1989.

[LS93] Gad M. Landau and Jeanette P. Schmidt. An Algorithm for Approximate Tandem Repeats. In *Proc. 4th Symp. Combinatorial Pattern Matching*, volume 648 of *LNCS*, pages 120–133. Springer-Verlag, 1993.

[LV90] Ming Li and Paul M.B. Vitanyi. *Handbook of Theoretical Computer Science*, chapter Kolmogorov Complexity and its Applications, pages 189–254. Elsevier, 1990.

[LV93] Ming Li and Paul M.B. Vitanyi. *Introduction to Kolmogorov Complexity.* Springer-Verlag, 1993.

[MDBB93] B. Michot, L. Despres, F. Bonhomme, and J-P. Bachellerie. Conserved secondary structures in the ITS2 of trematode pre-rRNA. *FEBS*, 316(3):247–252, 1993.

[Mil93] Aleksandar Milosavljevič. Discovering Sequence Similarity by the Algorithmic Significance. In *Intelligent Systems for Molecular Biology*, pages 284–291. AAAI Press, 1993.

[MJ93a] Aleksandar Milosavljevič and Jerzy Jurka. Discovering Simple DNA Sequences by the Algorithmic Significance Method. *CABIOS*, 9(4):407–411, 1993.

[MJ93b] Aleksandar Milosavljevič and Jerzy Jurka. Discovery by Minimal Length Encoding: a Case Study in Molecular Evolution. *Machine Learning*, 12:69–87, 1993.

[ML84] M.G. Main and R.J. Lorentz. An $o(n \log(n))$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.

[MW84] G.A. Miller and M.N. Wegman. Variations on a theme by Ziv and Lempel. In A. Apostolico and Z. Galil, editors, *Combinatorial algorithms on words.* Springer-Verlag, Berlin, 1984.

[Nel91] Mark Nelson. *The Data Compression Book.* M&T Publishing Inc., 1991.

[OdAAC+92] S.G. Oliver, Q.J.M. Van der Aart, M.L. Agostini-Carbone, M. Aigle, and al. The complete DNA sequence of yeast chromosome III. *Nature*, 357:383–401, 1992.

[ODH95] E. Ollivier, M.O. Delorme, and A. Henaut. DosDNA occurs along yeast chromosomes, regardless of functional significance of the sequence. *Compte Rendu de l'Académie des Sciences*, 318:599–608, 1995.

[RDDD94] É. Rivals, O. Delgrange, M. Dauchet, and J-P. Delahaye. Compression and Sequence Comparison. In A. Apostolico, editor, *DIMACS Workshop on Sequence Comparison*, nov. 1994.

[RDDD95] É. Rivals, O. Delgrange, M. Dauchet, and J-P. Delahaye. A First Step Towards Chromosome Analysis by Compression Algorithms. In N.G. Bourbakis, editor, *First International IEEE Symposium on Intelligence in Neural and Biological Systems*, pages 233–239, Washington DC, USA, may 1995. IEEE Computer Society Press.

[Riv94] Éric Rivals. Compression pour l'analyse de séquences. In Maxime Crochemore, editor, *Actes de la rencontre pour les Recherches de Motifs dans les Séquences, GREG, Marseille fev-94.* Institut Gaspard Monge, Université de Marne La Vallée, 1994.

[RPE81] M. Rodeh, V.R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *Journal Association for Computing Machinery*, 28(1):16–24, January 1981.

[SB92] Maxime Singer and Paul Berg. *Gènes et génomes.* Vigot, 1992.

[Sch94] Jeanette P. Schmidt. All Shortest Paths in Weighted Grid Graphs and its Application to Finding All Approximate Repeats in Strings. 1994.

[Sha48] C.E. Shannon. A Mathematical Theory of Communications. *Bell System Technical Journal*, 27, 1948.

[SK83] D. Sankoff and B. Kruskal. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison.* Addison-Wesley (Reading Massachusset), 1983.

[Sto88] J.A. Storer. *Data Compression: Methods and Theory.* Computer Sciences Press, 1988.

[SW49] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communications.* University of Illinois Press, Urbana IL, 1949.

[Tis87] P. Tischer. A modified Lempel-Ziv-Welch data compression scheme. *Australian Computer Science Communications*, 9(1):262–272, 1987.

[TMD+85] S.W. Thomas, J. McKie, S. Davies, K. Turkowski, J.A. Woods, and J.W. Orost. *Compress (version 4.0) program and documentation*, 1985.

[Ukk92] E. Ukkonen. Constructing suffix trees on-line in linear time. In *Proceedings of IFIP 92*, pages 484–492, 1992.

[Wat89] Michael S. Waterman. *Mathematical Methods for DNA Sequences.* Boca Raton, FL, CRC Press Inc., 1989.

[Wat92] O. Watanabe. *Kolmogorov Complexity and Computational Complexity.* Springer-Verlag, 1992.

[Wei73] P. Weiner. Linear pattern matching algorithms. In *Conf. Record of the 14th Annual Symposium on Swithcing and Automata Theory*, 1973.

[Wel84] T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[WS93] R.D. Wells and R.R. Sinden. *Genome Rearrangement and Stability*, chapter Defined Ordered Sequence DNA, DNA Structure, and DNA-directed Mutation. Genome Analysis. Cold Spring Harbor Laboratory Press, 1993.

[Yoc92] P.H. Yockey. *Information Theory and Molecular Biology.* Cambridge Univ. Press, 1992.

[ZL77] Jacob Ziv and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of Individual Sequence via Variable Length Coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.