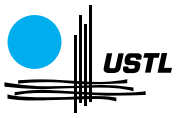
Algorithmes pour la comparaison de génomes et la recherche de signaux cis-régulateurs

Jean-Stéphane Varré




Algorithmes pour la comparaison de génomes et la recherche de signaux cis-régulateurs

Mémoire présenté le 4 décembre 2008

pour l'obtention de

l'Habilitation à Diriger des Recherches

en Sciences mathématiques (spécialité informatique)

par

Jean-Stéphane Varré

Composition du jury

Rapporteurs :	Thierry Lecroq, Professeur, Université Rouen Francis Quétier, Professeur, Université Evry Val d'Essonne Marie-France Sagot, DR INRIA, Projet Helix et UMR CNRS 5558
Examinateurs :	Mireille Régnier, DR INRIA Sophie Tison, Professeur, Université des Sciences et Technologies de Lille
Directeur :	Hélène Touzet, CR1 CNRS HDR, Projet INRIA SEQUOIA et UMR CNRS 8022

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UMR CNRS 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX

À ma tribu,

Merci

Je tiens en premier lieu à remercier mes rapporteurs Thierry Lecroq, Francis Quétier et Marie-France Sagot qui m'ont fait l'honneur d'accepter d'évaluer mon travail. Merci pour le temps consacré à la lecture de ce manuscrit et à l'écriture du rapport. Merci également aux examinateurs Mireille Régnier et Sophie Tison de participer à ce jury malgré un emploi du temps chargé.

Des remerciements spéciaux vont à Hélène Touzet à qui nous devons beaucoup dans l'équipe. Elle a su reprendre les rênes de l'équipe au début des années 2000 et insufler un air de renouveau et de conquête qui a aboutit à l'établissement d'un projet commun. J'ai énormément de respect pour le travail qu'elle a accompli durant ces années.

Des remerciements chaleureux à Max Dauchet avec qui je ne me lasserais jamais de discuter.

Merci aux étudiants que j'ai encadré ou que j'encadre aujourd'hui, qui réalisent un important travail, en particulier les doctorants : Martin Figeac, Aude Liefooghe et Aude Darracq.

Une pensée aux autres membres de l'équipe, des « moins jeunes » Maude, Gregory et Stéphane aux « plus jeunes » Laurent et Mathieu (merci beaucoup Mathieu), ainsi qu'envers tous ceux qui marquent un temps la vie d'équipe : doctorants, stagiaires, précieux ingénieurs, etc. Nous essayons ensemble de construire une vie d'équipe, en plus de la relation de travail qui nous lie.

Merci également à Pascal Touzet pour son enthousiasme débordant et communicatif.

Je ne pourrais terminer ces remerciments sans penser à ceux qui comptent le plus pour moi et qui me permettent de poursuivre mes buts. Ma vie est la leur. Marie-Astrid, Bastien, Arthur, Léo et Tao vous m'accompagnez à chaque instant.

Table des matières

Avant-propos

Ce document retrace les travaux de recherche menés au sein de l'équipe Bioinfo/SEQUOIA du LIFL depuis 2001, année de ma nomination. Il est organisé en deux parties correspondant aux deux thèmes de recherche développés.

La première partie se situe dans le contexte de l'analyse de l'organisation des génomes. Ce travail a fait suite à mes travaux de thèse. Ces derniers avaient pour objectif l'établissement de nouvelles mesures de distance entre séquences d'ADN avec deux buts. Le premier était de disposer d'un outil de mesure capable de s'affranchir d'un alignement entre les séquences. Le second était de pouvoir ainsi réaliser des phylogénies avec des données difficiles, voire impossible à aligner. Le parti pris avait été de construire ces distances en se basant sur des idées issues de la théorie de l'information. Cela m'avait conduit à établir une distance d'édition par copie de blocs de séquences. Les développements de ces travaux invitaient naturellement à se rapprocher des travaux ayant débuté quelques années auparavant sur la comparaison de génomes et les distances par réarrangement de gènes. C'est ainsi que j'entrepris des pistes de recherche sur l'introduction de contraintes dans les calculs de distances de réarrangements. Ce travail se poursuit aujourd'hui dans le cadre d'une collaboration avec le laboratoire de Génétique et Évolution des Populations Végétales.

La seconde partie développe les travaux autour des motifs représentés par des matrices score-position utilisées assez fréquemment en bio-informatique et notamment pour modéliser les sites de fixation de facteurs de transcription. Un préliminaire, nécessaire à toute recherche utilisant ces matrices, est de trouver les occurrences de celles-ci dans une séquence. Nous nous sommes d'abord intéressés au problème de trouver les occurrences d'un ensemble de matrices simultanément. Ce problème faisait du sens parce que bien souvent on va rechercher, pour un ensemble de gènes supposés co-régulés, quels sont les facteurs susceptibles de les réguler. Dans cette démarche, on recherche les occurrences de tous les sites de fixation correspondant à un ensemble plus ou moins grand de facteurs avant d'affiner les résultats. Un tel outil trouve aussi un intérêt dans la recherche de coopération de facteurs. D'autre part, d'un point de vue plus informatique, nous nous sommes intéressés à des méthodes capables de rechercher efficacement une unique matrice. Les méthodes pour la recherche de motifs avec de telles matrices n'avait pas encore été développées alors qu'il existe déjà une littérature pour la recherche de motifs exacts dans des textes pondérés (textes avec une probabilité pour chaque lettre de l'alphabet). Les travaux menés sur ce thème concourent également au développement d'un projet plus large de l'équipe dont le but est la création d'une plate-forme d'annotation des régions non codantes.

L'écriture de ce document a été réalisée avec le souci de donner de l'intuition et éviter, tant que possible, une succession d'énoncés de lemmes pour décrire les méthodes. C'est pourquoi les preuves ont été omises et remplacées parfois par une explication dans le texte. Dans chaque partie la première section introduit les problématiques et les autres sections relatent les contributions.

Chapitre 1

Organisation des génomes

Ce chapitre présente les travaux entrepris autour de l'étude de l'organisation des génomes. Depuis le début de mes travaux en bio-informatique, j'ai toujours été intéressé par les méthodes liées à l'étude de l'organisation des séquences vues comme des segments ré-assemblés (sujet de ma thèse de doctorat). C'était pour moi une suite logique en 2000 d'entreprendre des travaux autour de ce que l'on appelle les réarrangements génomiques, c'est-à-dire l'étude de la plasticité des génomes en terme d'opérations sur les gènes. Les travaux ont débuté ainsi avec l'encadrement du mémoire de DEA de Martin Figeac mi 2001 et se sont poursuivis durant sa thèse de 2001 à 2005 [47]. J'ai également encadré sur le thème des clusters de gènes chez les eucaryotes le stage de DEA de Christophe Colomb en 2005. Le travail sur les réarrangements génomiques s'est poursuivi avec la thèse de Aude Darracq, débutée en 2006, sur l'étude des génomes des mitochondries de plantes. C'est un travail plus appliqué mais dont la confrontation aux données réelles engendrera la production de méthodes plus en adéquation avec la réalité biologique.

1.1 Introduction et motivations

Les réarrangements génomiques désignent l'ensemble des modifications s'opérant sur l'ordre des gènes d'un génome. Ces remaniements sont connus depuis fort longtemps [41] et ont été observés chez tous les types d'organismes, aussi bien dans les génomes nucléaires que les génomes des organelles. L'étude de ces remaniements permet de répondre à plusieurs questions. Tout d'abord, la distance de réarrangements (le nombre de remaniements qui se sont déroulés au cours de l'évolution) fournit une méthode supplémentaire pour reconstruire une phylogénie des espèces [26, 22, 107, 112, 122]. Cela peut s'avérer particulièrement utile pour les cas où la conservation des gènes est très importante comme pour les génomes mitochondriaux. L'obtention d'une histoire des réarrangements, des scénarios évolutifs, donne également de précieuses informations sur la coopération entre gènes, ce qui est important du point de vue fonctionnel. Dans cette optique, il est intéressant de ne pas se limiter à la seule étude des réarrangements mais de regarder également la conservation des groupes de gènes parmi les espèces. Cette conservation de structure est bien connue chez les bactéries avec des groupements de gènes que l'on appelle les opérons. Il s'agit de gènes se trouvant sur le même brin d'ADN et possédant une seule unité de transcription, ils sont donc co-transcrits, le plus souvent parce que nécessaires à une même fonction. Cependant, si on regarde d'autres organismes, on retrouve aussi des traces de structures conservées [118, 27, 117, 109]. Ces blocs de synténie sont aussi cruciaux à étudier que les scénarios car ils fournissent

bien sûr des informations sur la coopération entre gènes [83, 108, 73, 123] mais également le matériel initial pour étudier justement les scénarios. La connaissance des gènes d'un génome, et encore moins les relations d'orthologie[1] de ces gènes avec les gènes d'un autre organisme, ne sont pas des données facilement accessibles. Un moyen d'outrepasser ce manque consiste à observer les parties conservées entre organismes au niveau de la séquence [27]. On n'est plus tout à fait dans l'idée initiale d'analyser l'ordre des gènes mais bien souvent on assimile ces blocs de synténie à des gènes conservés. Si l'ensemble des génomes devient trop grand ou si les génomes proviennent d'espèces trop éloignées, alors la conservation des blocs de synténie à l'ensemble des espèces étudiées diminue, naturellement. Cependant, il peut subsister des traces de ceux-ci : un bloc peut être coupé en deux, ou trois dans une espèce. Imaginons être capables de détecter ces relations, alors on peut formuler l'hypothèse que si de telles structures conservées existent, malgré les réarrangements génomiques, c'est qu'elles ont sans doute été conservées au cours du temps. Posséder des méthodes capables d'allier réarrangements génomiques et structures conservées constitue un axe de développement permettant d'apporter des réponses à l'organisation spatiale des génomes en rapport avec les aspects fonctionnels.

Abstraction mathématique du problème. Dans le cadre de l'étude des réarrangements génomiques, les génomes sont vus comme des suites de gènes (ou blocs de synténie) dont chacun porte un numéro. La distance entre les gènes et leur contenu sont oubliés. On ajoute souvent un signe à ce numéro suivant le brin d'ADN par lequel est porté le gène. Les numéros sont donnés généralement entre 1 et n si n est le nombre de gènes différents.

Si le génome ne possède pas de gène dupliqué, alors le génome est représenté par une *permutation*. La plupart du temps, lorsque l'on étudie deux génomes, les génomes sont renumérotés afin que l'un des deux soit la permutation identité, c'est-à-dire la suite $1, \ldots, n$. On parlera de permutation identité positive pour désigner $+1, \ldots, +n$, et de permutation identité négative pour désigner $-n, \ldots, -1$. On notera le plus souvent dans le document π^s pour désigner une permutation signée (dont chaque gène porte un signe) et π^u pour désigner une permutation non signée. Si un gène est dupliqué alors le même numéro apparaît plusieurs fois. On parlera alors de *séquence* pour représenter un génome.

Le choix de la représentation dépend de la présence ou non de gènes dupliqués et a une incidence directe sur les méthodes mises en œuvre pour le calcul.

Une notion importante dans les permutations est le *point de cassure* [8, 98]. Il existe un point de cassure dans la permutation initiale entre deux gènes si l'adjacence n'est pas retrouvée dans le génome cible. Dans le cas des permutations signées, il faudra que les signes soient inversés si les gènes sont renversés. Un exemple est donné Figure 1.8 ou Figure 1.18.

★
★ ★

Nous commençons par décrire les définitions de regroupement de gènes qui ont été utilisées dans les travaux puis nous donnons un bref historique des méthodes de calcul de scénarios évolutifs. Nous présentons dans les sections suivantes les contributions apportées dans ces deux thèmes. D'une part une méthode de calcul de structures conservées entre génomes procaryotes,

[1] Deux gènes sont dits orthologues s'ils sont hérités d'un ancêtre commun et non issus d'une duplication.

car basée sur les opérons, mais extensible à n'importe quel type de génome dès que l'on dispose d'une relation de conservation spatiale entre couples de gènes orthologues. D'autre part nous présentons le calcul de scénarios évolutifs conservant des structures appelées intervalles communs.

1.1.1 Regroupements de gènes

Intervalles communs. Les intervalles communs désignent un ensemble de gènes consécutifs qui sont retrouvés au signe près et à l'ordre près dans un ensemble de permutations. Des algorithmes de détection de ces structures ont été introduits dans [119] en dehors du contexte des génomes. Le lien avec l'étude de l'ordre des gènes est dû à Heber et Stoye un peu plus tard [58]. Remarquons que la définition est valable sur les permutations signées ou non. Un exemple est donné Figure 1.1. Nous supposerons toujours dans la suite que le génome cible est la permutation identité positive. Cela nous permet de désigner un intervalle commun par les positions de ses bornes dans la permutation courante.

Définition 1.1 ([119]) *Pour deux permutations π et σ de taille n, σ étant la permutation identité. $[x, y]$, $1 \leq x \leq y \leq n$, un intervalle de π, est un intervalle* commun *de π et σ si, et seulement si,* $\max(\pi[x, y]) - \min(\pi[x, y]) = y - x$ *avec $\pi[x, y]$ l'ensemble des éléments de la positions x à la position y.*

Le nombre d'intervalles communs est en $\mathcal{O}(n^2)$. Cependant, certains intervalles sont plus intéressants que d'autres. Chaque gène unique forme un intervalle commun trivial, tout comme la permutation complète. Les intervalles communs chevauchant fournissent quant à eux des propriétés intéressantes. Deux intervalles communs se chevauchent s'ils possèdent un gène au moins en commun. L'analyse des intervalles communs chevauchants montre que si I et J sont deux intervalles communs chevauchant alors $I \cap J$, $I \cup J$, $I \setminus J$ et $J \setminus I$ sont également des intervalles communs. Cette propriété a conduit à la notion d'intervalle commun fort [69]. Un intervalle commun est dit *fort* si il ne chevauche aucun autre. L'ensemble des intervalles communs forts peut être représenté dans une structure arborescente représentant les inclusions de ceux-ci. Cette structure nommée PQ-tree (connue par ailleurs depuis les années soixante dix), est largement utilisée depuis lors pour l'analyse des réarrangements génomiques [69]. Un exemple est donné Figure 1.1. Notons que le lien entre PQ-tree et les permutations avait déjà été évoqué dans [19]. Nous discuterons plus en détails de la structure des PQ-tree comme outil d'analyse des scénarios Section 1.3.3.

Les algorithmes de détection des intervalles communs se sont succédés depuis 2000. L'algorithme de départ proposé par Uno et Yagiura [119] calculait l'ensemble des intervalles communs entre deux permutations en temps $\mathcal{O}(n+K)$ où n est la longueur des permutations et K le nombre d'intervalles communs. Herber et Stoye [59] proposèrent en 2001 une extension à k génomes dont la complexité est $\mathcal{O}(kn + K)$. En 2005, Bergeron et *al.* [15] proposèrent un algorithme en $\mathcal{O}(kn)$ pour calculer l'ensemble des intervalles communs forts à k génomes.

Bien sûr, il paraît logique de vouloir étendre la notion d'intervalles communs à des génomes avec duplication et/ou autoriser des erreurs au sein d'un même intervalle : insertion d'autres gènes, perte de l'un d'entre eux.

Dans [44], Eres et *al.* ont introduit la notion de π-pattern qui représente un ensemble de gènes qui apparaît au moins k fois dans une séquence. Il existe une occurrence à une posi-

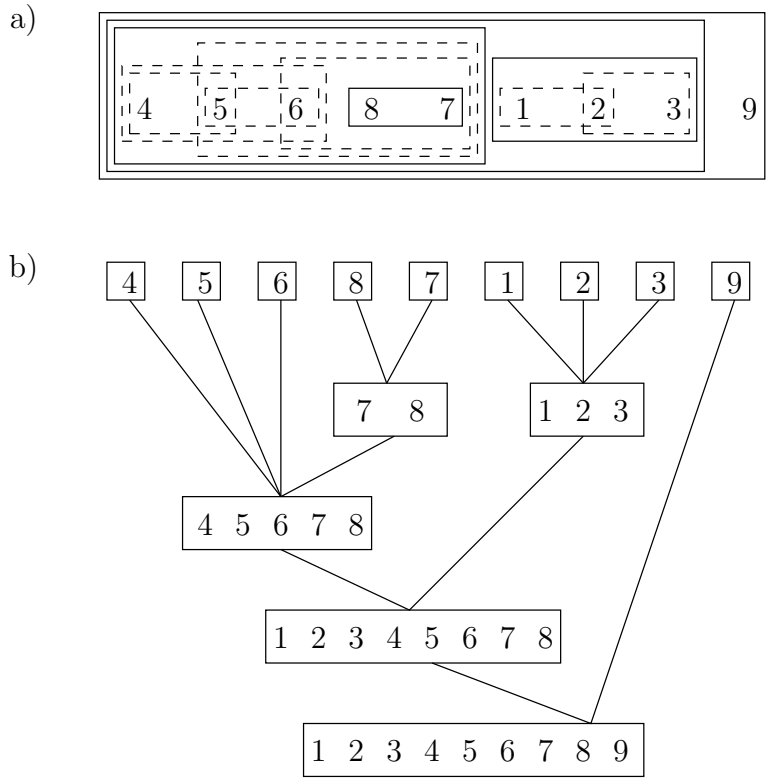


FIG. 1.1 – Exemple d'intervalles communs.

Représentation des intervalles communs entre la permutation $+4\ +5\ +6\ +8\ +7\ +1\ +2\ +3\ +9$ et la permutation identité. a) chaque boîte représente un intervalle commun. $[1,3]$ désigne les éléments $\{4,5,6\}$ dans π qui sont côte-à-côte dans la permutation identité, $[1,3]$ est bien un intervalle commun. Les intervalles communs forts correspondent aux boîtes qui ne se chevauchent pas (en trait plein). b) le PQ-tree représentant les intervalles communs forts.

tion d'un génome si les gènes de l'ensemble sont retrouvés de manière contiguë et chaque gène peut apparaître plusieurs fois. Cela permet donc de prendre en compte les gènes paralogues[2]. L'énumération des π-patterns montre une certaine redondance puisque si, par exemple, trois gènes se suivent à deux positions alors on extraira à la fois le groupe des deux premiers gènes, des deux derniers gènes, de l'ensemble des trois gènes. Sur ce problème un lien a été établi dans [69] avec les PQ-tree. Schmidt et *al.* [101] ont également proposé de prendre en compte les paralogues en définissant la notion de CS-facteur qui est extrêmement proche des π-patterns. Récemment, l'algorithme de calcul des CS-facteurs a été amélioré dans [39].

Dans [33] des extensions des intervalles communs sont proposés autorisant les erreurs où on utilise une distance pour évaluer la similarité entre intervalles communs, contrairement à d'autres approches plus orientées structure telles que les Gene Teams que nous allons décrire ensuite. Pour conclure sur les extensions des intervalles communs, Amir et *al.* [5] ont proposé d'introduire des erreurs dans les intervalles communs sur des séquences en utilisant la différence symétrique comme mesure de comparaison entre les ensembles de caractères en s'appuyant sur les CS-facteurs.

On notera également que Blin et *al.* [23] ont utilisé les intervalles communs pour rechercher les relations d'orthologie entre gènes.

Gene teams. Cette structure de gènes conservées est une extension des intervalles communs proposée par [75] en autorisant les erreurs. L'inconvénient des intervalles communs est qu'il suffit qu'il y ait un seul gène différent pour ne pas identifier l'intervalle commun. L'idée est donc de permettre quelques erreurs dans l'intervalle en terme d'insertion de gènes. Ce nombre d'erreurs n'est pas borné mais c'est le nombre d'erreurs successives qui est borné. Suite aux travaux différents autour de ce type de structure, de tels clusters de gènes sont dénommés des clusters avec saut maximum (max-gap clusters).

La représentation des permutations est légèrement modifiée pour autoriser l'ajout d'un gène spécial noté ★ qui ne sera l'orthologue d'aucun autre. Les clusters extraits sont fonction d'un paramètre d'erreur maximum δ correspondant à la distance entre deux gènes (la différence de positions). Un exemple est donné Figure 1.2.

$\pi_1 = 1 \quad 2 \quad ★ \quad 3 \quad ★ \quad ★ \quad 4 \quad 5 \quad ★ \quad ★ \quad 6$

$\pi_2 = 3 \quad 4 \quad 1 \quad 5 \quad 2 \quad ★ \quad ★ \quad ★ \quad 6$

δ	Gene Teams
1	$\emptyset$
2	$\{1,2,3\}, \{4,5\}$
3	$\{1,2,3,4,5\}$
4	$\{1,2,3,4,5,6\}$

FIG. 1.2 – Exemple de Gene Teams pour deux permutations.
On trouve l'ensemble $\{1,2,3\}$ pour $\delta = 2$ car la différence de positions est de 2 entre 2 et 3 dans π_1 et la différence de positions est de 2 entre 3 et 1 et 1 et 2 dans π_2. On ne trouve pas l'ensemble $\{1,2,3,4,5,6\}$ pour $\delta = 3$ car la différence de positions entre 2 et 6 dans π_2 vaut 4.

L'algorithme le plus efficace est en $\mathcal{O}(Cn \log_2 n)$ avec C le nombre de chromosomes par

[2]Deux gènes sont dit paralogues si l'un d'eux est le résultat d'une duplication de l'autre dans son histoire évolutive.

génome [9]. Un inconvénient est que le nombre de Gene Teams peut être exponentiel [86]. Une extension avec les paralogues a été proposée par He et Goldwasser [57].

Dans [66], les auteurs proposent de relâcher les contraintes de distance sur les Gene Teams afin de permettre la détection de structure « éparpillées » dans les génomes. Cette façon de faire est très proche de ce qui a été proposé dans la notion de über-operon décrite ci-après.

Connected Gene Neighborhood et Über-operons. En 2000, Lathe et *al.* [70] ont introduit un concept de cluster de gènes, construit à partir des opérons : les über-operons (sur-opérons). Rappellons qu'un opéron est une suite de gènes consécutifs sur un même brin d'ADN. Bien souvent un critère de distance minimale entre deux gènes adjacents suffit à détecter bon nombre d'opérons [83, 93, 76, 62]. Si l'on peut observer des opérons parfaitement conservés entre plusieurs espèces proches du point de vue évolutif, le signal est vite dégradé dès que l'on passe à un plus grand nombre d'espèces et/ou un groupe dont la divergence s'est produite il y a longtemps (voir Figure 1.3). Cela montre aussi l'intérêt pour une méthode de détection de clusters d'accepter des insertions/délétions de gènes. Si les opérons ne sont plus visibles, c'est qu'ils ont

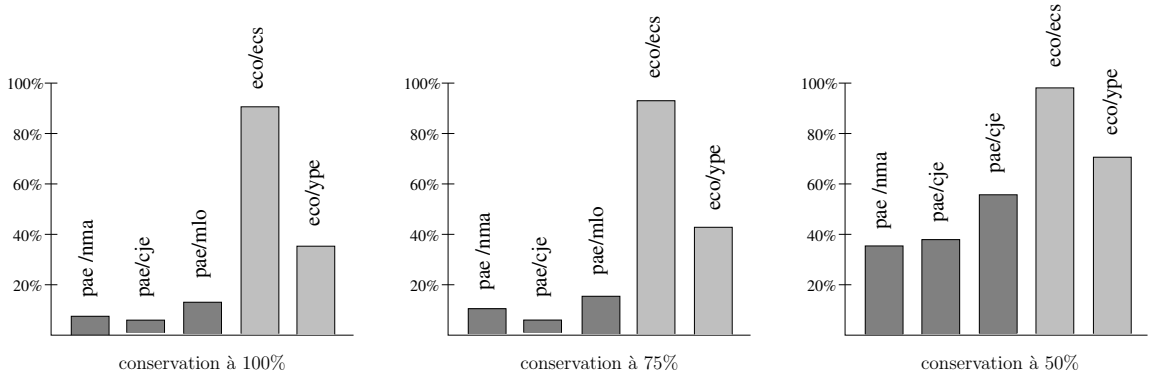


FIG. 1.3 – Étude de la conservation des opérons entre espèces évolutivement proches et espèces plus éloignées.
Comparaison des opérons de *Escherichia coli K12*' (eco) avec ceux de *Escherichia coli O157 :H7* (ecs) et de *Yersinia pestis* (ype), et des opérons de *Pseudomonas aeruginosa* (pae) avec ceux de *Neisseria meningitidis Z2491* (nma), de *Campylobacter jejuni* (cje) et de *Mesorhizobium loti* (mlo). Dans le premier groupe, les espèces sont plus proches entre-elles que dans le second groupe. Est représenté le pourcentage d'opérons considérés comme communs lorsqu'ils sont identiques, lorsqu'ils partagent 75% de gènes, lorsqu'ils partagent 50% de gènes. La méthode proposée par Overbeek et *al.* [83] a été utilisée pour prédire les opérons.

été remaniés, probablement suite à des réarrangements. On peut donc se demander si il existe des traces de ces réarrangements. L'idée proposée par Lathe et *al.* est d'observer des ensembles de gènes qui restent dans le même voisinage, comme ceux vus jusqu'à maintenant, mais répartis sur le génome. On peut voir cette structure comme un intervalle commun réparti : si on supprime tous les gènes hormis ceux du über-operon, on a un intervalle commun. Une illustration d'une telle structure est donnée Figure 1.4. Appliqué à un exemple biologique concernant les gènes relatifs au mécanisme du flagelle, on obtient des répartitions très différentes suivant l'espèce considérée comme cela peut être remarqué Figure 1.5. Cela montre la pertinence de l'existence

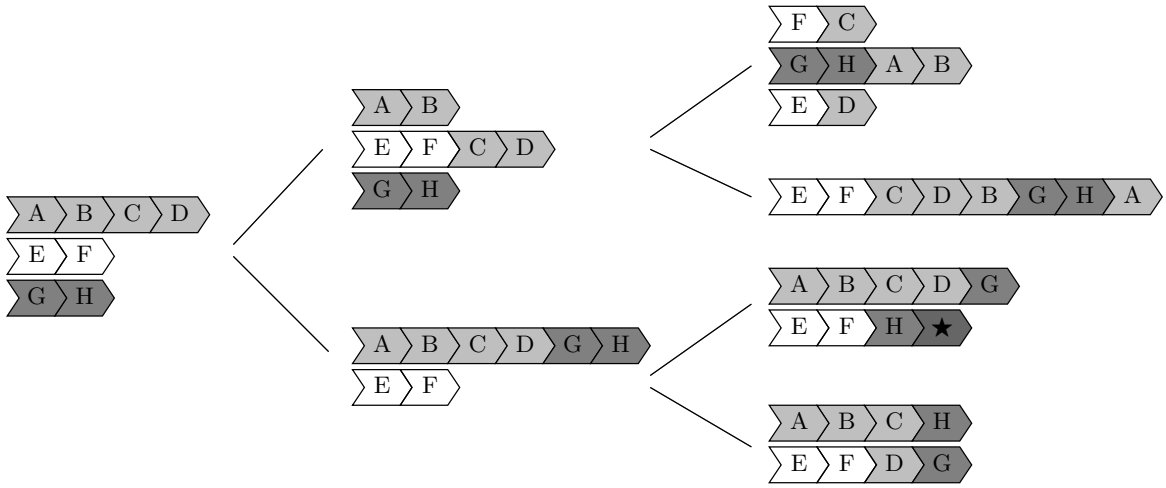


FIG. 1.4 – Illustration du concept de über-operon, tiré de [70]
Les quatre espèces possèdent les gènes A,B,C,D,E,F,G,H impliqués dans une même fonction. Les gènes sont réparties en différents groupes suivant les espèces. On peut même envisager une reconstruction phylogénétique permettant d'expliquer la dynamique de ces gènes. Tiré de [70].

de telles structures et l'intérêt qu'il y a à disposer d'outils les détectant. Le procédé itératif de calcul proposé par Lathe et *al.* est le suivant : sélectionner un gène, par exemple un gène lié à un mécanisme particulier ; pour ce gène, déterminer, dans toutes les espèces comparées, tous les autres gènes appartenant aux mêmes opérons que celui-ci. Les gènes détectés forment un début d'über-operon. On itère le processus avec comme gènes de départ les gènes nouvellement ajoutés à l'über-operon. On s'arrête quand tous les gènes de l'über-operon ont été étudiés. Rogozin et *al.* [91] ont proposé une méthode de détection de cette structure en s'appuyant sur une modélisation en graphe des relations entre gènes. L'algorithme procède en trois étapes. Les relations entre les gènes sont représentées dans un graphe dont la construction représente la première étape. Les sommets sont les familles de gènes. Une arête existe entre deux sommets si les deux gènes sont sur le même brin, séparés par au plus deux autres gènes et ceci dans au moins trois génomes. La seconde étape consiste à recherche les chemins maximaux dans le graphe tels que n'importe quelle suite de trois sommets dans un chemin implique l'existence de la suite de gènes correspondant dans au moins un génome (en acceptant toujours un écart d'au plus deux gènes entre les génomes). La troisième étape consiste à unir les chemins qui possèdent au moins trois gènes en commun. En 2006, Che et *al.* [34] proposèrent une méthode d'identification des über-operons ne nécessitant pas d'avoir les relations d'orthologie entre les gènes. Il est fait usage de BLAST pour déterminer des relations entre les gènes (les relations de n gènes à n gènes sont autorisées). Les relations d'orthologie sont déduites grâce à la donnée des opérons : deux couples de gènes similaires dans des opérons dans deux espèces ont plus de chance de former un couple d'orthologues.

FIG. 1.5 – Les gènes impliqués dans le mécanisme du flagelle.
Gènes repérés sur les génomes de *Bacillus subtilis*, *Clostridium acetobutylicum*, *Escherichia coli K12* et *Yersinia pestis*. Les unités blanches montrent des gènes qui ne sont pas connus pour être impliqués dans la fonction alors que les unités noires ou grises indiquent une implication directe, respectivement indirecte, dans le mécanisme (d'après KEGG [82]).

1.1.2 Scénarios évolutifs

Pour préciser ce que sont les scénarios de réarrangement, nous commençons par une description des opérations de réarrangement les plus courantes puis nous retraçons un court historique des méthodes.

Opérations de réarrangements et scénario évolutif. Une *opération de réarrangement* consiste à modifier une portion d'un génome. Cette opération peut affecter soit un unique chromosome, soit plusieurs chromosomes. Dans le cas d'un unique chromosome, les opérations rencontrées sont l'inversion, la transposition, la duplication, la perte, le gain d'une portion. Dans le cas de plusieurs chromosomes, il existe la translocation, la fission et la fusion. D'autres opérations telles que l'échange de blocs [35] ou encore le *double-cut-and-join* [126] ont également été introduites. Les opérations les plus courantes sont illustrées Figure 1.6.

L'étude des réarrangements génomiques consiste à identifier les scénarios qui ont permis de passer d'un génome à un autre au cours de l'évolution. On parlera de *scénario de réarrangement* ou de *scénario évolutif*. Le but est de comprendre comment les gènes s'organisent spatialement dans les génomes au cours de l'évolution. On peut aussi caractériser les scénarios évolutifs par leur longueur. On comprend alors l'idée intuitive que si un scénario est court entre deux espèces c'est que celles-ci ont sans doute divergé depuis peu. On cherche dans ce cadre à calculer le scénario le plus parcimonieux, c'est-à-dire le plus court. Les scénarios peuvent ainsi être un outil de reconstruction phylogénétique entre espèces. Un exemple de scénarios et de phylogénie est donné Figure 1.7.

Le problème le plus étudié est sans aucun doute celui du nombre minimum d'inversions entre deux permutations. Il a été étudié pour le cas signé et non signé. Si le calcul pour le cas non

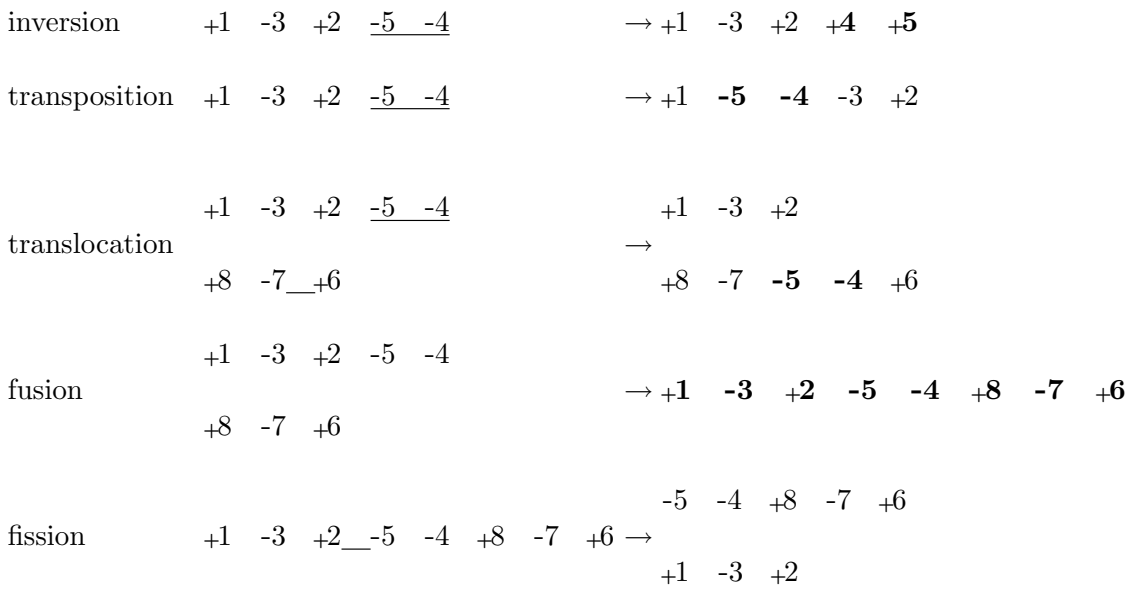


FIG. 1.6 – Illustration de quelques opérations de réarrangement.
Figurent en souligné les gènes concernés par l'opération (à gauche) et en gras l'état des gènes après l'opération (à gauche).

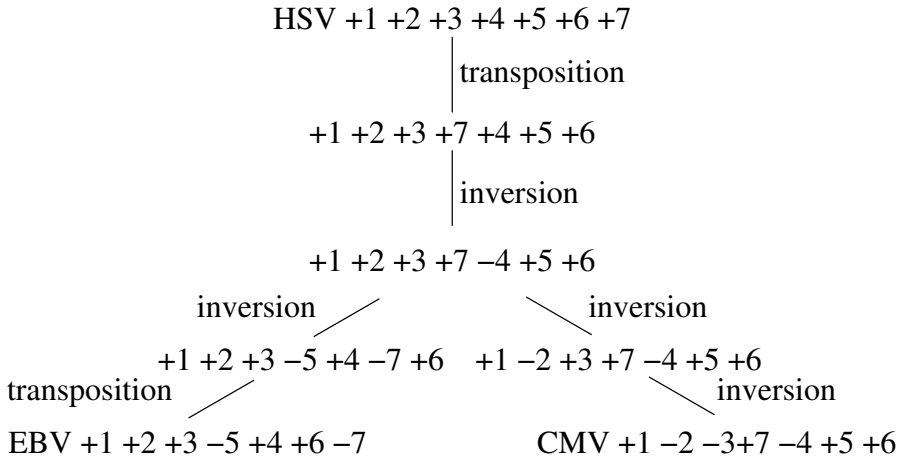


FIG. 1.7 – Des scénarios expliquant l'évolution de trois virus de l'herpès. (tiré de [54]).
La permutation centrale représente l'ancêtre hypothétique des trois espèces. Pour chacun un scénario a été trouvé utilisant un modèle d'évolution où inversions et transpositions peuvent survenir.

signé a été montré NP-dur [30], l'étude du problème dans le cas signé a donné lieu à une abondante littérature. On passa par différentes méthodes pour le calcul, différentes modélisations, améliorant à chaque fois la complexité. On sait aujourd'hui calculer la distance la plus courte entre deux génomes signés en un temps linéaire [6].

Un problème connexe aux scénarios de réarrangements concerne la recherche d'un médian entre trois génomes. Il s'agit de trouver une permutation qui minimise la somme des distances entre celle-ci et les trois génomes. C'est un aspect fondamental pour reconstruire une phylogénie. La littérature est également abondante sur ce sujet [98, 77, 78, 121].

Structures du graphe des points de cassure. La structure de données sur laquelle des objets combinatoires sont calculés est le graphe des points de cassure [8], encore appelé graphe de Hannehalli et Pevzner ou graphe de réalité et désir car reflétant à la fois les relations de voisinage entres les gènes du génome de départ et aussi entre les gènes du génome but. La mise en place de ce graphe a permis l'identification de structures dans celui-ci aboutissant à une formule donnant le nombre exact du minimum d'inversions $d(\pi^s)$ sur une permutation signée π^s en fonction de ces structures [55, 105] :

$$d(\pi^s) = n + 1 - c(\pi^s) + h(\pi^s) + f(\pi^s)$$

avec π^s une permutation signée tel qu'il existe un point de cassure entre chaque gène et où $c(\pi^s)$ est le nombre de cycles dans le graphe et h et f des fonctions sur d'autres structures que nous ne détaillons pas.

Nous décrivons brièvement ce qu'est le graphe des points de cassure pour les permutations signées afin de donner les clés à la compréhension de la suite. Le graphe des points de cassure est construit à partir de la permutation de départ où chaque gène est dédoublé en deux bornes, positive et négative, puis on ajoute aux extrémités les valeurs 0 et $n+1$. Un exemple est donné Figure 1.8. En reliant les bornes avec des arcs noirs (réalité) en suivant l'ordre de la permutation puis avec des arcs gris (désir) en suivant l'ordre de la permutation identité, on fait apparaître des cycles. Les cycles sont soit *convergents* si tous les arcs vont dans le même sens, *divergents* sinon. Un ensemble de cycles entrelacés et convergents est appelé une *mauvaise composante* en opposition aux *bonnes composantes*. Dans les bonnes composantes, on peut choisir facilement une inversion triante (qui va mener à la permutation but de manière parcimonieuse).

Il existe de nombreuses permutations pour lesquelles il n'existe pas dans le graphe de structures qui rendent complexe le choix d'une inversion triante. Une présentation élégante du problème par Bergeron et *al.* [17, 18] a permis de simplifier la définition d'inversion triante et notamment de faire le lien entre les cycles du graphe des points de cassure et les intervalles communs. Tannier et *al.* [114] ont proposé récemment un algorithme fournissant un scénario parcimonieux en $\mathcal{O}(n^{3/2}\sqrt{\log n})$. Siepel [106] a quant à lui proposé un algorithme capable d'énumérer l'ensemble des scénarios parcimonieux.

Algorithme de tri de permutations non signées. Le problème du traitement de permutations non signées provient du fait que l'on ne connaît pas nécessairement le brin d'ADN sur lequel est porté le gène. Bien qu'il soit maintenant (très) facile (d'un point de vue complexité algorithmique) de calculer la distance d'inversions entre permutations signées ou de trouver un scénario, les mêmes problèmes sur des permutations non signés sont difficiles. Nous avons choisi de présenter l'algorithme de Hannehalli et Pevzner [56] pour trier une permutation non signée

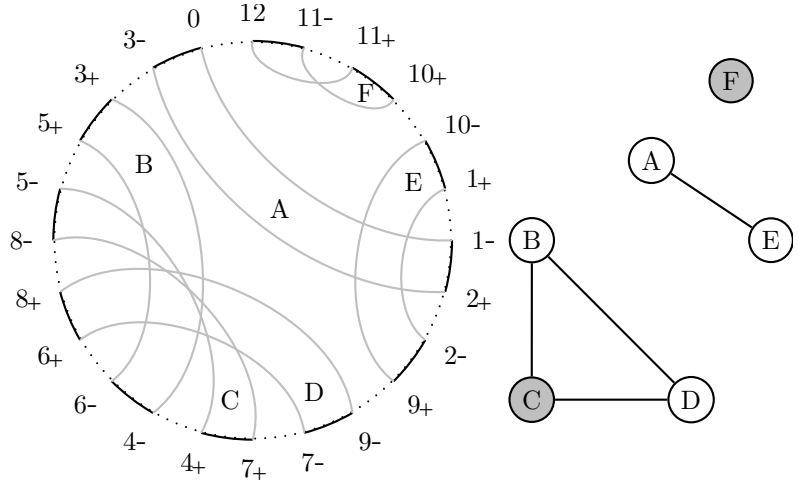


FIG. 1.8 – Graphe des points de cassure et graphe des composantes.
À gauche, le graphe des points de cassure pour la permutation +3 -5 +8 -6 +4 -7 +9 +2 +1 +10 -11 contient 6 cycles. Il existe des points de cassure entre tous les gènes. F (10+,11-,12,11+) est un cycle divergent, en inversant l'élément 11 on crée deux cycles (12,11+) et (11-,10+), on a ainsi augmenté le nombre de cycles donc diminué la distance, cette inversion est triante. À droite le graphe des composantes, construit à partir des entrelacements des cycles. Les cycles grisés correspondent aux cycles divergents (C et F). La composante (A,E) est une mauvaise composante car ne contenant que des cycles convergents. Les autres sont des bonnes composantes. Tiré de [56].

car nous utiliserons leurs résultats pour le tri avec intervalles communs, en étendant les lemmes aux permutations partiellement signées.

L'algorithme proposé par Hannenhalli et Pevzner consiste grossièrement à calculer la distance par inversions pour toutes les attributions de signes de la permutation non signée (attribution d'un signe à chaque gène) puis à sélectionner la permutation signée de plus petite distance. Cependant, les auteurs remarquèrent qu'il était possible de ne pas envisager les 2^n attributions de signes.

Une permutation signée π^s est un *spin* d'une permutation non signée π^u, telle que pour tout i, $1 \leq i \leq n$, $\pi^s(i) = +\pi^u(i)$ ou $\pi^s(i) = -\pi^u(i)$ (avec $\pi(i)$ le i-ème élément de la permutation π). Pour une permutation non signée arbitraire π^u d'ordre n nous définissons Π^u l'ensemble des 2^n spins de π^u. Un spin π^s de π^u est dit optimal, s'il donne la distance minimale, c'est-à-dire si $d(\pi^u) = d(\pi^s)$. L'algorithme décrit plus haut fonctionne grâce au lemme suivant :

Lemme 1.1 *[56] Pour une permutation non signée π^u,*

$$d(\pi^u) = \min_{\pi^s \in \Pi^u} d(\pi^s)$$

Hannenhalli et Pevzner ont mis en avant des structures sur les permutations permettant de décider à l'avance du signe de certains éléments.

Un *bloc* d'une permutation non signée π^u est un intervalle $[x, y]$ de π^u qui ne contient aucun

point de cassure. Un *strip* d'une permutation non signée π^u est un bloc maximal, c'est-à-dire un bloc $[x, y]$ tel qu'il existe un point de cassure entre $\pi^u(x-1)$ et $\pi^u(x)$ et entre $\pi^u(y)$ et $\pi^u(y+1)$.

Un bloc ou un strip $[x, y]$ d'une permutation signée ou non est dit *croissant* si $|\pi(x)| < |\pi(y)|$, sinon il est dit *décroissant*. Un strip croissant (respectivement décroissant) d'une permutation signée est dit *canoniquement* signé si tous ses éléments sont positifs (respectivement négatifs). Un strip canoniquement signé n'a aucun point de cassure et correspond donc à un intervalle trié, soit positivement, soit négativement. Un strip décroissant (respectivement croissant) avec tous ses éléments signés positifs (respectivement négatifs) est dit *anti-canoniquement* signé.

Le lemme suivant permet de montrer qu'il existe un scénario parcimonieux où les strips longs ne sont jamais coupés par une inversion. Autrement dit, une fois qu'il sont formés, ils constitueront une unité indivisible.

Lemme 1.2 *[56] Pour une permutation non signée π^u, il existe π^s un spin optimal tel que tous ses strips de taille au moins trois sont canoniquement signés.*

Pour les strips de taille deux, le traitement est plus délicat. Le lemme suivant permet de savoir que les strips doivent être triés tels que tous les éléments sont signés soit positivement, soit négativement, ce qui complète le lemme précédent.

Lemme 1.3 *[56] Pour une permutation non signée π^u, il existe π^s un spin optimal de π^u tel que tous les strips de longueur deux sont soit signés canoniquement, soit signés anti-canoniquement.*

Dans le graphe des points de cassure, on a distingué les bonnes composantes des mauvaises composantes.

Lemme 1.4 *[56] Pour une permutation non signée π^u, il existe π^s un spin optimal de π^u tel que, une bonne composante du graphe des points de cassure de π^u ne contient aucun strip de longueur deux anti-canoniquement signé, et une mauvaise composante en contient au plus un.*

Un *super spin* de π^u est un spin de π^u dont tous les strips sont canoniquement signés ; on a alors sélectionné dans chaque mauvaise composante un strip de longueur deux, s'il en existe, et on l'a transformé en anti strip.

Théorème 1.1 *[56] Tout super spin d'une permutation non signée composé uniquement de strips est un spin optimal.*

Pour calculer la distance par inversions entre deux permutations non signées d'ordre n, Hannenhalli et Pevzner ont proposé un algorithme en $\mathcal{O}(n2^k + n)$, k étant le nombre d'éléments des permutations n'étant pas dans des strips. Il faut tester les 2^k spins différents pour ces k éléments puis en appliquant les règles de construction d'un super spin, qui peuvent être effectuées en temps linéaire, on obtient un spin optimal.

Extension et contraintes sur les opérations dans les scénarios. Si beaucoup de travaux furent ainsi réalisés pour obtenir des résultats de complexité ou des algorithmes performants permettant de traiter toutes les opérations et/ou toutes les combinaisons d'opérations, la vue

extrêmement simple de la modélisation des génomes reste le point faible qui empêche le passage des algorithmes aux outils utiles à la communauté biologique. Le passage à l'étude de génomes non simplifiés demandait et demande encore plus de travaux sur les méthodes.

La première extension sur laquelle des résultats ont été obtenus concerne la prise en compte d'un contenu différent pour chaque génome mais sans duplication. Plus précisément, comment trouver un scénario entre deux génomes pour lesquels certains gènes sont absents chez l'un ou l'autre des génomes [43]. Un autre problème, d'un intérêt majeur, et sur lequel beaucoup de travail a été réalisé concerne la prise en compte des gènes dupliqués. Deux approches sont actuellement utilisées. La première, appelée *exemplarisation* consiste à ne garder qu'une seule copie de chaque gène, puis à appliquer une méthode classique de réarrangement. La seconde, appelée *couplage maximal*, consiste à trouver, un peu comme dans un alignement, un maximum d'associations de gènes du premier génome avec ceux du second. On peut ainsi différencier les gènes paralogues. Il est possible de choisir, pour l'une ou l'autre des méthodes, des associations optimisant certaines contraintes [96, 25, 113, 24].

Malgré le fait que l'on ne s'intéresse qu'aux scénarios parcimonieux, le nombre de ceux-ci pour deux génomes donnés est encore très grand. Par exemple, il existe deux cents scénarios parcimonieux pour trier la permutation -4 +1 -3 +6 -7 -5 +2. Pour des permutations plus grande (une trentaine de gènes) on aboutit à plusieurs centaines de milliards de scénarios parcimonieux. Un résultat dû à Bergeron et *al.* [16] montre qu'une inversion prise au hasard a quasiment toutes les chances d'être triante (la probabilité qu'elle ne le soit pas est en $\mathcal{O}(1/n^2)$, avec n le nombre d'éléments de la permutation).

Si autant de scénarios sont possibles, il est légitime de se poser la question de la pertinence biologique de tel ou tel scénario. C'est partant de ce constat que nous avons essayé, comme d'autres, d'introduire plus de réalisme dans le calcul des scénarios en proposant de réduire le nombre de scénarios parcimonieux. Cela peut être réalisé en attribuant des poids aux inversions [3, 88]. Mais une autre façon de faire est de n'autoriser que certaines inversions parmi celles possibles (ou plus largement certaines des opérations autorisées). Par exemple, partant de l'analyse de la taille des inversions chez les génomes bactériens, il est possible d'envisager des scénarios où vont être préférées des inversions courtes et des inversions longues centrées sur l'origine de réplication [72]. En 2003 Bergeron et Stoye [19] constatèrent qu'une mesure de la distance phylogénétique entre espèces basé sur le décompte du nombre d'intervalles conservés est un meilleur estimateur que le nombre de points de cassure pour des génomes mitochondriaux. Fort de ce constat, Bergeron et *al.* [14] proposèrent en 2004 une méthode de reconstruction phylogénétique utilisant les intervalles conservés. Pour notre part, nous avons proposé d'utiliser les intervalles communs pour contraindre les inversions autorisées dans un scénario. Ces travaux sont exposés Section 1.3.

1.2 Étude des clusters de gènes

Nous avons proposé un algorithme de détection d'une super-structure conservée, largement inspirée de celle d'über-operon. Nous voulions que la méthode puisse prendre en compte des gènes dupliqués, autorise la suppression ou l'insertion de quelques gènes, et être capable d'extraire une hiérarchie de structure entre les gènes. Le plus haut niveau étant l'ensemble de tous les gènes. Cela reprend quelques propriétés énumérées dans [61] sur les qualités que doit fournir une méthode de détection de clusters de gènes.

1.2.1 Un nouvel algorithme pour détecter des über-operons.

Comme dans [91], nous nous appuyons naturellement sur un graphe représentant les relations entre les gènes à partir de la donnée des opérons. Nous autorisons comme dans le cas des Gene Teams d'avoir des gènes dont aucun orthologue n'est connu pour les autres espèces étudiées (toujours représentés par ★). On définit les familles de gènes comme l'ensemble des gènes possédant au moins un orthologue auquel on ajoute le gène spécial ★. On définit maintenant le graphe non orienté et pondéré $G = (S, A, \omega)$ où l'ensemble S des sommets est formé par les familles de gènes et l'ensemble A des arêtes est tel qu'il existe une arête de poids $\omega(u, v)$ entre deux sommets u et v si les gènes correspondants sont trouvés $\omega(u, v)$ fois dans le même opéron. Une illustration est donnée Figure 1.9.

À partir de ce graphe nous allons extraire des composantes connexes avec un niveau de connexion α. L'ensemble des gènes de ces structures représenterons les super-structures conservées. Par exemple, pour $\alpha = 2$ et les trois génomes de la Figure 1.9, on trouve trois super-structures conservées $\{1, 2, 3, 10, 11, 13\}$, $\{14, 15\}$ et $\{8, 9\}$.

Définition 1.2 *Un α-chemin dans un graphe non orienté pondéré $G = (S, A, \omega)$ est une suite de nœuds $s_1, ..., s_n$ de S tel que pour tout $1 \leq i < n$, $(s_i, s_{i+1}) \in A$ et $\omega(s_i, s_{i+1}) \geq \alpha$.*

Définition 1.3 *Une composante α-connectée d'un graphe non orienté pondéré $G = (S, A, \omega)$ est un sous graphe $G' = (S', A')$ de G tel que pour tout $u \in S'$ et $v \in S'$ il existe un α-chemin dans G' passant par u et v.*

Définition 1.4 *Une composante α-isolée d'un graphe non orienté pondéré $G = (S, A, \omega)$ est un sous graphe $G' = (S', A')$ de G tel que pour tout $u \in S'$ et pour tout $v \in S \setminus S'$, il n'existe pas de α-chemin passant par u et v.*

Définition 1.5 *Une composante α-connexe (maximale) d'un graphe non orienté pondéré $G = (S, A, \omega)$ est une composante α-connectée et α-isolée.*

Le graphe des opérons se construit en parcourant les génomes pour déterminer leurs opérons et pour chaque opéron o de taille $|o|$, en ajoutant les (au plus) $\frac{|o| \times (|o|-1)}{2}$ arcs. Le nombre d'arcs ajouté au total est dans le pire des cas en $\mathcal{O}(n \times m^2)$ pour n permutations, chacune de taille au plus m gènes. Finalement, l'algorithme détermine les composantes connexes du graphe des opérons en ne prenant en compte que les arcs de poids supérieur au seuil α fixé. Les composantes connexes d'un graphe $G = (S, A)$ sont recherchées en parcourant une et une seule fois les arcs du graphe G. La complexité de cette étape est dans le pire des cas en $\mathcal{O}(|A|)$. Par conséquent, la complexité du cœur de l'algorithme de recherche des über-operons est, dans le pire des cas, en $\mathcal{O}(n \times m^2)$.

1.2.2 Représentation hiérarchique

Il est bien sûr possible à partir du graphe d'extraire l'ensemble des super-structures pour toutes les valeurs de α possibles. On comprend bien qu'il existe une relation d'inclusion entre les composantes α-connexe pour des α de plus en plus petits. On a ainsi adopté une représentation hiérarchique de l'ensemble des super-structures. Un exemple est donné Figure 1.13.

$\pi^1 = \underline{1\ \ 2\ \ 3}\ \ \underline{4\ \ 5}\ \ \underline{6\ \ \bigstar}\ \ \underline{8\ \ 9}\ \ \underline{10\ \ 11\ \ \bigstar\ \ 13}\ \ \underline{14\ \ 15}$
$\pi^2 = \underline{10\ \ 2\ \ \bigstar\ \ 1\ \ 11}\ \ \underline{9\ \ 8}\ \ \underline{14\ \ \bigstar}\ \ \underline{11\ \ 3\ \ 13}\ \ \underline{4\ \ 6}$
$\pi^3 = \underline{5\ \ 15}\ \ \underline{8\ \ 9\ \ \bigstar\ \ 14}\ \ \underline{1\ \ 3}\ \ \underline{4\ \ 15}\ \ \underline{14\ \ \bigstar\ \ 15\ \ 13\ \ 11\ \ 2}$

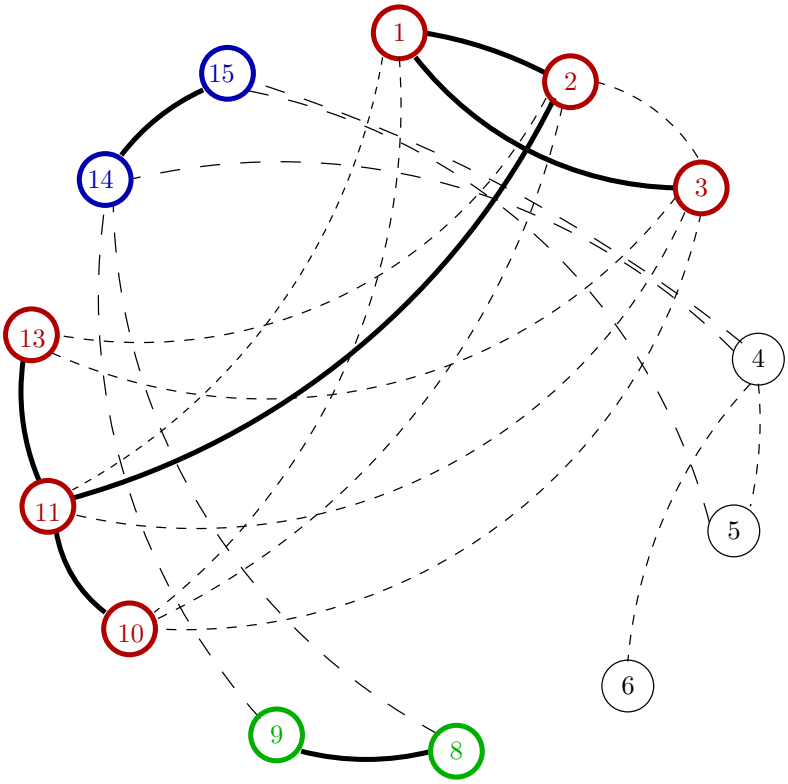


FIG. 1.9 – Le graphe permettant de détecter les super-structures.
Trois génomes, leurs opérons (soulignés) et le graphe associé. Les arcs en pointillés sont les arcs de poids strictement inférieur à deux, par opposition aux arcs en gras de poids supérieur ou égal à deux. L'arc $(1, 2)$ est de poids deux car les gènes 1 et 2 sont deux fois présents dans un même opéron, une fois dans π^1 et une autre fois dans π^2. Bien que les gènes 3 et 4 soient adjacents dans π^1, il n'y a pas d'arc $(3, 4)$ dans le graphe des opérons car les gènes 3 et 4 ne font jamais partie d'un même opéron. Pour $\alpha = 2$, les composantes α-connexes sont représentées par les nœuds reliés par des arcs en gras. On obtient trois composantes α-connexes : $\{1, 2, 3, 10, 11, 13\}$, $\{8, 9\}$ et $\{14, 15\}$. Les composantes α-connexes correspondent à la définition d'über-operon donnée par Lathe et *al.* : une union d'opérons.

Score d'un über-operon. Un über-operon pour une valeur de α fixée, est donc une composante connexe du graphe des opérons où l'on a supprimé les arcs de poids inférieur à α. Par conséquent, le score que l'on peut associer à un über-operon est aussi le score d'une composante α-connexe. Il semble naturel de vouloir qu'un über-operon ait un score d'autant plus élevé que ses gènes sont souvent dans les mêmes opérons. De plus, un über-operon est d'autant plus conservé, donc doit avoir un score d'autant plus élevé, qu'il y a peu de gènes insérés. Pour résumer, plus la composante α-connexe est dense, plus le score doit être élevé, et plus la composante α-connexe est isolée du reste du graphe (elle n'est pas complètement isolée à cause du paramètre α) plus le score doit être élevé.

Étant donné une composante α-connexe $G' = (S', A')$ d'un graphe $G = (S, A, \omega)$, nous utilisons les notions de densité relative (notée $\delta_r(G', G)$) et de densité locale (notée $\delta_l(G', G)$) pour mesurer la force des composantes connexes et en déduire un score

$$f(G', G) = \delta_r(G', G) \times \delta_l(G', G)$$

afin d'avoir une mesure capable d'appréhender les notions de densité et d'isolement (adapté de [120]).

À partir du degré d'un nœud, que nous définissons comme la somme des poids de tous les arcs passant par ce nœud :

$$\deg(u, G', G) = \sum_{\{v \in S' | (u,v) \in A'\}} \omega(u, v),$$

la densité relative s'exprime comme :

$$\delta_r(G', G) = \frac{\deg_{in}(G', G)}{\deg_{in}(G', G) + \deg_{out}(G', G)}$$

et la densité locale comme :

$$\delta_l(G', G) = \frac{\deg_{in}(G', G)}{\sum_{u \in S', v \in S'} \max(\deg(u, G', G), \deg(v, G', G))}$$

avec $\deg_{in}(G', G) = \sum_{u \in S', v \in S', (u,v) \in A'} \omega_{u,v}$ et $\deg_{out}(G', G) = \sum_{u \in S', v \in S \setminus S', (u,v) \in A} \omega_{u,v}$.

δ_r mesure l'isolement de la composante connexe et δ_l mesure la densité des composantes connexes. Le score f est tel que moins un über-operon est constitué d'opérons avec des gènes insérés, plus son score sera élevé. Plus un über-operon a ses gènes dans les mêmes opérons pour diverses espèces, plus son score sera élevé.

1.2.3 Extension à des génomes quelconques

Nous avons détaillé un algorithme capable de détecter des über-operons. La donnée initiale étant un ensemble d'opérons, cela limite l'usage aux génomes pour lesquels il existe de telles structures : bactéries, et éventuellement nématodes.

Cependant si des suites de gènes coexistent dans un ensemble de génomes, alors on peut utiliser ces suites de gènes comme des opérons. Utiliser, par exemple, les intervalles communs est sans doute trop restrictif à cause de la nécessité que tous les gènes soient présents dans la structure de base pour construire la super-structure. C'est une propriété que l'on avait implicitement sur les opérons : le contenu en gènes de ceux-ci n'est pas le même pour toutes les espèces.

On peut alors utiliser la distance entre gènes pour calculer les relations d'adjacence entre gènes. Une fois donné un paramètre γ fixant la distance maximale, on construit un graphe comme celui des opérons, puis on extrait les composantes α-connexes pour obtenir des super-structures. Un exemple illustratif est donné Figure 1.10 et une application aux génomes mitochondriaux de maïs est donnée Section 1.4.

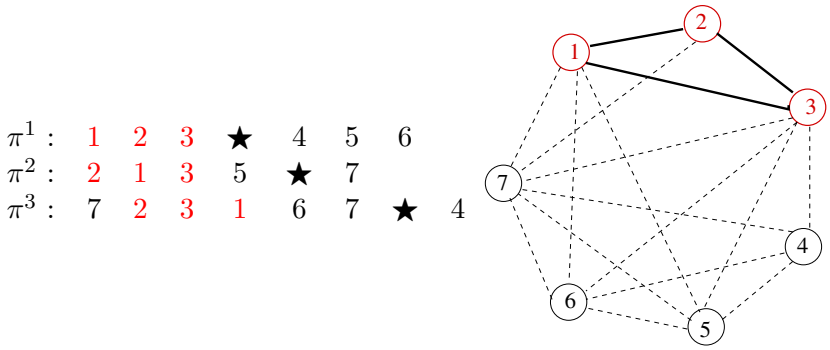


FIG. 1.10 – Illustration de la détection de super-structures basées sur les relations d'adjacence. Le paramètre d'adjacence γ a été fixé à 2. Ainsi il existe un arc entre deux sommets g_1 et g_2 du graphe si le nombre de gènes séparant g_1 et g_2 est inférieur à 2. Les arcs en pointillés ont un poids de 1 alors que les arcs en trait plein ont un poids de 3. En prenant α égal à 2 ou 3, on détecte que $\{1, 2, 3\}$ forme une structure. Pour détecter la structure $\{1, 2, 3, 5\}$ il faudrait $\gamma = 3$ et $\alpha = 2$.

1.2.4 Application à la détection des über-operons chez les bactéries.

Nous avons mené une étude sur une soixantaine de génomes procaryotes grâce au logiciel HUGO[3] qui implémente la méthode de détection décrite ci-dessus.

Nous n'avons pas discuté jusqu'à maintenant du traitement réservé aux gènes dupliqués dans la méthode. Deux cas de figure sont possibles. Soit les dupliqués sont dans des opérons différents, soit ils sont dans le même opéron. Pour le premier cas, cela ne pose pas de problème particulier puisque les arcs du graphe représentent les liens entre les gènes d'un opéron, on aura alors deux faisceaux d'arcs : l'un provenant de la première occurrence, l'autre provenant de la seconde occurrence. Pour le second cas, il se pose la question de savoir si le poids des arcs issus du gène dupliqué doit être doublé ou non. Nous avons pris le parti de ne le compter qu'une seule fois pour rester fidèle à la définition qu'un arc reflète l'appartenance d'un gène à un ensemble.

Familles de gènes. Nous avons choisi d'utiliser les COG, *Clusters of Orthologous Group* [115], pour identifier les familles de gènes. Les relations d'orthologie et de paralogie des COG sont déterminées en comparant la similarité des gènes mais aussi en prenant en compte les relations complexes avec les protéines à domaines multiples. Nous avons ainsi disposé d'un ensemble d'un peu plus de 138000 protéines.

Détection des opérons. Les opérons ne sont bien sûr pas connus pour l'ensemble des espèces séquencées. Il faut alors utiliser une méthode de prédiction de ceux-ci. Nous avons choisi une méthode simple basée sur la distance entre gènes. Plusieurs études [83, 93, 76, 62] ont montré que cette distance semblait contrainte et qu'elle fournissait un bon indicateur de regroupement en opérons des gènes chez les procaryotes. La répartition des opérons prédits suivant leur taille pour 93 génomes séquencés est donnée Figure 1.11. Notons enfin que la méthode de détection des opérons n'a pas à être trop spécifique puisque les faux positifs, à moins d'avoir un même

[3]Hierarchical Union of Genes from Operons, disponible sur `http://bioinfo.lifl.fr`

faux positif dans de nombreuses espèces, ne fourniront pas des relations entre gènes suffisamment fortes pour être détectées comme étant un über-operon.

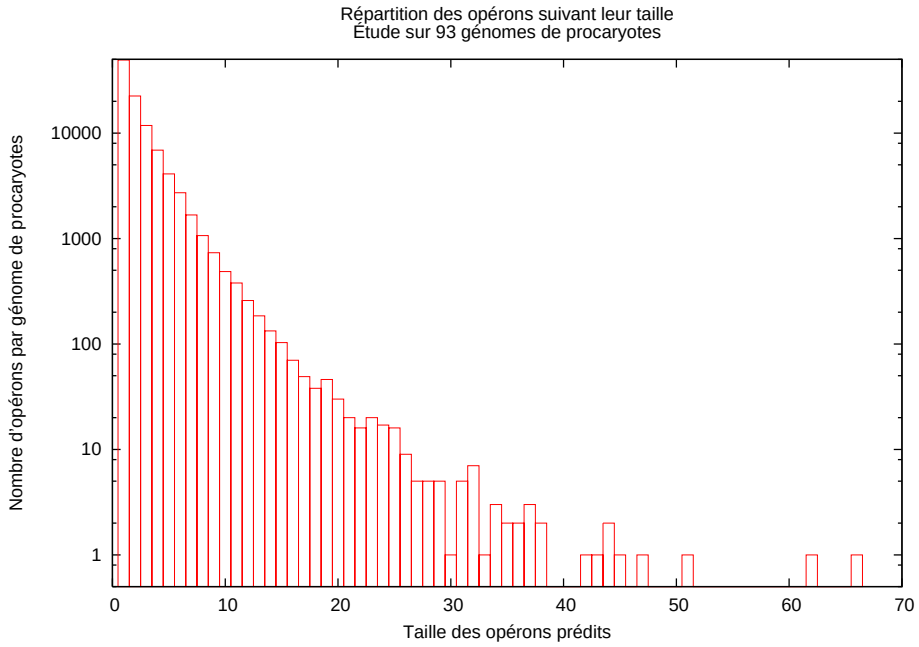


FIG. 1.11 – Distribution des opérons prédits suivant leur taille

Sélection des espèces. La présence d'espèces très proches peut modifier le résultat en donnant un poids plus important à certaines relations du fait de la présence d'opérons identiques conservés. Cependant nous avons voulu veiller à ne pas éliminer des espèces possédant des opérons semblables, à quelques remaniements près. Ainsi nous avons choisi pour exprimer la proximité des espèces de mesurer le taux de paires de gènes adjacents, comme dans [97, 96]. Nous représentons alors toutes les espèces dans un graphe reliées entre elles sous la condition que leur taux de paires de gènes adjacents soit supérieur à un seuil fixé (c'est-à-dire très proches). Il suffit ensuite d'extraire de ce graphe un sous-ensemble le plus grand de sommets non reliés entre-eux. Ce problème, connu sous le nom de *Maximum Independent Set* est malheureusement NP-complet [50]. Cependant, la taille du problème que nous traitons est raisonnable et a pu être résolu grâce à l'algorithme présenté dans [11]. Il a résulté une sélection de 54 espèces sur les 63 espèces de l'étude.

Exemple de résultat. Nous nous sommes intéressés à la capacité de détection de notre algorithme pour l'ensemble des gènes autour du mécanisme du flagelle. Le mécanisme du flagelle est un mécanisme complexe permettant aux bactéries qui en sont équipées de se déplacer. D'après la base de données KEGG [82], le flagelle et les mécanismes permettant le déplacement des bactéries

sont constitués d'une trentaine de gènes (voir Figure 1.12). Nous comparons les résultats obtenus avec notre méthode à ceux des Gene Teams et des Connected Gene Neighborhood.

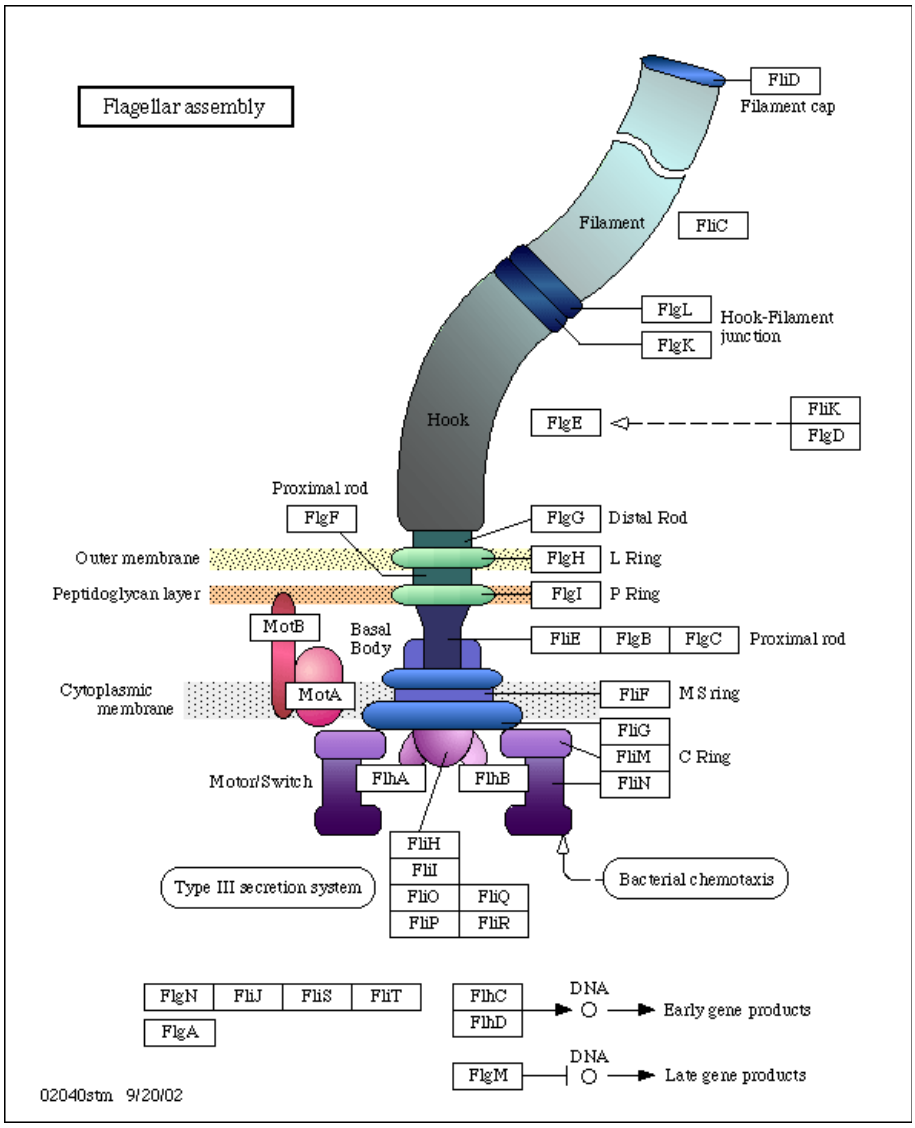


FIG. 1.12 – Le mécanisme du flagelle chez les bactéries (source : KEGG [82]).
Le flagelle est un mécanisme complexe, d'une part il est constitué d'un ensemble de protéines formant la cellule filamenteuse, et d'autre part il est constitué de « moteurs » actionnant la structure filamenteuse. De plus les « moteurs » interagissent avec d'autres systèmes comme la détection de stimulus extérieurs. Le système de sécrétion de type III chez les bactéries est lié au mécanisme du flagelle comme l'est également le mécanisme du chimiotactisme.

a) Détection de clusters de gènes avec HUGO. HUGO nous permet de détecter de gènes lié au mécanisme du flagelle. L'über-operon détecté par HUGO regroupe 34 gènes liés au flagelle mais aussi 5 gènes du système de sécrétion de type III. L'über-operon forme ainsi

un groupe de 39 gènes. On observe bien dans la représentation hiérarchique (Figure 1.13) le regroupement des gènes liés au système de sécrétion de type III et celui des gènes liés plus spécifiquement au flagelle. En plus des 39 gènes précédents on trouve aussi 6 gènes du chimiotactisme et 3 autres gènes en rapport avec le flagelle (groupe H et gènes flgMNA). On remarque aussi le regroupement des gènes motA et motB avec ceux du chimiotactisme, gènes impliqués dans le mécanisme du moteur flagellaire. De plus, les gènes dont les protéines interagissent étroitement ensembles sont regroupés, comme flhAB (groupe A Figure 1.13), flgBC (groupe C), flgFGHI (groupe F).

b) Détection de clusters de gènes avec les Gene Teams. L'expérience de détection des gènes liés à ce mécanisme avec les Gene Teams ne s'est pas avérée concluante, notamment parce que les Gene Teams souffrent de la nécessité que l'ensemble des gènes de la structure d'intérêt soient présents dans tous les génomes étudiés, ce qui limite fortement leur capacité de détection sur un grand nombre de génomes. Avec les quatre espèces relativement proches dont nous avons déjà discuté (Figure 1.5), *Escherichia coli K12*, *Bacillus subtilis*, *Clostridium acetobutylicum* et *Yersinia pestis*, on a détecté cinq ensembles de gènes comptant au total dix-sept de la trentaine gènes liés au mécanisme du flagelle. Nous avons utilisé des valeurs allant de 11 à 50 pour le paramètre δ. Il est évident que si on ajoute encore des espèces à la comparaison, les groupes de gènes seront de plus en plus morcelés.

c) Détection de clusters de gènes avec les Connected Gene Neighborhoods. Les CGN donnent six groupes de gènes relatifs aux mécanismes du flagelle regroupant la totalité des gènes auxquels quelques autres gènes sont ajoutés. Contrairement à notre approche, il n'y a pas de mise en évidence de cet ensemble avec d'autres ensemble de gènes dont les fonctions sont apparentées au mécanisme du flagelle.

1.3 Tri par inversion sous contrainte d'intervalles communs

Nous allons maintenant décrire les travaux où nous avons proposé de calculer des scénarios par inversions entre deux permutations qui respectent un ensemble d'intervalles communs. Les résultats ont été publiés dans [48]. Le problème que nous abordons est le suivant :

> Nombre minimal d'inversions respectant un ensemble d'intervalles communs
> Soit π une permutation, soit C un ensemble d'intervalles communs entre π et la permutation identité positive, calculer $d_C(\pi)$, le nombre minimum d'inversions tel que pour toute permutation π' du scénario de tri, C est inclus dans l'ensemble des intervalles communs de π' et la permutation identité positive.

Pour illustrer le problème, nous donnons deux exemples Figure 1.14. Remarquons qu'il n'est pas demandé de respecter l'ensemble des intervalles communs de π mais un sous-ensemble, qui, par exemple pourra être déterminé selon des critères biologiques. Remarquons également que si les intervalles communs de l'ensemble C ne contiennent pas de point de cassure (définition page 4), le scénario le plus parcimonieux respecte nécessairement les intervalles communs.

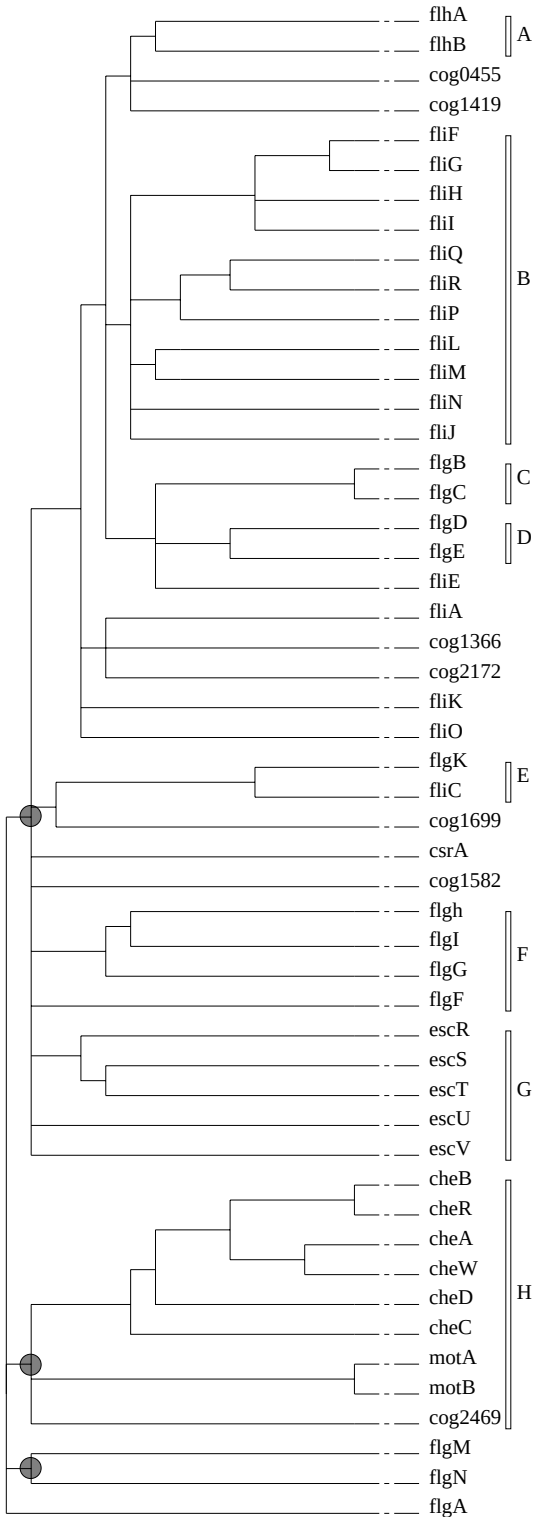


FIG. 1.13 – Cluster des gènes liés au mécanisme du flagelle (groupes A et C à E), au système de sécrétion de type III (groupe B) et du chimiotactisme (groupe H). Les trois gènes flgMNA sont également liés à ces mécanismes. La clusterisation met en avant les groupes selon leurs fonctions. Les trois groupes marqués montrent une interaction entre les fonctions.

a)

$$
\begin{array}{rcccccccccc}
\pi^s & = & \text{-}5 & +1 & \mathbf{\underline{-9}} & \mathbf{\underline{-8}} & \mathbf{\underline{-7}} & \underline{+4} & \underline{+6} & \mathbf{\underline{-3}} & \mathbf{\underline{-2}} \\
 & \rightarrow & \text{-}5 & +1 & \mathbf{+2} & \mathbf{+3} & \underline{\text{-}6} & \underline{\text{-}4} & \mathbf{+7} & \mathbf{+8} & \mathbf{+9} \\
 & \rightarrow & \text{-}5 & \underline{+1} & \underline{+\mathbf{2}} & \underline{+\mathbf{3}} & \underline{+4} & +6 & \mathbf{+7} & \mathbf{+8} & \mathbf{+9} \\
 & \rightarrow & \underline{\text{-}5} & \underline{\text{-}4} & \underline{\mathbf{-3}} & \underline{\mathbf{-2}} & \underline{\text{-}1} & +6 & \mathbf{+7} & \mathbf{+8} & \mathbf{+9} \\
 & \rightarrow & +1 & +2 & \mathbf{+3} & \mathbf{+4} & +5 & +6 & \mathbf{+7} & \mathbf{+8} & \mathbf{+9}
\end{array}
$$

b)

$$
\begin{array}{rcccccccc}
\pi^s & = & +2 & \mathbf{-5} & \underline{+\mathbf{7}} & \underline{+\mathbf{4}} & \mathbf{-6} & +1 & +3 \\
 & \rightarrow & +2 & \mathbf{-5} & \mathbf{-4} & \underline{\mathbf{-7}} & \mathbf{-6} & \underline{+1} & \underline{+3} \\
 & \rightarrow & +2 & \underline{\text{-}5} & \underline{\text{-}4} & \underline{\text{-}3} & \text{-}1 & +6 & +7 \\
 & \rightarrow & \underline{+2} & \underline{+3} & \underline{+4} & \underline{+5} & \text{-}1 & +6 & +7 \\
 & \rightarrow & \underline{\text{-}5} & \underline{\text{-}4} & \underline{\text{-}3} & \underline{\text{-}2} & \underline{\text{-}1} & +6 & +7 \\
 & \rightarrow & +1 & +2 & +3 & +4 & +5 & +6 & +7
\end{array}
$$

FIG. 1.14 – Comparaison de scénarios avec et sans conservation des intervalles communs. En gras sont représentés les intervalles communs à respecter, en souligné les gènes affectés par une inversion. a) un exemple de scénario parcimonieux qui respecte les intervalles communs $\{2, 3\}$ et $\{7, 8, 9\}$. b) un exemple où le scénario le plus parcimonieux ne respecte pas l'intervalle commun $\{4, 5, 6, 7\}$.

1.3.1 Algorithme de tri

Nous avons vu sur l'exemple Figure 1.14 qu'il existait des scénarios parcimonieux qui ne respectaient pas les intervalles communs. Existe-t-il toujours un scénario parcimonieux qui respecte les intervalles communs ? Est-il pour autant difficile de trouver un scénario parcimonieux qui respecte les intervalles communs ? La réponse à ces deux questions est négative. Un exemple où il n'existe pas de scénario parcimonieux respectant les intervalles communs est donné Figure 1.15. Cela implique qu'il est possible pendant une étape du scénario que l'on ne dispose d'aucune inversion triante. Il faut alors choisir l'inversion à réaliser.

Cas des intervalles communs non chevauchants. Dans un premier temps nous nous intéressons aux ensembles d'intervalles communs qui ne se chevauchent pas deux à deux. C'est-à-dire que pour chaque couple I, J d'intervalles communs de l'ensemble on a $I \subset J$ ou $J \subset I$ ou $I \cap J = \emptyset$. Dans ce cas, le théorème suivant permet de dissocier dans un scénario les inversions qui s'appliquent aux intervalles communs des autres.

Théorème 1.2 *Pour un ensemble d'intervalles communs non chevauchants , il existe toujours un scénario parcimonieux qui respecte les intervalles communs qui trie chaque intervalle commun avant de trier la permutation.*

On pourrait croire qu'il suffit de trier chaque intervalle commun pour arriver à une permutation intermédiaire π'. Comme un scénario parcimonieux ne crée pas de point de cassure, le tri de

a)

$\pi^s =$ **-2** **-3** +1
$\rightarrow$ **-2** -1 **+3**
$\rightarrow$ +1 **+2** **+3**

b)

$\pi^s =$ **-2** **-3** +1
$\rightarrow$ -1 **+3** **+2**
$\rightarrow$ +1 **+3** **+2**
$\rightarrow$ +1 **-3** **+2**
$\rightarrow$ +1 **-3** **-2**
$\rightarrow$ +1 **+2** **+3**

FIG. 1.15 – Exemple de permutation pour laquelle il n'existe pas de scénario le plus parcimonieux respectant les intervalles communs.
L'ensemble des intervalles communs entre π^s et la permutation identité est $\{2,3\}$ et $\{1,2,3\}$. Le scénario a) est l'unique scénario le plus parcimonieux. Il ne respecte pas l'intervalle commun $\{2,3\}$. Le scénario b) est le scénario le plus parcimonieux respectant tous les intervalles communs.

π' respectera les intervalles communs. Mais en fait on n'a aucune garantie sur la minimalité de la longueur du scénario : on sait que les scénarios de tri de chaque intervalle commun sont minimaux, et aussi que le scénario de tri de la permutation est également minimal, mais on ne sait rien sur la somme. L'exemple donné Figure 1.16 illustre cela. Le théorème précédent nous dit que ce scénario existe, nous allons maintenant voir comment le trouver. Comme toute inversion doit respecter les intervalles communs, la seule inconnue est de savoir si le tri de chaque intervalle doit se faire vers la permutation identité ou son inverse. Nous nous retrouvons dans un cas tout à fait semblable à celui du tri des permutations non signées. Il faut donc attribuer un signe aux intervalles communs.

Lemme 1.5 *Un intervalle commun est tel que :*
- *soit le scénario pour le trier vers la permutation croissante positive est de même longueur que pour le trier vers la permutation décroissante négative,*
- *soit il est de même longueur à plus ou moins un près.*

Dans le premier cas on le qualifiera de *neutre*, dans les deux autres cas respectivement de *positif* ou *négatif*. On aboutit au lemme suivant :

Lemme 1.6 *Soit c un intervalle commun de C tel qu'il n'existe pas d'intervalle c' de C tel que $c' \subset c$, alors :*
- *si c est positif, le scénario (global) le plus court est obtenu en triant c vers la permutation identité négative,*
- *si c est négatif, le scénario (global) le plus court est obtenu en triant c vers la permutation identité positive.*

Malheureusement, pour les intervalles communs neutres, il faut essayer les deux attributions de signe. Le squelette de l'algorithme devient alors : trier d'abord les intervalles dont le niveau d'inclusion est le plus faible pour terminer par ceux dont le niveau d'inclusion est le plus élevé (remarquons qu'il suffit de les envisager par ordre croissant de taille), suivant le type d'intervalle

$\pi^s =$ +**3** +**2** +5 +1 +4
$\rightarrow$ -**2** -**3** +5 +1 +4
$\rightarrow$ +**2** -**3** +5 +1 +4
$\rightarrow$ +**2** +**3** +5 +1 +4
$\rightarrow$ +**2** +**3** +5 -1 +4
$\rightarrow$ +1 -5 -**3** -**2** +4
$\rightarrow$ +1 -5 -4 +**2** +**3**
$\rightarrow$ +1 -**3** -**2** +4 +5
$\rightarrow$ +1 +**2** +**3** +4 +5

$\pi^s =$ +**3** +**2** +5 +1 +4
$\rightarrow$ -**3** +**2** +5 +1 +4
$\rightarrow$ -**3** -**2** +5 +1 +4
$\rightarrow$ -**3** -**2** +5 -1 +4
$\rightarrow$ +1 -5 +**2** +**3** +4
$\rightarrow$ +1 -5 -4 -**3** -**2**
$\rightarrow$ +1 +**2** +**3** +4 +5

FIG. 1.16 – Choix de tri d'un intervalle commun.

Les deux scénarios présentés respectent les intervalles communs et appliquent le principe de tri des intervalles communs avant de trier la permutation. La différence, pour les premières opérations est qu'en a) on trie l'intervalle $\{2,3\}$ vers la permutation +2 +3 alors que pour b) on trie l'intervalle $\{2,3\}$ vers la permutation -3 -2. Ensuite, dans les deux scénarios, on applique un nombre minimum d'inversion pour obtenir la permutation identité.

commun faire le choix ou essayer les deux possibilités. L'algorithme est donné Algorithme 1. Bien sûr dans le pire des cas l'algorithme s'exécute en temps exponentiel.

Théorème 1.3 *Pour une permutation signée π^s et C un ensemble d'intervalles communs qui ne se chevauchent pas et trié par taille croissante de ses éléments, l'algorithme* TRIICNC *calcule $d_C(\pi^s)$.*

La preuve se fait par récurrence sur la taille des intervalles communs envisagés. Il suffit de montrer que si l'algorithme calcule correctement $d_C(\pi^s)$ pour un ensemble C dont la taille du plus grand intervalle commun est k, alors il calculera correctement $d_{C'}(\pi^s)$ avec C' contenant C et auquel sont ajoutés des intervalles communs de taille $k+1$.

Cas des intervalles communs chevauchants. Le cas général où des intervalles communs peuvent se chevaucher est plus délicat. Lorsque deux intervalles communs se chevauchent alors il existe nécessairement d'autres intervalles communs, inclus dans ou incluant les deux chevauchants.

Lemme 1.7 *Pour tout couple d'intervalles communs chevauchant, $[i,j]$ et $[k,\ell]$, $k<j$, il existe quatre autres intervalles communs, $[i,k-1],[k,j],[j+1,\ell]$ et $[i,\ell]$.*

On a vu précédemment que l'on pouvait trier indépendamment des intervalles non chevauchants. Comme de plus les intervalles doivent être triés par ordre croissant de taille, on montre que l'on peut trier deux intervalles chevauchants en triant d'abord les sous-intervalles. Pour cela nous avons étendu les notions de bloc et de strip introduites par Hannenhalli et Pevzner définies page 12. Nous retrouvons des propriétés sur les strips de taille trois. Rappelons qu'un bloc est une suite de gènes d'une permutation ne contenant aucun point de cassure.

Entrée : π^s, C un ensemble d'intervalles communs non chevauchants trié par taille croissante de ses éléments
Sortie : $d_C(\pi^s)$
si $C = \emptyset$ **alors**
retourne $d(\pi^s)$
sinon
$c \leftarrow C[0]$ $\{c \leftarrow \pi^s([x+1, y-1])\}$
si $d(c, +c) < d(c, -c)$ **alors**
retourne $d(c, +c) + \text{TRIICNC}(\pi^s([1, x]) \cdot +c \cdot \pi^s([y, n]), C \setminus \{c\})$
sinon si $d(c, -c) > d(c, +c)$ **alors**
retourne $d(c, -c) + \text{TRIICNC}(\pi^s([1, x]) \cdot -c \cdot \pi^s([y, n]), C \setminus \{c\})$
sinon
retourne $\min \begin{cases} d(c, +c) + \text{TRIICNC}(\pi^s([1, x]) \cdot +c \cdot \pi^s([y, n]), C \setminus \{c\}) \\ d(c, -c) + \text{TRIICNC}(\pi^s([1, x]) \cdot -c \cdot \pi^s([y, n]), C \setminus \{c\}) \end{cases}$
fin si
fin si

Algorithme de tri par inversion respectant un ensemble d'intervalles communs non chevauchants. $\pi^s([x, y])$ désigne la sous-permutation constituée des éléments de π^s compris entre les indices x et y (éventuellement la permutation sans éléments si $y < x$), $+c$ et $-c$ représentent la permutation constitués des éléments de c triés soit dans l'ordre croissant et tous positifs, soit dans l'ordre décroissant et tous négatifs. Le '$\cdot$' est l'opérateur de concaténation.

Algorithme 1 : TRIICNC : Tri avec Intervalles Communs Non Chevauchants

Un *bloc multiple* est une liste de blocs consécutifs $(b_i, \ldots, b_j)$. Par exemple (+1 +2 -4 +3 +5) est un bloc multiple constitué des deux blocs b_1 = +1 +2 et b_2 = -4 +3 +5. Ces deux blocs sont consécutifs car l'élément maximal de b_1 est 2 et l'élément minimal de b_2 est 3. Un bloc multiple $b_i, ..., b_j$ est un bloc multiple *croissant* si des éléments (en valeur absolue) maximaux de chaque bloc est croissante, sinon c'est un bloc multiple *décroissant*. Le bloc multiple précédent est croissant, alors que le bloc multiple (-4 +3 +5 +1 +2) est décroissant.

Un *multi-strip*, croissant ou décroissant, est un bloc multiple tel que ses blocs soient maximaux. Prenons par exemple la permutation signée π^s = (+1 +2 +3 -6 -5 -4 -9 -8 -7), elle possède un multi-strip croissant composé de trois blocs maximaux : $\pi^s[1, 3] = \{1, 2, 3\}, \pi^s[4, 6] = \{4, 5, 6\}$ et $\pi^s[7, 9] = \{7, 8, 9\}$. Le lemme suivant est important puisqu'il montre que trier deux intervalles communs chevauchants amène toujours à un multi-strip.

Lemme 1.8 *Pour deux intervalles communs chevauchants, $[i, j]$ et $[k, \ell]$, $k < j$, d'une permutation signée π^s, trier les trois intervalles communs non chevauchants $[i, k-1]$, $[k, j]$ et $[j+1, \ell]$ donne un multi-strip croissant ou décroissant.*

Grâce aux deux lemmes précédents, nous savons qu'il est possible d'obtenir un multi-strip en triant séparément les trois sous-intervalles communs de deux intervalles chevauchants. On indique maintenant comment trier un multi-strip.

Lemme 1.9 *Trier un multi-strip croissant (respectivement décroissant) avec k blocs anti-canoniquement signés (respectivement canoniquement signés), en respectant les intervalles communs,*

nécessite exactement k inversions retournant chacune un bloc anti-canoniquement signé (respectivement canoniquement signé).

Poursuivons avec l'exemple de la permutation $\pi^s = (+1\ +2\ +3\ -6\ -5\ -4\ -9\ -8\ -7)$. C'est un multi-strip, croissant, qui possède deux blocs anti-canoniquement signés : $\pi^s[4,6] = \{4,5,6\}$ et $\pi^s[7,9] = \{7,8,9\}$. Il faut une inversion sur chacun de ces deux blocs pour trier π^s. On sait maintenant comment trier des intervalles communs chevauchants.

Lemme 1.10 *Pour deux intervalles communs $[i,j]$ et $[k,\ell]$ chevauchants avec $k < j$, les trier sans casser un seul intervalle commun est optimalement fait en triant les trois intervalles communs $[i,k-1]$, $[k,j]$ et $[j+1,\ell]$, et finalement en triant le multi-strip résultant.*

Cependant, il reste la question des intervalles communs neutres. Trier un multi-strip en respectant les intervalles communs demande un nombre d'inversions correspondant au nombre de blocs positifs ou négatifs. Or si dans un multi-strip, un strip provient d'un intervalle commun neutre, il existe deux configurations pour ce multi-strip. Il existe une des configurations où cet intervalle commun est un strip croissant canoniquement signé et une autre où il est un strip décroissant canoniquement signé. Une des deux configurations est inutile car suivant le multi-strip, au moment de le trier, il faudra rendre tous ses strips soit positifs, soit négatifs. Il est possible d'éliminer la configuration inutile et par conséquent éliminer un des deux appels récursifs du cas des intervalles communs neutres. Toutefois nous arrêtons là la description des propriétés de l'algorithme qui ajouteraient encore des définitions techniques. L'algorithme est présenté Algorithme 2, le choix des signes non déterminés est réalisé lors de l'appel récursif terminal. Nous renvoyons le lecteur à [47, 48] pour plus de détails.

1.3.2 Complexité du calcul de la distance d'inversions sous contrainte d'intervalles communs

Nous avons vu dans la partie précédente que les algorithmes proposés restaient exponentiels dans le cas général.

Théorème 1.4 *Le calcul de $d_C(\pi^s)$ est NP-dur.*

Ce théorème s'appuie sur le fait que l'instance suivante du problème général est un problème NP-dur :

> Soit C un ensemble d'intervalles communs tels que pour tout c de C et c' de C, $c \neq c'$, on a $c \cap c' = \emptyset$, π^s une permutation signée et σ la permutation identité positive, calculer $d_C(\pi^s)$ le nombre minimum d'inversions respectant C permettant de passer de π^s à σ.

Afin de prouver que ce sous-problème est NP-dur, nous avons effectué une réduction polynomiale du problème du calcul du nombre minimum d'inversions entre une permutation non signée et la permutation identité croissante non signée vers ce problème. Rappelons que le calcul de cette distance est NP-dur [30].

L'idée est de remplacer chaque élément e de la permutation non signée π^u par un quintuplé qui sera un intervalle commun de la forme :

$$+(5 \times e - 1) \quad +(5 \times e - 4) \quad +(5 \times e - 2) \quad +(5 \times e) \quad +(5 \times e - 3)$$

Entrée : π^p une permutation partiellement signée et C une suite d'intervalles communs triés dans l'ordre croissant
Sortie : $d_C(\pi^p)$
si $C = \emptyset$ **alors**
retourne $d(\pi^p)$
sinon
$c \leftarrow C[0]$ $\{c \leftarrow \pi^s([x+1, y-1])\}$
si c est un multi-strip croissant **alors**
d :=nombre_de_blocs_négatifs (c)
retourne $d + \text{TRIIC}(\text{COLLAPSE}(\pi^p([1,x]) \cdot {}_+c \cdot \pi^p([y,n])), C \setminus \{c\})$
sinon si c est multi-strip décroissant **alors**
d :=nombre_de_blocs_positifs (c)
retourne $d + \text{TRIIC}(\text{COLLAPSE}(\pi^p([1,x]) \cdot {}_-c \cdot \pi^p([y,n])), C \setminus \{c\})$
sinon si $d(c,+c) < d(c,-c)$ **alors**
retourne $d(c,+c) + \text{TRIIC}(\text{COLLAPSE}(\pi^p([1,x]) \cdot {}_+c \cdot \pi^p([y,n])))$
sinon si $d(c,+c) > d(c,-c)$ **alors**
retourne $d(c,-c) + \text{TRIIC}(\text{COLLAPSE}(\pi^p([1,x]) \cdot {}_-c \cdot \pi^p([y,n])))$
sinon
retourne $d(c,+c) + \text{TRIIC}(\text{COLLAPSE}(\pi^p([1,x]) \cdot {}_\pm c \cdot \pi^p([y,n])), \text{COLLAPSE}(C \setminus \{c\}))$
fin si
fin si

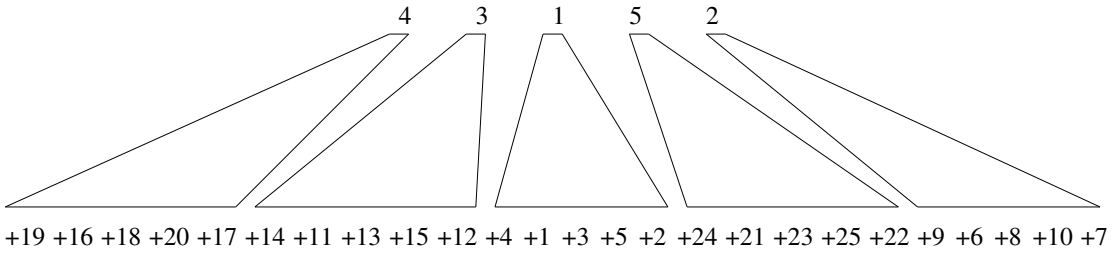
Algorithme de tri pour un ensemble quelconque d'intervalles communs. L'opération COLLAPSE comprime une suite de gènes croissante positive ou décroissante négative en un seul gène. Les autres notations sont identiques à celles de l'algorithme 1.

Algorithme 2 : TRIIC : Tri avec Intervalles Communs

afin de construire une permutation signée. Comme ces intervalles communs sont non chevauchants, on sait qu'il existe un scénario parcimonieux où l'on va trier chaque intervalle de manière indépendante avant de trier la permutation résultante. Chacun de ces intervalles communs nécessite cinq inversions pour être trié soit positivement, soit négativement, ces intervalles sont donc neutres. Il faut donc déterminer le signe de chacun et cela revient à calculer $d(\pi^u)$. On a ainsi :

$$d(\pi^u) = d_C(\pi^s) - 5 \times n$$

Un exemple est donné Figure 1.17.



$$C = \{[1,5],[6,10],[11,15],[16,20],[21,25]\}$$

$$\begin{aligned} d_C(\pi^s) &= d([1,5]) + d([6,10]) + d([11,15]) + d([16,20]) + d([21,25]) + d(\pm 4\ \pm 3\ \pm 1\ \pm 5\ \pm 2) \\ &= 5 \times n + d(\pi^u) \end{aligned}$$

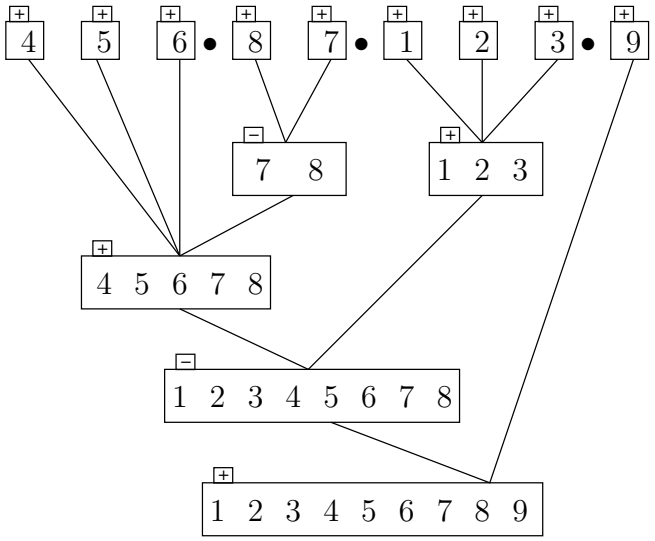
FIG. 1.17 – Réduction polynomiale du problème de tri non signé au problème de tri signé avec contrainte d'intervalles.
On transforme la permutation non signée en une permutation signée en remplaçant chaque élément par un intervalle commun de même structure, possédant la propriété d'être neutre. Il faut donc trier chaque intervalle et décider du signe de chacun.

1.3.3 Liens avec les travaux ultérieurs

Nous nous sommes intéressés à trouver un algorithme résolvant le problème général et à trouver des propriétés permettant d'obtenir un algorithme le plus efficace possible malgré une complexité exponentielle. Des problèmes connexes furent explorés ensuite. Existe-t-il un scénario parcimonieux au sens de la distance par inversion qui conservent les intervalles communs [92, 40] ? Existe-t-il des classes de permutations pour lesquelles le tri respectant les intervalles communs est facile [29, 12] ? Dans la littérature actuelle, les scénarios respectant les intervalles communs sont appelés scénarios *parfaits*.

Dans [12] ce que nous avons nommé intervalles non chevauchants est appelé intervalles forts comme dans [69]. L'ensemble des intervalles forts peut être représenté, comme nous l'avons mentionné, dans un arbre (Figure 1.18). Les nœuds sont typés suivant qu'ils sont, en quelque sorte, faciles à trier ou non. Très succinctement, si le sous-intervalle est facile à trier (non neutre) alors il sera représenté de type Q et de type P sinon (la classification est plus subtile que cela mais nous ne souhaitons pas donner l'ensemble des définitions mais se limiter à l'intuition). On

peut ensuite donner un signe à chaque nœud comme nous avions proposé de signer les intervalles, ce qui indique vers quelle permutation canonique le tri doit être effectué : positive ou négative (voir Figure 1.18). On comprend le lien entre ce que nous avons développé et cette représentation



+4	+5	+6	+8	+7	+1	+2	+3	+9
+4	+5	+6	-8	+7	+1	+2	+3	+9
+4	+5	+6	-8	-7	+1	+2	+3	+9
+4	+5	+6	+7	+8	+1	+2	+3	+9
-8	-7	-6	-5	-4	+1	+2	+3	+9
-8	-7	-6	-5	-4	-3	-2	-1	+9
+1	+2	+3	+4	+5	+6	+7	+8	+9

FIG. 1.18 – Utilisation d'un PQ-tree pour trier.

À chaque fois que l'on observe un changement de signe entre un noeud et l'un de ses fils, on doit effectuer une inversion sur le fils. Si les feuilles 7 et 8 avaient été de signe moins (la permutation à trier aurait été +4 +5 +6 -8 -7 +1 +2 +3), les deux premières inversions n'auraient pas eu à être réalisées. Le signe du nœud 7,8 aurait été le même que celui des feuilles 7 et 8. Les • représentent les points de cassure.

même si l'algorithme auquel nous avons aboutit n'est pas le même [13]. Dans ce même article, une amélioration de [12] est proposée grâce à la combinaison de la caractérisation des intervalles et de l'arbre des intervalles forts. Dans [20] les auteurs ont étendu le problème au calcul du génome médian en considérant la conservation des intervalles communs.

1.4 Analyse des génomes mitochondriaux de plantes

En 2006 a commencé avec la thèse de Aude Darracq une étude du génome mitochondrial de la betterave en collaboration avec Pascal Touzet du laboratoire de Génétique et Évolution des Populations Végétales, UMR CNRS 8016. Ce travail a pour but l'analyse des réarrangements intervenus dans l'histoire de l'espèce et donc la comparaison de génomes d'haplotypes. Il est connu que les génomes mitochondriaux des plantes subissent de nombreux réarrangements [84, 99, 4]. Cela en fait un sujet d'étude très intéressant pour comprendre pourquoi et comment ces génomes recombinent autant, particulièrement au sein d'une même espèce. Un autre intérêt d'étudier ces génomes réside dans le fait que la stérilité mâle cytoplasmique provient du génome mitochondrial. La stérilité mâle cytoplasmique, caractérisé par l'incapacité à produire du pollen, est héritée par la mère. Ce phénomène, trouvé chez de nombreuses plantes (riz, maïs, betterave,

...) est utilisé pour créer des hybrides qui permettent d'augmenter la production.

Les données pour la betterave n'étant pas disponibles (seuls deux génomes séquencés [68, 99]), nous avons commencé par la culture des plantes puis l'extraction de l'ADN séquencé grâce à un projet Génoscope. Les séquences sont attendues pour le quatrième trimestre 2008. Une étude de la phylogénie chloroplastique a déjà été réalisée [46], ce qui nous a permis de sélectionner les espèces d'intérêt (Figure 1.19). Nous disposons donc actuellement des génomes des haplotypes Sv et Nv et le séquençage nous apportera les génomes des haplotypes A, B, E, G chez *Beta vulgaris maritima* ainsi que le génome d'un groupe externe *Beta vulgaris macrocarpa*.

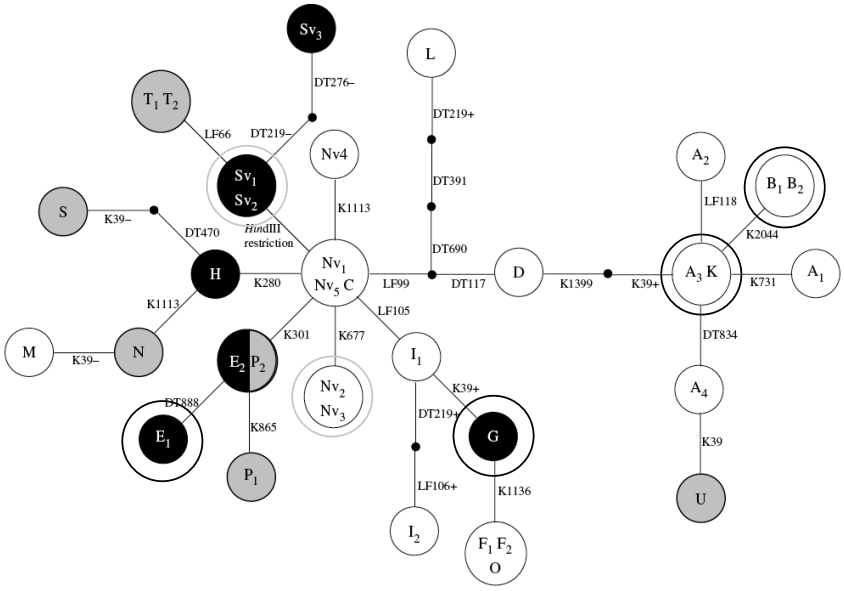


FIG. 1.19 – Phylogénie des haplotypes chloroplastiques [46].
Cet arbre nous a servi de guide pour choisir les haplotypes à séquencer. Nv et Sv sont déjà séquencés (entourés en gris clair). A, B, E et G seront séquencés (entourés en noir).

Les génomes mitochondriaux ont été l'objet des démonstrations des premiers articles sur les réarrangements [55, 98, 26]. Les génomes mitochondriaux de plantes possèdent la particularité d'être beaucoup plus grands que ceux des animaux, les gènes sont très peu modifiés au cours de l'évolution, et le taux de similarité est souvent de plus de 99% entre les gènes orthologues.

Étude chez le maïs. Dans l'attente des séquences de betterave, nous avons voulu analyser les données chez le maïs qui doivent révéler une problématique proche de celle que nous allons trouver chez la betterave. Ces premiers résultats ont été publiés dans le cadre d'un poster [Darracq et *al.* 08] et nous préparons un article de revue.

Il existe cinq génomes disponibles [4], dont les caractéristiques sont résumées Table 1.1. Les auteurs de l'étude menée dans cet article soulignent que les génomes sont fortement réarrangés sans pour autant utiliser un outil afin de mesurer le taux de réarrangements ou d'analyser ceux-ci.

L'identification des blocs de synténie (détectés grâce à HUGO) montre des duplications de certains groupes de gènes sans pour autant que ceux-ci soient totalement conservés (Figure

Génome	Longueur (bp)	Longueur sans dupliqués
Zea mays mays cytoplasme fertile NA	701046	537147
Zea mays mays cytoplasme fertile NB	569630	520223
Zea mays mays cytoplasme stérile CMS-C	739719	506772
Zea mays mays cytoplasme stérile CMS-S	557162	512139
Zea mays mays cytoplasme stérile CMS-T	535825	509941
Zea mays parviglumis	680603	537235

Tab. 1.1 – Les génomes de maïs étudiés.
La longueur sans dupliqués est calculée en prélevant dans chaque génome les portions dupliquées à l'exception d'une seule.

1.20). Comment traiter ces données avec les outils actuels ? Le traitement d'exemplarisation fait perdre trop d'informations (voir page 15). D'ailleurs les phylogénies construites avec l'ensemble des exemplarisation sont discordantes. Le traitement de couplage maximal (voir page 15) pose des difficultés lorsque le génome est très réarrangé, comme celui de la CMS-T.

L'observation des données laisse penser qu'il existe deux phénomènes liés : d'une part des duplications en tandem, et d'autre part la conservation de proximité ou de blocs de proximité de certains gènes. Chez les génomes mitochondriaux animaux, une opération de réarrangement connue et mettant en œuvre une duplication est la *duplication en tandem avec perte aléatoire* (TDRL) qui consiste à dupliquer en tandem un bloc de gènes consécutifs et à supprimer immédiatement exactement une copie de chaque gène dupliqué [80, 32]. Bernt et *al.* [21, 87] ont proposé un algorithme heuristique permettant de prendre en compte ce type d'opération, qui permet d'expliquer simplement les nœuds P (difficiles à trier) de l'arbre des intervalles forts (voir Figure 1.21). Il se trouve que si l'on observe les données issues des génomes de maïs, nous pouvons observer ce phénomène également, sans que toutes les copies aient disparues. Un exemple pour le génome de *Zea mays parviglumis* est donné Figure 1.22.

1.5 Perspectives

La problématique qui nous a occupé durant ce chapitre était la relation entre la conservation des intervalles communs dans les génomes et l'évolution de ceux-ci dans des scénarios d'inversions. La mise au point de méthodes combinant opérations de réarrangement et conservation de structure est le cœur de ce que nous souhaitons continuer à développer.

Les problèmes issues des données que nous avons à traiter avec les génomes mitochondriaux de plantes permettent de préciser des pistes de recherche motivées par une difficulté à analyser des données réelles.

La première d'entre elles consiste en l'étude du modèle de réplication en tandem avec perte partielle. Étant donnés deux génomes pour lesquels on repère des parties dupliquées en tandem, avec erreurs, trouver un scénario combinant cette opération et les inversions. L'ensemble des problèmes algorithmiques habituels d'un nouveau modèle sont alors posés. La détection des répétions en tandem avec erreur doit pouvoir se faire en adaptant des résultats obtenus dans les extensions des intervalles communs avec erreur.

Une autre piste de recherche concerne la reconstruction de l'histoire des espèces en considérant

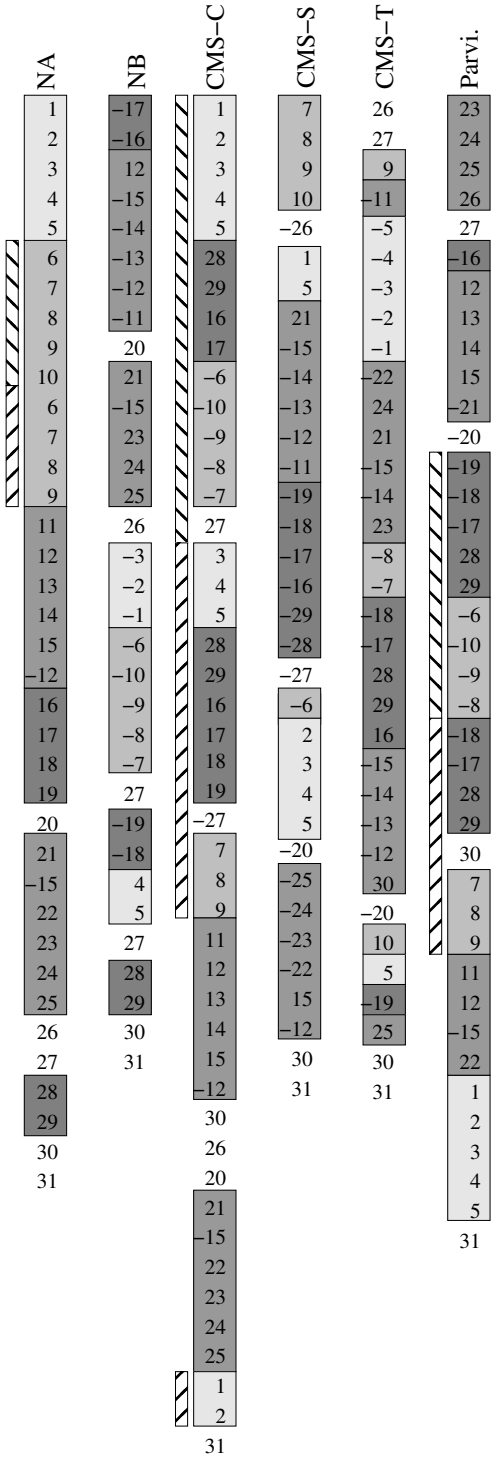


FIG. 1.20 – Blocs de synténie identifiés entres les génomes de maïs.
Nous avons appliqué notre algorithme de détection de super-structures aux données du génome mitochondrial de maïs. Cela permet de distinguer des groupes de gènes conservés et répartis sur le génome, représentés par les boîtes en niveaux de gris. La mise en avant de ces ensembles permet également de délimiter des duplications (boîtes hachurées) chez NA, CMS-C et *Zea mays parviglumis*.

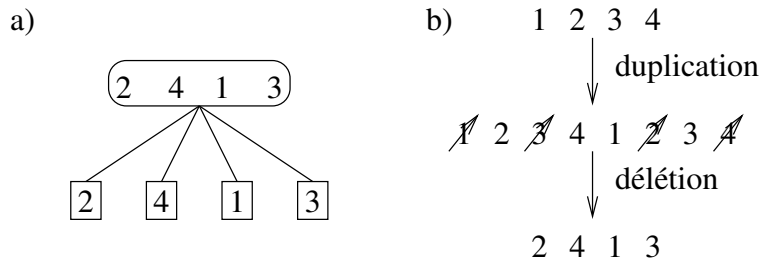


FIG. 1.21 – Détection d'un TDRL dans un arbre des intervalles communs forts (tiré de [87]). Les éléments 2,4,1,3 sont dans un ordre complètement déstructuré. C'est donc un nœud P. Il peut être facilement traité grâce à une duplication en tandem avec perte aléatoire. a) l'arbre, b) le scénario.

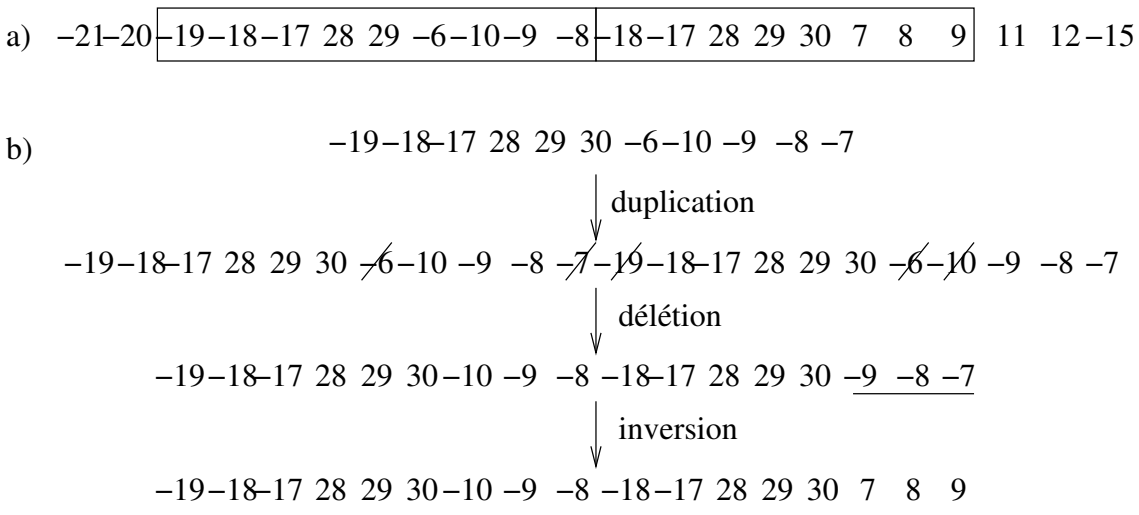


FIG. 1.22 – Illustration d'une duplication avec perte partielle.
a) une partie du génome de *Zea mays parviglumis*. b) un scénario d'évolution possible d'une séquence ancestrale à travers une duplication suivi de pertes de quelques gènes, combiné à des réarrangements internes.

des super-structures. Étant donnés un ensemble de génomes et un ensemble de super-structures, comment reconstruire les génomes ancestraux, grâce à des opérations de réarrangement, en supposant que chez l'ancêtre commun, les gènes étaient regroupés ? Faut-il supposer un ordre entre ces ensembles de gènes chez le génome ancestral ? Une méthode construisant une histoire pour chaque structure permettrait-elle d'aboutir à une histoire globale des génomes ? Il doit être possible de s'inspirer des travaux sur l'histoire des familles multigéniques pour répondre à ces questions.

Enfin, les génomes mitochondriaux sont la plupart du temps représentés par un cercle maître mais on sait qu'en fait ces génomes existent sous la forme de sous-cercles [84], ce qui expliquerait les recombinaisons au cours du temps. Ces recombinaisons étant possibles grâce à la présence de répétitions [45]. Établir un modèle capable de construire des scénarios sur la base de réarrangements entre sous-cercles est un sujet d'intérêt. Cela pose la question de l'opération à utiliser (par exemple le double-cut-and-join [126] est sans doute une piste), et nécessite la capacité à détecter les répétitions dans le génome susceptibles d'engendrer la formation de sous-cercles.

Chapitre 2

Analyse des régions cis-régulatrices

Le travail autour de l'analyse des sites de fixation des facteurs de transcription a débuté en 2003 dans l'équipe Bioinfo avec la thèse de Mathieu Defrance [38] encadrée par Hélène Touzet. Le travail porta sur la construction d'une méthode capable d'extraire des occurrences pertinentes de sites de fixation de facteurs de transcription dans des séquences supposées être régulées par les même facteurs. Il s'agissait notamment d'intégrer des données de génomique comparative afin d'accroître la qualité des prédictions. Un pré-requis à cette méthode était de connaître l'ensemble des occurrences des sites de fixation des facteurs d'intérêt. C'est dans ce sens que ce thème se développa ensuite en 2004 avec le mémoire de DEA puis la thèse de Aude Liefooghe [74] sur le développement d'algorithmes efficaces de localisation se sites de fixation co-encadré avec Hélène Touzet. Le développement d'une extension du thème sites de fixations autour de la recherche de modules cis-régulateurs a été initié avec le mémoire de master recherche de Sébastien Le Maguer, mi 2008, sous mon encadrement. Des extensions aux travaux de Mathieu Defrance ont été entrepris grâce à un stage de mastère recherche de Imen Hammami, fin 2008, sous mon encadrement.

2.1 Introduction et motivations

L'expression des gènes se fait à travers des mécanismes complexes qui permettent d'adapter la réponse au contexte dans lequel se trouve la cellule. C'est grâce à ces mécanismes que l'expression est différenciée suivant les tissus. Parmi les différentes étapes permettant l'expression, celle de l'initiation de la transcription du gène est la toute première. Pour débuter la transcription d'un gène en ARN, un complexe protéique appelé *ARN polymérase* vient se fixer en amont du gène, quelques bases avant le premier nucléotide transcrit appelé *site d'initiation de transcription*. Cette polymérase se chargera de transcrire le gène en ARN. Mais, pour que celle-ci puisse fonctionner, il est nécessaire qu'elle se trouve associée à d'autres éléments, des protéines auxiliaires, qui sont les *facteurs de transcription*.

Les facteurs de transcription viennent se fixer à l'ADN en reconnaissant de courts motifs, que l'on appelle *sites de fixation des facteurs de transcription*. Ces motifs sont généralement situés en amont du gène, dans la partie appelée promotrice. Suivant les organismes étudiés, la partie promotrice du gène peut être courte, quelques dizaines de nucléotides chez les bactéries, ou beaucoup plus longue, quelques milliers de nucléotides chez les eucaryotes. Sauf pour les sites les plus proches du site d'initiation de la transcription, la position des sites de fixation n'est pas

précise.

Les sites de fixation sont connus pour certains facteurs de transcription. Ils sont référencés dans des bases de données telles que Transfac [124] ou Jaspar [95]. Pour un même facteur, le segment d'ADN reconnu est variable (voir Figure 2.1). Si la reconnaissance d'un site connu dans un génome n'assure pas de sa pertinence biologique, c'est en tout cas une première étape dans le décryptage des mécanismes d'expression. D'autres aspects devraient être pris en compte, notamment l'accessibilité à ces sites [53].

```
        agatCTTGATGACGCAatacgccatg
 agatcttccagCCGGGTGACGCCatgg
   agatctgttCGCGATGACGTCccatgg
  ccatggacaaTTGGATGACGTAgatct
    agatcttaCGTAGTGACGTCgccatgg
     agatctgGCGGGTGACGTGtgccatgg
    agatcttgGCGGGTGACGTTctccatgg
         agaTCTGATGACGCGatgcccccatgg
agatctcggggTATGTTGACGCCatgg
        agatCTTCGTGACGTTctttgccatgg
        agatCTGGGTGACGACaattcccatgg
           aGATCTTGACGTCcgcaggtaccatgg
           aGATCTTGACGTAgttgagttccatgg
          ccaTGGGATGACGCGtccccgagatct
   agatctcggGGCAGTGACGTAccatgg
  ccatgggatgCCCGCTTACGTAgatct
```

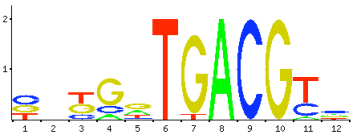


FIG. 2.1 – Exemple de sites reconnus pour le facteur de transcription CREB1 (tiré de la base de données Jaspar [95]).
Les lettres en majuscules caractérisent les site de fixation, elles correspondent au motif. À droite, le logo [100] qui représente la proportion relative de chaque lettre dans chaque colonne du motif. Il existe clairement un cœur conservé, contrairement aux extrémités, beaucoup plus variables. C'est fréquemment le cas pour les matrices représentant des sites de fixation de facteurs de transcription.

2.1.1 Matrices score-position, définitions et notations

La variabilité qui peut exister pour un site de fixation nécessite d'avoir une représentation capable de capturer celle-ci. L'utilisation d'une simple séquence consensus, même écrite sur un alphabet étendu ne permet pas de conserver la subtilité de composition de chaque position du site.

Les matrices score-position [110, 60, 111] fournissent un modèle probabiliste plus adéquat. La construction d'une matrice score-position se fait à partir d'un échantillon de sites de fixations connus pour un facteur de transcription donné (tel que celui Figure 2.1). L'idée de la matrice score-position est de mesurer l'adéquation d'un mot (le segment d'ADN potentiellement site de fixation) avec le motif. Pour cela, on calcule le logarithme du rapport entre la probabilité de ce mot dans le motif (l'alignement multiple) et celle du mot dans le modèle de fond (le génome, ou le promoteur). Pour réaliser ce calcul, on utilise les hypothèses : i) que les positions d'un site

sont indépendantes et ii) que le modèle de fond peut être simplement représenté par un modèle de Bernoulli. Cela a pour conséquence de disposer d'un système de score additif. Il suffit donc de définir le score d'une lettre à une position du site comme :

$$M(i,x) = \log_2\left(\frac{f(i,x)}{p(x)}\right)$$

où $f(i,x)$ est la fréquence de la lettre x à la position i du motif et $p(x)$ la probabilité de la lettre x dans le modèle de fond. Pour une matrice M, on obtient ainsi le score d'un mot $u = u_0..u_{m-1}$, avec la définition donnée plus haut, comme la somme des scores de chaque lettre du mot :

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

Les éléments $M(i,x)$ sont les éléments de la matrice de score.

Pour éviter des problèmes de sur-adaptation, le calcul des coefficients de la matrice se fait rarement avec la fréquence mais avec une fréquence corrigée [60] :

$$\hat{f}(i,x) = \frac{c(i,x) + p(i)}{\sum_{y\in\Sigma} c(i,y) + 1}$$

où $c(i,x)$ représente le nombre d'occurrences de la lettre y à la position i de l'alignement multiple. On obtient ainsi une matrice qui ne contient pas de score infini. Un exemple de calcul complet d'une matrice score-position est donné Figure 2.2.

A	0	3	0	2	5	0	0	16	0	0	1	5
C	7	5	3	3	1	0	0	0	16	0	5	6
G	5	4	6	11	7	0	15	0	0	16	0	3
T	4	4	7	0	3	16	1	0	0	0	10	2

$\downarrow \hat{f}(i,x)$

A	0.01	0.19	0.01	0.13	0.30	0.01	0.01	0.95	0.01	0.01	0.07	0.30
C	0.42	0.30	0.19	0.19	0.07	0.01	0.01	0.01	0.95	0.01	0.30	0.36
G	0.30	0.25	0.36	0.66	0.42	0.01	0.89	0.01	0.01	0.95	0.01	0.19
T	0.25	0.25	0.42	0.01	0.19	0.95	0.07	0.01	0.01	0.01	0.60	0.13

$\downarrow \log_2 \frac{\hat{f}(i.x)}{p(i)}$

A	-4.08	-0.38	-4.08	-0.9	0.30	-4.08	-4.08	1.93	-4.08	-4.08	-1.76	0.30
C	0.77	0.30	-0.38	-0.38	-1.76	-4.08	-4.08	-4.08	1.93	-4.08	0.30	0.55
G	0.3	0	0.55	1.40	0.77	-4.08	1.84	-4.08	-4.08	1.93	-4.08	-0.38
T	0	0	0.77	-4.08	-0.38	1.93	-1.76	-4.08	-4.08	-4.08	1.27	-0.91

FIG. 2.2 – Matrice score-position pour CREB1.

Obtention d'une matrice score-position pour le facteur CREB1 à partir de l'alignement multiple Figure 2.1. Les valeurs réelles ont été tronquées à deux chiffres après la virgule pour une question de présentation.

Dans la suite on notera $M[i..j]$ la sous-matrice composée des colonnes i à j incluses. On notera $\texttt{ScoreMax}(M[i..j]) = \sum_{k=i}^{j} \max_{x\in\Sigma} M(k,x)$ et $\texttt{ScoreMin}(M[i..j]) = \sum_{k=i}^{j} \min_{x\in\Sigma} M(k,x)$.

La suite du chapitre détaille les contributions apportées sur le thèmes de l'analyse des régions cis-régulatrices. La section 2.2 décrit les algorithmes élaborés pour la calcul efficace du seuil score et de la P-valeur d'un score d'une matrice score-position. La section 2.3 présente la technique de découpage et pré-calcul d'index pour une ou plusieurs matrices et les algorithmes correspondants. Dans la section 2.5 nous traitons du problème du calcul des occurrences d'une matrice grâce à l'extension de l'algorithme de Knuth-Morris-Pratt [67] aux matrices score-position. Nous introduisons ces problèmes ci-dessous.

2.1.2 Les problèmes abordés

Définition du score seuil. Maintenant que nous disposons d'une fonction de calcul de score pour un facteur d'une séquence à une position, il faut encore pouvoir dire si ce score est significatif ou non. Si on juge ce score significatif, alors on dira qu'il y a une *occurrence* de la matrice à cette position. La recherche d'occurrences d'une ou plusieurs matrices score-position nécessite ainsi de fixer un *score seuil* au delà duquel on considérera qu'il existe une occurrence de la matrice.

Il existe plusieurs façons de fixer ce score seuil. La plus communément utilisée consiste à considérer la significativité du score au sens probabiliste du terme [36, 111]. Autrement dit, on va calculer la P-valeur d'un score à une position et décider si cette position est occurrence en fonction de la P-valeur. C'est ensuite à l'utilisateur de choisir une P-valeur qui lui convienne.

La *P-valeur* de la matrice M et du score α, notée $\texttt{P-valeur}(M, \alpha)$ est la somme des probabilités de chaque score supérieur ou égal à α. Autrement dit $\texttt{P-valeur}(M, \alpha)$ est la probabilité de l'ensemble

$$\{u \in \Sigma^m \mid \texttt{Score}(u, M) \geq \alpha\}$$

Cela donne lieu au problème suivant :

De la P-valeur au score :

Étant donnés une matrice score-position M et un score α, calculer $\texttt{P-valeur}(M, \alpha)$.

Le calcul de cette quantité peut se faire simplement par énumération des $|\Sigma|^m$ mots pour une matrice de longueur m, ce qui est bien sûr très brutal. Une telle énumération n'est pas facilement réalisable si on ne veut envisager que les mots de score supérieur à un seuil. L'utilisation de la technique d'élagage proposée par Wu et *al.* [125], détaillée plus loin, permet de décider en cours d'énumération si un préfixe d'un mot a la possibilité d'atteindre le score seuil. Cela permet de réduire l'espace de recherche et a été utilisé dans [94] dans le cadre de l'énumération des mots, mais dans un autre but que celui du calcul de la P-valeur. Malgré cette technique, cette approche énumérative a le désavantage d'être lente dès que l'on veut effectuer des calculs sur des matrices dont la longueur dépasse une dizaine de colonnes.

Pour calculer la P-valeur, il ne nous importe pas de connaître l'ensemble des mots mais simplement le cardinal de celui-ci. En fait plutôt que d'énumérer les mots, on va énumérer les scores par ordre décroissant[4] et compter le nombre de mots pour chaque score. Cette fois on

[4]Généralement, les scores seuils sont élevés et il y a donc moins de mots de score supérieur au score seuil que de mots de score inférieur.

s'arrêtera dès que l'on aura atteint le score seuil. Toujours dans une approche énumérative, on peut alors utiliser d'autres « astuces ». Il peut être intelligent de réécrire la matrice sous une autre forme afin de favoriser la fonction d'élagage. Ainsi, on pourra réordonner les valeurs dans une colonne de la matrice. Chaque colonne peut être triée de manière décroissante, et les colonnes permutées en mettant en tête les plus discriminantes au sens de la différence de score maximale [10].

Plus formellement, le calcul effectif de la P-valeur peut être obtenu grâce à la définition intermédiaire de la fonction $Q(M, \alpha)$ qui représente la probabilité de l'ensemble

$$\{u \in \Sigma^m \mid \texttt{Score}(u, M) = \alpha\}$$

On obtient alors :

$$\texttt{P-valeur}(M, \alpha) = \sum_{s \geq \alpha} Q(M, s)$$

Le calcul de Q peut être réalisé par programmation dynamique grâce à la récurrence suivante :

$$\begin{aligned} Q(M[0..\text{-}1], s) &= \begin{cases} 1 & \text{si } s = 0 \\ 0 & \text{sinon} \end{cases} \\ Q(M[0..i], s) &= \sum_{x \in \Sigma} Q(M[0..i-1], s - M(i, x)) \times p(x) \qquad 0 \leq i \leq m-1 \end{aligned} \tag{2.1}$$

avec $M[0..\text{-}1]$ la matrice vide.

La complexité en temps est $\mathcal{O}(m|\Sigma|S)$, et la complexité en espace $\mathcal{O}(S)$, où S est le nombre de scores qui devront être visités dans la matrice de programmation dynamique. Si les coefficients de M sont entiers (positifs ou nuls), alors le nombre de scores S est borné par $m \times \max\{M(i, x) \mid x \in \Sigma, 0 \leq i \leq m-1\}$ [90].

L'équation 2.1 permet de résoudre le problème du calcul de la P-valeur associée à un score mais également celui du calcul du score associé à une P-valeur. Étant données une matrice M et une P-valeur P, $\texttt{Seuil}(M, P)$ est le score α tel que $\texttt{P-valeur}(M, \alpha) = P$. Savoir calculer ce score est d'un intérêt majeur puisque les logiciels utilisent souvent la P-valeur pour fixer le score seuil d'une matrice. On a donc un second problème :

Du score à la P-valeur

Étant données une matrice score-position M et une P-valeur P, calculer $\texttt{Seuil}(M, P)$.

$\texttt{Seuil}(M, P)$ est calculable à partir des valeurs de Q. On parcourt les valeurs $Q(M, s)$ dans l'ordre décroissant des scores s, en retenant la somme S. On recherche le plus grand score accessible (un réel x est un score *accessible* s'il existe un mot u tel que $\texttt{Score}(u, M) = x$) jusqu'à ce que la P-valeur $P = S/4^m$ soit atteinte. Beckstette et *al.* [10] ont combiné les techniques d'élagage, de réécriture de la matrice et le calcul de Q par programmation dynamique pour aboutir à une implémentation paresseuse du calcul de la P-valeur. En quelques mots, la combinaison de ces idées permet de remplir la matrice de programmation dynamique en diagonale en débutant par la diagonale principale, c'est-à-dire celle correspondant aux mots (et ses préfixes) de score maximal. Cela revient à envisager les scores par ordre décroissant.

L'algorithme que nous proposons en section 2.2 montrera qu'il est possible d'accélérer encore le calcul de la P-valeur en s'appuyant sur des itérations du calcul de Q.

Recherche des occurrences. Le problème de la recherche des occurrences d'une matrice score-position dans une séquence est sans doute le plus important, au moins du point de vue informatique.

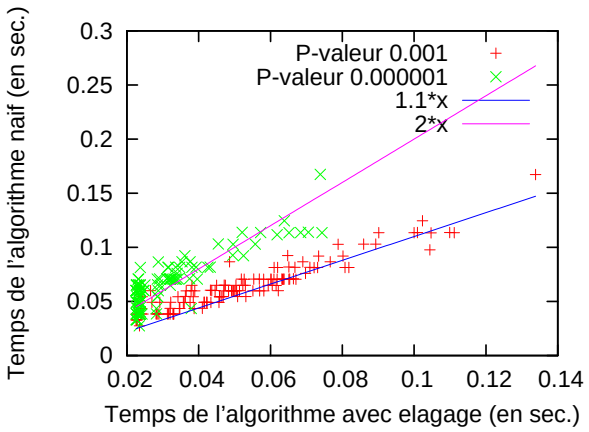
RECHERCHE DES OCCURRENCES

Étant donnés une matrice M de longueur m, un texte T de longueur n, et un score seuil α, trouver toutes les positions $0 \leq i \leq n-m$ de T telles que $\texttt{Score}(T_i..T_{i+m-1}, M) \geq \alpha$.

L'algorithme naïf, qui consiste à calculer le score à chaque position, a une complexité en $\Theta(m \times n)$. Il est mis en œuvre dans plusieurs outils [1, 31, 64]. La première optimisation de cet algorithme a été proposée par Wu et *al.* [125]. Elle vient du constat simple que, si à la colonne p de la matrice le score n'a pas atteint la borne inférieure

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

alors on ne peut pas atteindre le score seuil α. La conséquence directe est qu'il est alors possible d'abandonner le calcul du score pour le mot en cours et passer à la position suivante. La mise en œuvre de cette propriété dans l'algorithme de recherche est assez efficace (Figure 2.3). La complexité en temps devient alors $\mathcal{O}(m \times n)$. Remarquons qu'il est également possible d'aban-



P-valeur	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Position d'arrêt	3.8	2.8	2.2	1.8

FIG. 2.3 – Efficacité de la technique d'élagage.

Pour une séquence générée aléatoirement, nous avons lancé la recherche d'occurrences avec l'algorithme naïf et avec l'accélération par élagage sur l'ensemble des 123 matrices de JASPAR. a) nous reportons le temps de calcul sur une séquence aléatoire de 10^6 lettres pour un score seuil calculé avec une P-valeur de 10^{-3} et une P-valeur de 10^{-6}. Dans le premier cas un gain moyen de 10% est observé alors que dans le second cas un gain de 50% est observé. b) nous reportons la position dans la matrice où l'élagage a permis d'arrêter le calcul. La valeur donnée est une moyenne sur toutes les matrices, normalisée à une longueur de matrice de 10. Cela montre que l'on sait arrêter le calcul assez rapidement et démontre donc l'efficacité de l'élagage.

donner le calcul du score d'un mot dès lors que l'on sait que le score seuil sera atteint. Si à la position p, le score atteint la borne supérieure

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

alors nécessairement le score du mot sera supérieur à α.

Des techniques de pré-traitement du texte ont été mises en œuvre pour accélérer la recherche des occurrences d'une matrice : compression du texte [49], arbre des suffixes [42], table des suffixes [10]. Nous sommes intéressés pour notre part au pré-traitement des matrices. Les résultats sont développés Section 2.3 et Section 2.5.

Classification des matrices. La découverte de nouveaux sites de fixations, et donc l'établissement de nouvelles matrices, pose la question de savoir si celles-ci ne sont pas incluses dans celles déjà présentes dans les bases de données ou sont au moins assez proches de certaines (voir une illustration Figure 2.4). Répondre à cette question nécessite d'avoir un outil de comparaison de

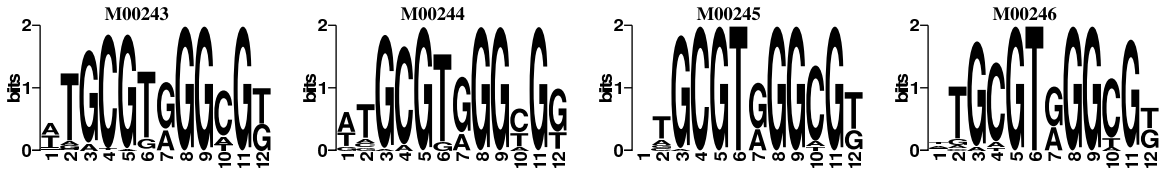


FIG. 2.4 – Quatre matrices similaires de la famille EGR-1 extraites de Transfac [124]

matrices. Cependant, posséder un tel outil permet d'utiliser la similarité entre les matrices pour accélérer la recherche d'occurrence. Ces deux applications ont été proposées pour argumenter des propositions d'outils de mesure de similarité [103, 65, 51]. La majorité des méthodes mettent en œuvre des calculs de type corrélation entre les colonnes des matrices. Parfois les mesures sont faites en comparant le nombre d'occurrences et leurs positions sur une séquence générée aléatoirement.

Nous verrons que nous avons proposé une mesure de similarité capable de quantifier le taux de faux positifs ou faux négatifs d'une matrice par rapport à une autre. Cela est particulièrement approprié lorsque l'objectif est de concevoir une méthode de type filtrage, utilisant une matrice pour prédire ou limiter l'espace de recherche des occurrences d'une autre matrice. Cette mesure est discutée Section 2.4.

2.2 Calcul du score seuil et de la P-valeur

Comme nous l'avons évoqué plus haut, deux problèmes, complémentaires, sont relatifs au calcul de la P-valeur. Le premier consiste à calculer la P-valeur étant donnés une matrice et un score. Le second consiste à calculer le score correspondant à une P-valeur donnée pour une matrice. Nous allons maintenant discuter de la complexité du problème, qui est NP-dur, et d'un algorithme efficace et exact basé sur la programmation dynamique permettant de calculer la P-valeur. L'ensemble des résultats décrits dans cette section ont été publiés dans [Touzet et Varré 07].

Difficulté pratique du calcul de la P-valeur. Nous avons vu plus haut la façon dont une matrice score-position était obtenue. Le passage au logarithme fait que nous avons des matrices sur des réels. Bien souvent ces matrices sont tronquées. Suivant le nombre de chiffres après la virgule conservés (calculés grâce à un paramètre que nous appellerons *granularité* défini formellement Définition 2.1) la difficulté de calculer l'association P-valeur / score varie. En effet, plus la granularité est fine (la précision est grande), plus le nombre de scores différents est

grand. L'inconvénient d'utiliser directement la récurrence 2.1 pour le calcul de la P-valeur est que la complexité en temps et en espace dépend précisément du nombre de scores visités. Une illustration de la croissance des scores en fonction de la granularité est donnée Figure 2.5. D'autre

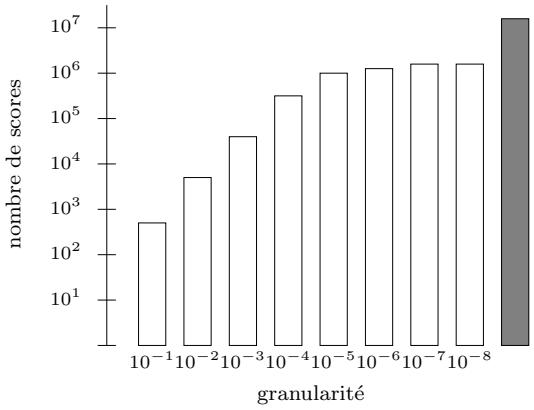


FIG. 2.5 – Évolution du nombre de scores en fonction de la précision.
La matrice M00041 de longueur 12 de la base de données JASPAR [95] a été utilisée. La granularité représente le nombre de chiffres après la virgule. Sont reportés le nombre de scores différents pour chaque granularité entre 1 et 8 chiffres après la virgule. La barre grise représente le nombre de mots différents, soit 4^{12}. Le nombre de scores croît assez vite, tout en restant un ordre de grandeur en deçà du nombre de mots.

part, lorsque l'on arrondit les scores d'une matrice, on introduit nécessairement une erreur dans le calcul de la P-valeur. Le tableau 2.1 donne le pourcentage de matrices pour lesquelles il y a une erreur sur le calcul de la P-valeur lorsque les scores sont tronqués. Il peut donc être intéressant

granularité	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
% matrices avec erreur	76	55	30	15	7	1	1

TAB. 2.1 – Erreur en pourcentage de matrices sur l'estimation du score seuil pour une P-valeur de 10^{-3} sur l'ensemble des matrices de Jaspar.

de disposer d'un algorithme capable de calculer avec exactitude les associations score/P-valeur pour des précisions assez grandes pour des objets tels que les matrices score-position. Toutefois, dans le cas des matrices représentant des sites de fixation, calculer avec une grande précision ne fait pas trop de sens vu le faible nombre d'échantillons au départ. Nous verrons que notre algorithme reste efficace avec un faible nombre de chiffres après la virgule.

2.2.1 Complexité du calcul de la P-valeur

Rappelons qu'un score s de $\mathbb{R}$ est *accessible* s'il existe un mot u de Σ^m tel que $\texttt{Score}(u, M) = s$. Nous nous intéressons au problème de décision suivant :

SCORE ACCESSIBLE

Instance : *Un alphabet fini Σ, une matrice M de longueur m dont les coefficients sont des entiers positifs ou nuls, un entier positif ou nul t*
Question : *Existe-t-il un mot u de Σ^m tel que* $\texttt{Score}(u, M) = t$ *?*

Théorème 2.1 Score accessible *est NP-dur.*

La preuve du théorème se fait par réduction du problème de décision Subset Sum, qui est un problème pseudo-polynomial NP-complet [50].

Subset Sum

Instance : *Un ensemble d'entiers positifs* $A = \{a_0, \ldots, a_n\}$ *et un entier positif* s
Question : *Existe-t-il un sous-ensemble* A' *de* A *tel que la somme des éléments de* A' *est égale à* s *?*

Il existe une réduction polynomiale directe du problème Subset Sum au problème Score accessible. L'idée consiste à construire une matrice de scores de longueur le nombre d'éléments de A sur un alphabet à deux lettres. Une des lettres prend comme score les valeurs de A et l'autre 0. Il est immédiat de voir la bijection entre les deux problèmes. Une illustration est donnée Figure 2.6.

$A = \{2, 3, 7, 11\}$
$s = 10$

Trouver un sous-ensemble d'éléments de A tel que leur somme vaut s.

$\Longleftrightarrow$

	0	1	2	3
a	2	3	7	11
b	0	0	0	0

$s = 10$
Existe-t-il $u \in \{a, b\}^4$ tel que $\texttt{Score}(u) = s$?

Fig. 2.6 – Réduction polynomiale de Subset Sum en score accessible.

Notons qu'une preuve a été également produite en 2007 par Zhang et *al.* [127] mais celle-ci est plus complexe et utilise le problème de calcul de graines espacées.

2.2.2 Calcul efficace de la P-valeur

Le calcul d'un couple score/P-valeur nécessite d'énumérer les scores accessibles de la matrice. Pour améliorer ce calcul, il faut donc avoir moins de scores à énumérer. D'ailleurs, il est évident que plus la P-valeur sera faible, et plus le calcul du score associé se terminera rapidement (en suivant le schéma de calcul donné page 40). Bien sûr, on peut mettre en œuvre, comme dans [10], des techniques de calcul permettant de limiter l'ensemble des cases visitées de la matrice de programmation dynamique. Cependant, on ne pourra pas éviter de faire certains calculs inutilement, ce qui peut devenir vite coûteux.

L'approche que nous avons proposée est de « regarder » la distribution de scores à des grains différents, allant du plus gros au plus fin. Des distributions de scores pour différentes valeurs de granularité sont données Figure 2.7. L'observation attentive des distributions de scores montre qu'il existe des sauts dans celle-ci. C'est-à-dire qu'il existe des plages de scores non accessibles. Nous montrerons que sous certaines conditions, si l'on observe une plage de scores $s, ..., s'$ non accessibles à une granularité g donnée, alors on continuera à l'observer à une granularité g' plus fine. Cela implique que le nombre de mots ayant un score supérieur à s est connu dès le grain g et qu'il est inutile de le recalculer pour une granularité plus fine. Une illustration est donnée Figure 2.8.

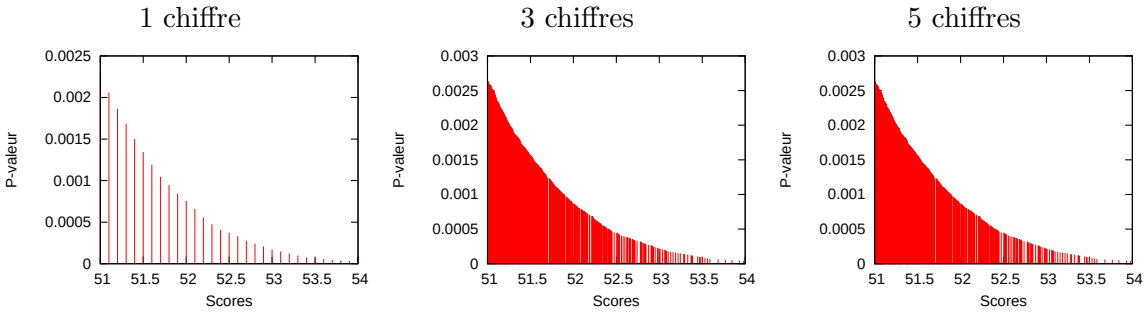


FIG. 2.7 – Queue de la distribution des scores pour la matrice M0041 pour différentes granularités.

Les graphiques laissent apparaître des discontinuités entre les scores accessibles. Certaines de celles-ci perdurent malgré un nombre de chiffres après la virgule plus grand.

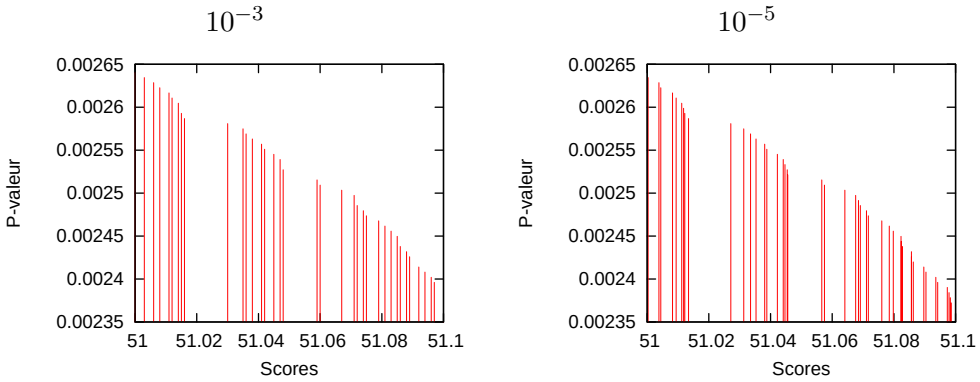


FIG. 2.8 – Illustration de plages de scores non accessibles.

On a effectué un zoom sur les distributions de score entre les scores 51 et 51.1. Dans ce cas, la plage sans score accessible autour de la valeur 51.02 est conservée et est suffisamment large entre une granularité à trois chiffres après la virgule et une granularité à cinq chiffres après la virgule. La valeur exacte de la P-valeur pour le score qui suit la valeur 51.02 est connue dès la granularité à trois chiffres après la virgule.

L'algorithme va donc procéder par étapes. Il calculera la distribution de scores à gros grain (généralement un chiffre après la virgule). On détectera alors s'il existe une plage de scores non accessibles permettant de connaître la P-valeur pour la queue de la distribution. Si elle existe, on itère le processus avec un grain plus fin et une borne supérieure pour le calcul de la distribution. Sinon, on itère sur la même borne supérieure.

Notons que les algorithmes que nous avons proposés utilisent comme modèle de fond un modèle de Bernouilli. Une extension de nos travaux a été proposée par da Fonseca et *al.* [37] pour les modèles de Markov.

Matrice arrondie. Nous l'avons compris, l'algorithme repose entièrement sur la détermination de plages de scores non accessibles suffisamment larges pour ne pas introduire d'erreur dans le calcul de la P-valeur. Nous définissons maintenant plus formellement ces plages.

Définition 2.1 *Soit M une matrice de scores à valeurs réelles de longueur m et soit ϵ un réel positif. On note M_ϵ la* matrice arrondie *déduite de M en arrondissant chaque valeur par ϵ :*

$$M_\epsilon(i,x) = \epsilon \left\lfloor \frac{M(i,x)}{\epsilon} \right\rfloor$$

ϵ appelé la granularité. *Étant donné ϵ, on définit E, l'*erreur maximale *induite par M_ϵ.*

$$E = \sum_{i=0}^{m-1} \max\{M(i,x) - M_\epsilon(i,x), x \in \Sigma\} \tag{2.2}$$

On exprime maintenant le fait que la matrice arrondie fournit toujours des scores plus petits que la matrice exacte et tel que la différence est bornée par E.

Lemme 2.1 *Soit M une matrice, ϵ la granularité, et E l'erreur maximale associée. Pour chaque mot u de Σ^m, nous avons $0 \leq \texttt{Score}(u,M) - \texttt{Score}(u,M_\epsilon) \leq E$.*

Le lemme suivant est au cœur des preuves de correction des algorithmes. Supposons avoir une matrice N, qui est une matrice arrondie de M pour laquelle on a trouvé dans la distribution de scores une plage de scores non accessibles (condition iii) et une autre matrice N', qui est une matrice arrondie de M, à plus gros grain, pour laquelle on a également trouvé dans la distribution de scores une plage de scores non accessibles (condition iv). Comme les plages de scores sont au moins aussi larges que l'erreur maximale, aucun mot de score inférieur à $\alpha - E$ pour N ne peut être de score plus grand que α dans M. Il en va de même pour N'. On sait donc que la P-valeur pour α est exacte lorsque calculée avec N, et que la P-valeur pour β est exacte lorsque calculée avec N'. Si on cherche à calculer la P-valeur pour α et que l'on dispose déjà de la P-valeur pour β, il reste à calculer la somme des $Q(N,t)$ sur l'intervalle $[\alpha,\beta[$.

Lemme 2.2 *Soit M, N et N' trois matrices de longueur m, E, E' deux nombres réels positifs ou nuls, α, β deux scores tels que $\alpha \leq \beta$, qui satisfont les hypothèses suivantes :*

(i) pour tout mot u de Σ^m, $\texttt{Score}(u,N) \leq \texttt{Score}(u,M) \leq \texttt{Score}(u,N) + E$,

(ii) pour tout mot u de Σ^m, $\texttt{Score}(u,N') \leq \texttt{Score}(u,N) \leq \texttt{Score}(u,M) \leq \texttt{Score}(u,N') + E'$,

(iii) $\texttt{P-valeur}(N, \alpha - E) = \texttt{P-valeur}(N, \alpha)$,

(iv) $\texttt{P-valeur}(N', \beta - E') = \texttt{P-valeur}(N', \beta)$,
alors

$$\sum_{\substack{\alpha \leq t < \beta \\ t \text{ accessible}}} Q(N,t) = \sum_{\substack{\alpha \leq t < \beta \\ t \text{ accessible}}} Q(M,t)$$

Cela fournit une condition suffisante pour que la distribution de scores Q calculée avec la matrice arrondie soit valide pour la matrice initiale. Autrement dit, il sera inutile de calculer avec des granularités plus fines.

Du score à la P-valeur. Le lemme 2.2 est utilisé comme décrit plus haut. Soit α le score pour lequel on souhaite calculer la P-valeur associée. On estime la distribution de scores Q de manière itérative. Pour une série de matrices arrondies à des granularités ϵ décroissantes, on calcule successivement $\texttt{P-valeur}(M_\epsilon, \alpha)$. Une condition d'arrêt, basée sur la détection d'une plage de score non accessible suffisamment large permet d'arrêter la suite des calculs dès que l'on est assuré que $\texttt{P-valeur}(M_\epsilon, \alpha) = \texttt{P-valeur}(M, \alpha)$. L'efficacité de la méthode se base sur le fait que la queue de la distribution n'est pas recalculée à chaque itération mais seulement une tranche. Le calcul de cette tranche est réalisé grâce à un algorithme (nommé ScoreDistribution) utilisant les principes d'élagage basé sur le lemme suivant, conséquence directe des propriétés de borne inférieure et supérieure :

Lemme 2.3 *Soit M une matrice de longueur m, soient α et β deux scores définissant un intervalle de scores pour lequel on veut calculer la distribution de scores Q. $Q(M[0..i], s)$ est* utile *si, et seulement si,*

$$\alpha - \texttt{ScoreMax}(M[i+1..m]) \leq s \leq \beta - \texttt{ScoreMin}(M[i+1..m])$$

L'algorithme de calcul de la P-valeur est donné Algorithme 3.

De la P-valeur au score. L'algorithme est très similaire à celui décrit pour calculer la P-valeur associée au score. La différence est qu'il faut savoir déterminer les bornes de l'intervalle pour lequel la distribution doit être calculée précisément. La réponse est apportée par ce lemme :

Lemme 2.4 *Soit M une matrice, ϵ une granularité et E l'erreur maximale associée. Étant donné P, $0 \leq P \leq 1$, nous avons*

$$\texttt{Seuil}(M_\epsilon, P) \leq \texttt{Seuil}(M, P) \leq \texttt{Seuil}(M_\epsilon, P) + E$$

Ainsi, il suffit de calculer sur un intervalle du double de l'erreur maximale engendrée par M_ϵ autour du seuil trouvé à cette granularité pour être certain de ne pas faire d'erreur à l'itération suivante. Pour les scores supérieurs à la borne supérieure de cet intervalle, nous utilisons un algorithme annexe (appelé PvaleurRapide), basé sur le lemme suivant :

Lemme 2.5 *Soit M une matrice de longueur m.*

$$\texttt{P-valeur}(M, \alpha) = \sum_{\substack{0 \leq i \leq m-1,\ x \in \Sigma,\ s < \alpha - \texttt{ScoreMin}(M[i..m-1]) \\ s + M(i,x) \geq \alpha - \texttt{ScoreMin}(M[i+1..m-1])}} Q(M[1..i-1], s) \times p(x)$$

L'algorithme de calcul du score est donné Algorithme 4.

Entrée : une matrice M de longueur m, un score seuil α
Sortie : P tel que $P = \texttt{P-valeur}(M, \alpha)$
  $\beta = \texttt{ScoreMax}(M) + 1$
  $P = 0$.
  choisir une granularité initiale ϵ et un $k > 1$
  repéter
    construire M_ϵ
    calculer l'erreur maximale E
    calculer $Q(M_\epsilon, t)$ pour chaque score accessible entre $\alpha - E..\beta$ en utilisant l'algorithme ScoreDistribution
    chercher s, la plus petite valeur telle que $\texttt{P-valeur}(M_\epsilon, s) = \texttt{P-valeur}(M_\epsilon, s - E)$
    si un tel s existe **alors**
      $P = P + \sum_{s \leq t < \beta} Q(M_\epsilon, t)$
      $\beta = s$
      $\epsilon = \frac{\epsilon}{k}$
    fin si
  jusqu'à $s = \alpha$

Algorithme 3 : ScorePvaleur : calcul de la P-valeur associée à un score.

Entrée : une matrice M de longueur m, P tel que $0 \leq P \leq 1$
Sortie : α tel que $\alpha = \texttt{Seuil}(M, P)$
  choisir une granularité initiale ϵ et $k > 1$
  construire M_ϵ
  calculer E, l'erreur maximale
  calculer $Q(M_\epsilon, t)$ pour chaque score accessible t plus grand que $\texttt{ScoreMin}(M_\epsilon)$ avec l'algorithme ScoreDistribution
  dans Q, chercher le plus grand score α tel que $\sum_{s \geq \alpha} Q(M_\epsilon, s) \geq P$
  tant que $\texttt{P-valeur}(M_\epsilon, \alpha - E) \neq \texttt{P-valeur}(M_\epsilon, \alpha)$ **faire**
    $\epsilon = \frac{\epsilon}{k}$
    construire M_ϵ
    calculer E, l'erreur maximale
    calculer $Q(M_\epsilon, t)$ pour chaque score t in $\alpha - E..\alpha + E$ avec l'algorithme ScoreDistribution
    calculer $\texttt{P-valeur}(M_\epsilon, \alpha + E)$ avec l'algorithme PvaleurRapide
    dans Q, chercher le plus grand score δ tel que
    $\sum_{\delta < s < \delta + E} Q(M_\epsilon, s) + \texttt{P-valeur}(M_\epsilon, \delta + E) \geq P$
    $\alpha = \delta$
  fin tant que

Algorithme 4 : PvaleurScore : calcul du score associé à une P-valeur.

Résultats expérimentaux. Les méthodes ont été implémentées dans un outil appelé TFM-PVALUE[5]. Les expériences menées pour évaluer les méthodes montrent une supériorité dans les temps de traitement même si on force l'algorithme à limiter sa granularité. Dans ce cas, nous forçons l'arrêt de la boucle pour une granularité maximale donnée. Les résultats, comparés à ceux trouvés avec l'algorithme de Beckstette et *al.* [10] sont donnés Figure 2.9. L'efficacité de la condition d'arrêt des algorithmes, basée sur la détection d'une plage de scores non accessibles suffisamment large, est validée expérimentalement puisque qu'il y a très peu de matrices pour lesquelles l'algorithme s'arrête sur des valeurs de granularité plus fines que nécessaires (voir Figure 2.10).

2.3 Découpage des matrices en sous-matrices

Dans cette partie, nous explorons le problème de la recherche d'une matrice dans une séquence basé sur l'idée de découpage des matrices en sous-matrices. L'idée est simple : accélérer le calcul en regroupant des colonnes consécutives de la matrice. On obtient alors des « tranches » de matrices. On pré-calcule les scores possibles pour chaque tranche et on obtient le score final d'un mot en sommant les scores des facteurs associés aux tranches. Cela est rendu possible grâce à la propriété d'additivité des scores d'une matrice score-position. Le gain en temps est obtenu grâce au stockage des scores possibles pour chaque tranche dans un index. On réduit alors le nombre d'additions et d'accès à la structure de stockage des scores (matrice ou index) au nombre de tranches. La figure 2.11 illustre le principe. Ces résultats ont été publiés dans [Liefooghe et *al.* 06].

2.3.1 Découpage d'une matrice

La taille de l'index constitué peut très vite devenir très grande en fonction de la largeur de la tranche. À l'extrême les meilleurs gains en temps seraient obtenus en constituant une seule tranche pour une matrice. Ce qui n'est pas réalisable si l'on veut contenir l'index dans un espace mémoire raisonnable. En effet, la longueur moyenne d'une matrice constatée dans les banques de donnés est $m = 14$; avec $\text{Card}(\Sigma) = 4$ il faut donc un index de taille 4^{14} soit de l'ordre d'un téra-octet[6].

La solution que nous avons mise en œuvre est à mi-chemin entre le calcul complet de tous les scores (une tranche de la taille de la matrice) et aucun pré-calcul (une tranche par colonne de matrice).

Définition 2.2 *Soit M une matrice score-position de longueur m. Un découpage de M en ℓ tranches est défini par une suite croissante de $\ell + 1$ positions $i_0, \ldots, i_\ell$ telles que $i_0 = 0$ et $i_\ell = m$. Les valeurs $i_0, \ldots, i_{\ell-1}$ correspondent chacune à la première position de chaque tranche. La* structure d'index *pour M et $i_0, \ldots, i_{\ell-1}$, notée $S_{M,i_0 \ldots i_\ell}$ est un tableau à deux entrées. Pour un indice de tranche k de $[0..\ell - 1]$ et un mot u de $\Sigma^{i_{k+1}-i_k}$, $S_{M,i_0 \ldots i_\ell}(k, u)$ est défini par*

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

[5] Disponible sur `http://bioinfo.lifl.fr/TFM`

[6] Chaque score étant stocké sur 4 octets

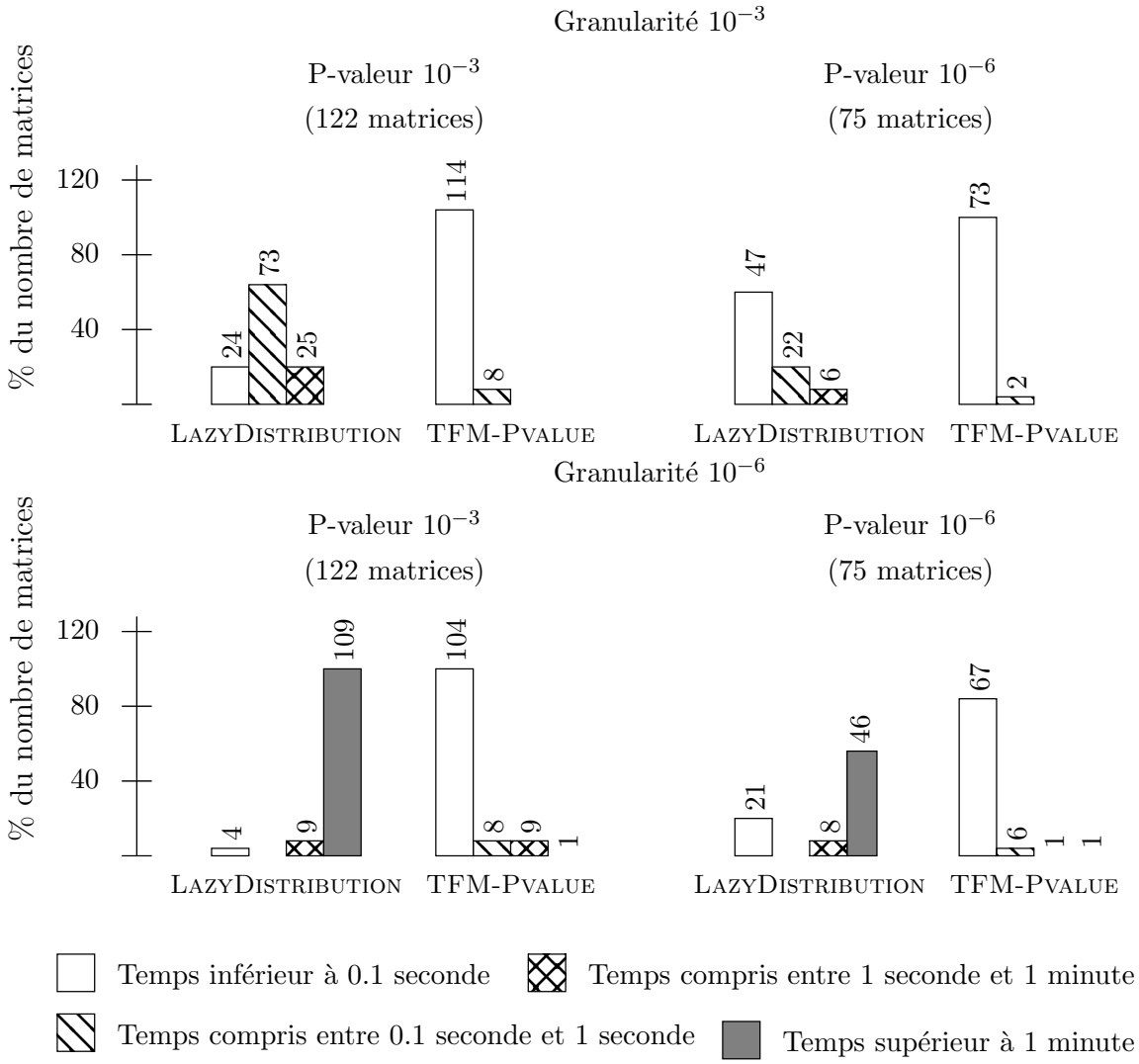


FIG. 2.9 – Répartition du nombre de matrices en fonction du temps, pour une granularité bornée à 10^{-3} en haut et 10^{-6} en bas.

L'algorithme a été modifié pour s'arrêter à la granularité donnée (ou avant si la condition d'arrêt est remplie). Les tests ont été effectués sur les 123 matrices de Jaspar [95]. Les temps de calcul ont été comparés avec l'algorithme LazyDistribution [10] mettant en œuvre les optimisations achevant les temps de calcul les plus rapides. Les temps obtenus par notre méthode sont toujours meilleurs. Cela montre que la technique d'algorithme affinant les résultats est capable, même pour une précision donnée, d'obtenir le même résultat plus rapidement. Notre méthode s'avère également beaucoup plus efficace dès lors que la P-valeur est grande (ce qui induit un nombre de scores à visiter plus important dans la matrice de programmation dynamique).

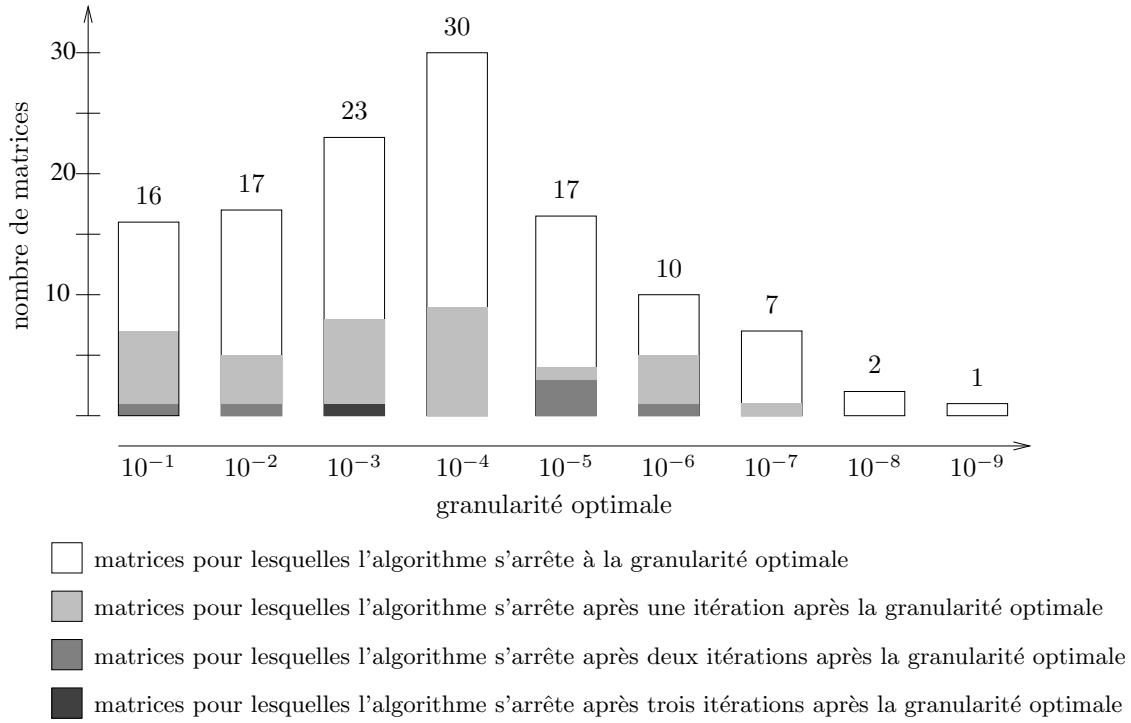


FIG. 2.10 – Qualité de la condition d'arrêt.

Pour une P-valeur de 10^{-3}, nous avons calculé les scores seuil des matrices de Jaspar [95]. Nous avons reporté, en fonction de la granularité optimale, le nombre de matrices pour lesquelles les itérations de l'algorithme s'arrêtaient à cette granularité. La granularité optimale est celle à partir de laquelle le bon score seuil est sélectionné (évalué en énumérant exactement les mots supérieurs à ce score seuil). La condition d'arrêt est assez efficace puisque pour plus de 90% des matrices l'algorithme s'arrête à la granularité optimale ou après une itération.

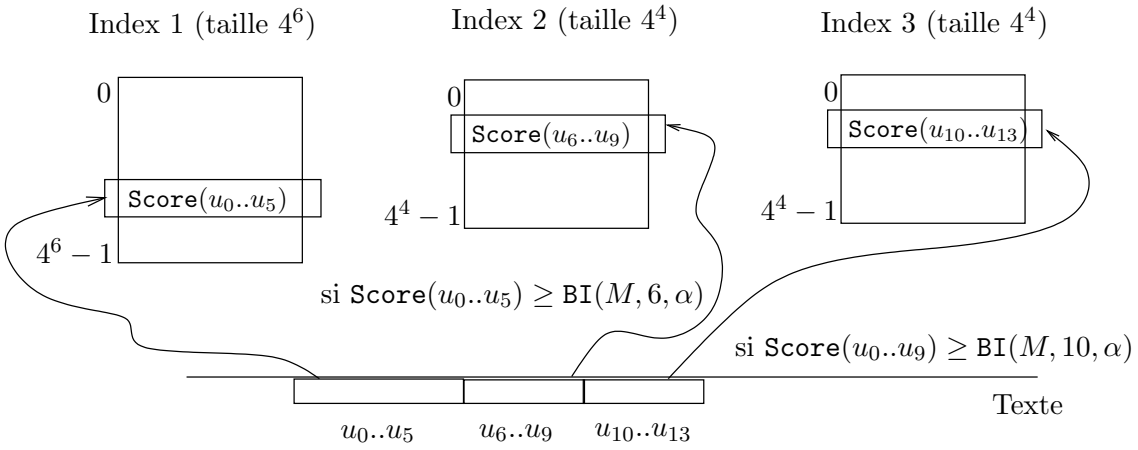


FIG. 2.11 – Exemple de découpage d'une matrice en trois tranches.

Dans la version combinée avec l'élagage, l'accès aux tranches 2 et 3 se fait sous condition.

Le score final d'un mot u de Σ^m pour la matrice M s'obtient par la somme entre les tranches :

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

Comme nous le signalions plus haut, le nombre d'accès pour un mot donné est égal au nombre de tranches du découpage ℓ, au lieu de m. C'est en fait l'espace mémoire disponible qui impose une limite inférieure sur le nombre de tranches, à travers la taille de l'index.

Sans autre optimisation que celle décrite ci-dessus, la structure d'index optimale est trouvée en remplissant l'espace mémoire au maximum pour un nombre minimum de tranches. Comme on a intérêt à limiter le nombre de tranches, une stratégie gloutonne peut être appliquée, sous la condition que les temps d'accès aux index des tranches soient bien en temps constant.

Découpage optimal. Cependant, comme nous l'avons vu en début de chapitre, la méthode d'élagage est extrêmement efficace. Rien ne nous empêche de combiner cette stratégie à celle du découpage. Ainsi, on pourra arrêter le calcul du score d'un mot à une tranche dès qu'il n'est plus possible d'atteindre le score seuil. Cela est illustré par les conditions sur l'accès aux tables Figure 2.11.

Le critère d'optimalité du découpage est alors modifié. On aura tout intérêt à ce que les bornes des tranches coïncident avec les positions où l'élagage est le plus fréquent. Inutile en effet de faire une tranche avec les deux premières positions s'il faut systématiquement lire une troisième lettre (au moins) avant de pouvoir décider d'une occurrence. Nous avons défini le découpage optimal comme celui minimisant le nombre moyen d'accès aux index (qui est le même que le nombre moyen d'additions). Afin de calculer ce nombre, nous allons, pour chaque position de la matrice, considérer l'ensemble des mots *candidats* qui nécessitent le calcul de leur score jusqu'à au moins cette position pour déterminer s'ils sont occurrence ou non.

Définition 2.3 *Soient M une matrice de longueur m, i une position de M et α un score seuil. On définit l'ensemble des mots candidats de M, i et α, noté* $\texttt{Cand}(M, i, \alpha)$*, par :*

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

Lemme 2.6 $\mathbb{P}(\texttt{Cand}(M, i, \alpha)) = \texttt{P-valeur}(M[0..i-1], \texttt{BI}(M, i-1, \alpha))$

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

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

$$\phi(0, e) = \begin{cases} 1, & \text{si } e \geq 0 \\ +\infty, & \text{si } e < 0 \end{cases} \tag{2.3}$$

$$\phi(j, e) = \min_{0 \leq i < j} \phi(i, e - |\Sigma|^{j-i}) + \mathbb{P}(\texttt{Cand}(M, i, \alpha)) \tag{2.4}$$

Lemme 2.7 *Soit M une matrice de longueur m. Le découpage optimal de M pour un espace mémoire maximal e est obtenu grâce au calcul de $\phi(m, e)$.*

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

Résultats expérimentaux. Si l'accélération que l'on peut attendre de l'optimisation proposée est, théoriquement, toujours efficace, les problèmes liés à l'accès des éléments de la table d'index ont un retentissement sur les performances obtenues en pratique. Par exemple, en découpant avec des tranches de taille un le temps de recherche est plus lent qu'avec l'algorithme direct. L'évolution du temps de recherche en fonction de découpages réguliers (taille de tranche égale) est donné Figure 2.12.

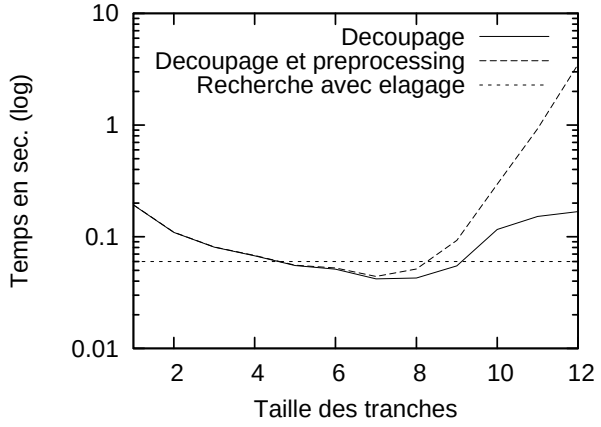


FIG. 2.12 – Efficacité du découpage en tranches.
La matrice M00041, de longueur 12, a été utilisée avec une P-valeur de 10^{-3} sur une séquence aléatoire de longueur 10^6. Temps comparés de la méthode d'élagage, avec des découpages uniformes pour toutes les tailles de tranches. Nous remarquons que le découpage permet d'accélérer la recherche pour certaines largeurs de tranches, même en considérant le prétraitement. Les résultats montrent également que le découpage donnant les temps de recherche les plus rapides ne sont pas nécessairement ceux avec le moins de tranches, sûrement à cause des temps d'accès à l'index. L'algorithme avec élagage coupe en moyenne à la position 6 pour la matrice M00041, ce qui est cohérent avec le meilleur temps obtenu.

2.3.2 Découpage d'un ensemble de matrices

Si le découpage d'une matrice en sous-matrices est efficace alors l'application du même principe à la recherche d'un ensemble de matrices devrait également être accéléré en mettant en œuvre le même principe. Bien sûr toutes les matrices ne vont pas avoir le même découpage optimal. Nous allons réécrire une formule de récurrence minimisant le nombre moyen d'accès à la table d'index pour un ensemble de matrices.

Le problème que nous adressons dans cette section est :

Recherche des occurrences d'un ensemble de matrices

Soit $\mathcal{M} = \{M_1, ..., M_k\}$ un ensemble de matrices, $\alpha_1, ..., \alpha_k$ les score seuils respectifs, et soit T un texte, trouver toutes les occurrences de chaque matrice $M_i, 1 \leq k \leq p$ de $\mathcal{M}$ dans T.

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

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

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

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

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

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

$$\phi(0, e) = \begin{cases} \text{Card}(\mathcal{M}) & \text{si } e \geq 0 \\ +\infty & \text{sinon} \end{cases} \tag{2.5}$$

$$\phi(j, e) = \min_{0 \leq i < j} \{\phi(i, e - |\Sigma|^{j-i} + \sum_{M_k \in \mathcal{M}} \mathbb{P}(\texttt{Cand}(M_k, i, \alpha_k)) \tag{2.6}$$

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

Résultats expérimentaux. L'étude menée sur les matrices de Jaspar [95] et Transfac [124] a révélé une accélération en temps $\times 10$ grâce à l'index, comparé à la recherche successive avec toutes les matrices et l'algorithme d'élagage. Le découpage optimal pour ces matrices et le graphique des temps est donné Figure 2.13.

2.4 Algorithmes de filtrage

Une technique souvent mise en œuvre pour accélérer la recherche d'occurrences consiste à procéder en plusieurs étapes successives de recherche, en affinant les résultats. Ainsi on peut imaginer que pour rechercher l'occurrence d'un mot dans un texte, on commence par rechercher les occurrences d'un sous-mot puis, sur ces positions, on vérifie la présence effective du mot.

Nous avons utilisé un principe similaire pour la recherche multiple de matrices. L'idée est que si un ensemble de matrices sont semblables, alors il y a toutes les chances pour que l'occurrence

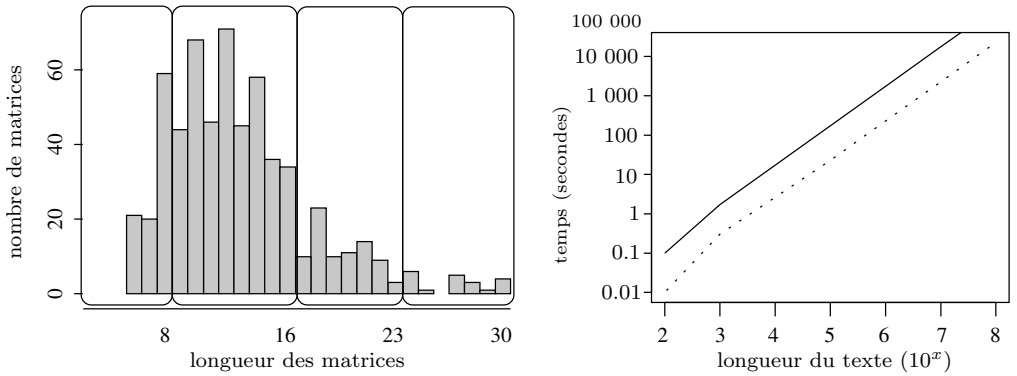


FIG. 2.13 – Découpage et accélération pour l'index multi-matrices.
À gauche, l'histogramme représente la répartition des longueurs des matrices utilisées dans l'expérience (602 au total). Le découpage optimal pour ce jeu de matrices est composé de 4 tranches : 1..8,9..16,17..23 et 24..30. Le nombre de matrices dans chaque tranche est respectivement 602, 502, 100 et 20. Le nombre moyen d'accès aux tranches est respectivement 602, 216, 18 et 1. À droite sont représentés le temps de calcul de l'algorithme avec élagage (en trait plein) avec celui de l'algorithme utilisant l'index construit sur le découpage donné avant (en pointillés). La mémoire occupée par l'index est ici 256 méga-octets.

de l'une d'entre elles implique les occurrences pour tout ou partie des autres. De même pour les non-occurrences. Nous définissons une notion de similarité entre matrices qui permet de mesurer cette corrélation d'occurrences. Ces résultats ont été publiés dans [Liefooghe et *al.* 06].

Pizzi et *al.* [89] ont également proposé un algorithme de type filtrage mais en construisant un automate de Aho-Corasick [2] sur un préfixe de la matrice (pour cause d'espace mémoire) et en calculant les occurrences sur les positions pré-sélectionnées.

2.4.1 Similarité entre deux matrices

Pour évaluer la similitude entre matrices, les articles [103, 65, 51] proposent différentes méthodes de mesure de la corrélation entre les coefficients des matrices. Nous présentons ici deux nouvelles manières de comparer deux matrices, qui travaillent directement sur les occurrences des matrices, et non sur leurs coefficients. C'est plus pertinent quand il s'agit de faire de la recherche de motifs.

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

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

Définition 2.4 *Soit M et N deux matrices de même longueur m. Le* seuil d'incertitude *entre*

M et N, noté $U(M,N)$, est défini comme le plus petit entier naturel positif n tel que

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

Lemme 2.8

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

La seconde approche consiste à affiner la définition de seuil d'incertitude, en mesurant la probabilité de l'ensemble des mots pour lesquelles deux matrices donnent des résultats divergents. Pour cela, nous définissons les notions de *faux positif* et de *faux négatif* pour un mot vis-à-vis de deux matrices.

Définition 2.5 *Soit M et N deux matrices et α et β leurs scores seuils respectifs. Soit u un mot de Σ^*.*

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

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

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

Le lemme ci-dessous donne une manière exacte de calculer les quantités FP et FN grâce à une formule de récurrence. La mise en œuvre se fait naturellement par programmation dynamique.

Lemme 2.9 *Soient M et N deux matrices de longueur m, α un score seuil pour M et β un score seuil pour N*
$FP(M,N,\alpha,\beta) = l(m,\alpha,\beta)/4^m$,
où la fonction ℓ est définie récursivement par

$$\begin{array}{rcl} l(0,s_1,s_2) & = & \begin{cases} 1 & \text{si } s_1 \leq 0 \text{ et } s_2 > 0 \\ 0 & \text{sinon} \end{cases} \\ l(i,s_1,s_2) & = & \sum_{x \in \Sigma} l(i-1, s_1 - M(x,i), s_2 - N(x,i)) \end{array}$$

$FN(M,N,\alpha_1,\alpha_2) = l'(m,\alpha_1,\alpha_2)/4^m$,
où la fonction ℓ' est définie récursivement par

$$\begin{array}{rcl} l'(0,s_1,s_2) & = & \begin{cases} 1 & \text{si } s_1 > 0 \text{ et } s_2 \leq 0 \\ 0 & \text{sinon} \end{cases} \\ l'(i,s_1,s_2) & = & \sum_{x \in \Sigma} l'(i-1, s_1 - M(x,i), s_2 - N(x,i)) \end{array}$$

La complexité en temps de calcul de ces probabilités dépend du nombre de scores accessibles comme pour le calcul de la P-valeur, c'est-à-dire $\mathcal{O}(m|\Sigma|SS')$ si S et S' sont le nombre de scores accessibles respectifs des matrices M et N.

Ces notions de faux positifs et faux négatifs ont été reprises par Pape et *al.* [85] en proposant une mesure basée sur celles-ci, calculant la probabilité de chevauchement.

2.4.2 Application à l'accélération de la recherche d'occurrences

Les deux définitions données ci-dessus aboutissent à deux algorithmes fonctionnant sur l'approche décrite brièvement en début de section. Plus précisément, étant donné un ensemble de matrices, nous commençons par construire une clusterisation de celles-ci. Ensuite, pour chaque groupe de matrices, nous définissons une matrice représentante qui sera utilisée par l'algorithme de recherche proprement dit. À partir des occurrences de cette représentante, nous déduisons les occurrences des matrices du groupe. Deux stratégies sont alors envisageables. Soit nous acceptons des erreurs et nous décidons que chaque matrice représentante décide des occurrences du groupe sans calcul supplémentaire. Nous pouvons alors contrôler le taux d'erreur. Soit nous calculons le score de chaque matrice du groupe pour les positions d'occurrence de la représentante.

Algorithme approché. L'algorithme ici est très simple puisque nous considérons comme occurrence toute occurrence de la matrice représentante (idem pour les non-occurrences). Remarquons que le calcul des occurrences de la matrice représentante peut être accéléré par les autres techniques mises en œuvre dans ce chapitre. Plus précisément, dans la cas de la recherche multiple, nous pouvons obtenir c clusters pour lesquels la recherche des occurrences des c matrices représentantes peut-être accéléré grâce au découpage de matrices présenté en section 2.3.

Nous allons contrôler le type d'erreur grâce au choix de la matrice représentante. Si par exemple nous voulons exclure les faux négatifs alors, étant donné un groupe de matrices $\mathcal{M} = \{M_1, \ldots, M_k\}$ de scores seuil respectifs $\alpha_1, \ldots, \alpha_k$, nous pouvons construire une matrice où chaque coefficient $M(i,x) = \max_k M_k(i,x)$ et choisir comme score seuil $\alpha = \min_k \alpha_k$.

Algorithme exact. Dans l'approche par filtrage sans perte nous allons utiliser le score d'incertitude pour limiter les recalculs. Pour chaque groupe de matrices $\mathcal{M} = \{M_1, \ldots, M_k\}$ de scores seuil respectifs $\alpha_1, \ldots, \alpha_k$, la matrice représentante calculée N sert de filtre. La définition du score d'incertitude permet de n'avoir à recalculer le score pour chaque $M_i, 1 \leq i \leq k$ que si le score de N possède une différence d'au plus $U(M_i, N)$ avec le score seuil α_i. L'algorithme est donné Figure 5.

Entrée : un texte T, un ensemble de matrices $\{(M_1, \alpha_1) \ldots (M_k, \alpha_k)\}$
Sortie : toutes les occurrences de $M_1, \ldots, M_k$ sur T
définir une matrice filtre N pour $\{(M_1, \alpha_1), \ldots, (M_k, \alpha_k)\}$
pour j allant de 1 à k **faire**
calculer $U(M_j, N)$
pour chaque position i du texte T **faire**
calculer $s \leftarrow \texttt{Score}(S, i, N)$
pour j allant de 1 à k **faire**
si $s > \alpha_j - U(M_j, N)$ **alors**
calculer $\texttt{Score}(S, i, M_j)$
fin si
fin pour
fin pour
fin pour

Algorithme 5 : Algorithme exact de filtrage

En terme de complexité, cela ne change rien dans le pire des cas mais, dans le meilleur des cas, cela permet d'avoir une complexité indépendante du nombre de matrices.

2.5 Extension des algorithmes de recherche classiques

Après ces méthodes traitant principalement de la recherche simultanée de plusieurs matrices, on peut se poser la question de trouver des algorithmes efficaces pour une et unique matrice, tout en restant dans l'idée de faire un pré-calcul sur la matrice et non sur la séquence. À notre connaissance l'algorithme le plus efficace procédant à un pré-calcul sur la séquence est celui proposé dans [10]. Quant aux méthodes basées sur un pré-calcul de la matrice, il n'en existait pas de plus efficace que celle utilisant le principe de l'élagage jusqu'en 2006. Nous avons pour notre part étendu les algorithmes basés sur le calcul d'une table de décalage (à la Knuth-Morris-Pratt [67]), permettant de ne pas tester certaines positions du texte. Nous espérons voir ces résultats publiés prochainement dans [Liefooghe et *al.* , 09].

S'ajoutèrent en 2007 plusieurs méthodes dont nous allons présenter le principe avant de discuter de notre proposition.

Pizzi et *al.* [89] proposèrent en premier lieu de calculer un automate de Aho-Corasick [2] (qui n'est autre qu'une représentation de l'ensemble des tables de Knuth-Morris-Pratt pour un ensemble de mots) sur l'ensemble des mots de score supérieur au score seuil. L'inconvénient est qu'il est nécessaire pour construire cet automate d'énumérer l'ensemble de ces mots. Nous avons déjà vu en section 2.2 les limites que cela pouvait avoir. Si l'on omet ce problème et l'espace mémoire nécessaire au stockage d'un tel automate, il ne fait aucun doute que cet algorithme est extrêmement rapide. Un algorithme dit de super-alphabet est également présenté dans cet article. Les auteurs proposent de réécrire la matrice sur des mots de longueur $q > 1$ au lieu de la matrice sur les lettres. Cette extension peut être vue comme un cas particulier de ce que nous avons décrit dans les sections précédentes.

Salmela et *al.* [94] ont proposé une extension de l'algorithme Shift-Add [7]. On utilise le principe classique de ce type d'algorithme, c'est-à-dire faire glisser un vecteur binaire le long du texte et propager un bit indiquant une occurrence. Dans le cas du Shift-Add, ce sont quelques bits qui sont propagés au lieu d'un unique, permettant de trouver les occurrences d'un motif exact à k erreurs près (le nombre de bits à propager est $\lceil \log(k+1) \rceil$). Dans l'adaptation proposée, on propage un nombre plus important de bits puisque c'est le score du préfixe qui est propagé ; il faut alors $\lceil \log \alpha \rceil + 1$ bits pour un score seuil α. L'efficacité de l'algorithme réside dans le fait que le score seuil n'est pas trop grand (en terme de représentation binaire). La complexité de l'algorithme est alors $\mathcal{O}(n\lceil \frac{m(\lceil \log \alpha \rceil + 1)}{w} \rceil)$ où w est la taille d'un mot machine.

2.5.1 Extension de Morris-Pratt

L'algorithme de recherche de mots exacts de Morris-Pratt [79] introduit l'idée que la lecture d'un préfixe du motif se concluant par un échec, c'est-à-dire une non-occurrence, donne une indication dans le texte sur la prochaine position susceptible d'être une occurrence. On appelera *tentative* une position du texte où l'on teste une occurrence. L'algorithme est basé sur le calcul du *bord* d'un mot qui représente le chevauchement du préfixe et du suffixe d'un mot.

Un facteur u d'un mot v est qualifié de *propre* si $u \neq v$. Un *bord* d'un mot non vide u est un

facteur propre de u qui est à la fois préfixe et suffixe de u. On note $\texttt{Bord}(u)$ la longueur du plus long bord de u. Le plus long bord peut aider dans un algorithme de recherche de mots exacts. Supposons rechercher un mot u dont le préfixe v de longueur ℓ a été lu à une position p du texte et tel que la lettre suivant ce mot v n'est pas celle à la position $p + \ell$. On sait donc qu'il n'y a pas d'occurrence à la position p du texte du mot u. Si maintenant on sait que le plus long bord de u est de longueur ℓ', alors nécessairement, pour toutes les positions entre p et $p + \ell - \ell'$ on ne peut avoir d'occurrence de u.

Cette propriété permet de calculer pour chaque position du motif recherché la prochaine position potentielle du motif dans le texte. La *table des correspondances partielles de Morris-Pratt* d'un mot u de longueur m, notée $\texttt{mpNext}$, est un vecteur de taille $m+1$ dont les éléments sont définis par :

$$\begin{aligned} \texttt{mpNext}[0] &= -1 \\ \texttt{mpNext}[i] &= \texttt{Bord}(u_0 \dots u_{i-1}), \forall i \in [1, \dots, m] \end{aligned}$$

L'utilisation de cette table dans l'algorithme de recherche permet des décalages potentiels de plus d'une position mais surtout cela permet de ne pas recommencer la comparaison des lettres du préfixe déjà lues avec le texte. Puisque l'on a un bord, nécessairement le préfixe fonctionne. On a donc un algorithme dont la complexité en temps est $\mathcal{O}(m + n)$ où m est la longueur du mot recherché et n la longueur du texte. Le pré-calcul de la table se fait simplement en temps linéaire.

Extension de la notion de bord aux matrices. Étendre la notion de bord aux matrices se fait assez directement. Au lieu que le bord soit défini sur la correspondance exacte entre un préfixe et un suffixe d'un mot, il est défini sur le notion de score. Il y a correspondance si le score d'un préfixe et le score d'un suffixe sont plus grand qu'un score seuil.

Définition 2.6 *Soit M une matrice de longueur m, et soit α un score seuil. Un* bord de M et de α *est un mot u de longueur $\ell \geq 1$ tel qu'il existe deux mots v et w de longueur $m - \ell$ satisfaisant*

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

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

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

Lemme 2.10 *Soit M une matrice de longueur m, et soit α un score seuil. Le mot u de longueur ℓ est un bord de M et de α si, et seulement si, u satisfait les deux conditions suivantes :*

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

Extension de l'algorithme de recherche aux matrices. La phase de recherche des occurrences d'une matrice score-position dans un texte en utilisant une table des décalages de la matrice est tout à fait similaire à celle dans le cas des mots exacts. L'algorithme consiste à

calculer les scores successifs des préfixes de la matrice à une position du texte. Dès que l'on se trouve dans un cas d'échec à une position i de la matrice (le score est inférieur au score $\texttt{BI}(M, i, \alpha)$), la table nous fournit en position i la prochaine position du texte à tester. Une illustration est donnée Figure 2.14. Nous donnons donc la définition de la table `mpNext` dans le cadre des matrices puis l'algorithme de recherche.

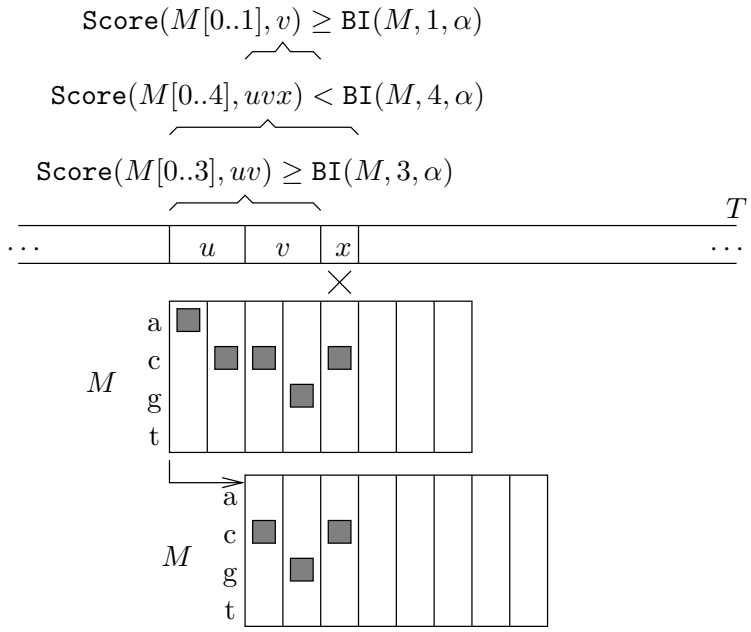


FIG. 2.14 – Illustration de la règle de décalage de Morris-Pratt dans le cas des matrices. À la position courante, on a lu le mot uv du texte dont le score est suffisamment grand pour qu'il soit potentiellement préfixe d'une occurrence. Par contre à la lecture de la lettre suivante, on sait que uvx ne peut pas être préfixe d'une occurrence. Le plus long bord de la matrice $M[0..3]$ est de longueur 2 ce qui implique que l'on peut décaler la matrice de $4-2$ positions vers la droite. On sait alors que le score de v sur le préfixe de la matrice de longueur 2 est supérieur au score minimum à atteindre en position 2. Par contre, on ne connaît pas ce score.

Définition 2.7 *Soit M une matrice de longueur m, α un score seuil, La table des correspondances partielles de M et α, notée* `mpNext`*, est un vecteur de taille $m+1$ dont les éléments sont définis par*

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

Lemme 2.11 *Soit M une matrice de longueur m, T un texte de longueur n, p une position dans le texte comprise entre 0 et $n-m-1$, α un score seuil, et i une position dans la matrice ($0 \leq i \leq m-1$). Si la tentative p se termine à la position i de M, alors on connaît par avance le résultat des tentatives $p+1 \ldots p+(i-\texttt{mpNext}[i])$.*

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

Dans les algorithmes de Morris-Pratt et de Knuth-Morris-Pratt, le décalage possède la propriété supplémentaire que, comme le préfixe (le bord) est exactement le texte, il est inutile de relire les lettres du texte correspondant à ce préfixe lors du test de l'occurrence suivante. Dans le cas des matrices score-position, on sait également que le préfixe est potentiellement une occurrence mais pour conclure il est nécessaire de connaître le score de ce préfixe, ce dont on ne dispose malheureusement pas. Il est donc nécessaire de relire les lettres du texte pour calculer le score. L'algorithme est donné Algorithme 6.

Entrée : T un texte de longueur n, M la matrice de longueur m, BI le tableau des scores minimum intermédiaire, `mpNext` la table des correspondances partielles
Sortie : les occurrences de M dans T
$j \leftarrow 0$
tant que $j < n - m + 1$ **faire**
  $i \leftarrow 0$
  $s \leftarrow 0$
  tant que $i < m$ and $s \geq BI[i]$ **faire**
    $s \leftarrow s + M(i, T[j + i])$
    $i++$
  fin tant que
  si $i = m$ **alors**
    print j
    $j++$
  sinon
    $j \leftarrow i -$ `mpNext`$[i]$
  fin si
fin tant que

Algorithme 6 : Recherche des occurrences avec une table de décalage

Calcul de la table `mpNext`. Le calcul de la table `mpNext` est plus délicat que dans le cas des mots exacts. Nous allons, comme pour la calcul de la P-valeur, raisonner sur les scores accessibles et non sur les mots. Nous introduisons la définition d'un prédicat $c(M, p, \alpha)$ qui permet de tester l'existence d'un bord de longueur p pour M. Le calcul de ce prédicat permet alors d'obtenir la valeur de `mpNext` pour chaque position p.

Définition 2.8 *On définit le prédicat $c(M, p, \alpha)$ vrai si, et seulement si, il existe un bord de longueur $m - p$ pour M et α.*

On introduit maintenant le prédicat b qui est utilisée dans le calcul de c. Intuitivement b permet de tester si un mot possède des scores donnés sur la partie préfixe et suffixe d'une matrice alors que c permet de tester l'existence de mots ayant des scores sur la partie préfixe et suffixe supérieurs à des seuils donnés.

Définition 2.9 *Soit M une matrice de longueur m, et p, i deux positions de M ($0 \leq p \leq i \leq m - 1$). On définit la fonction b :*

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

Le lemme suivant implique qu'il est suffisant de calculer b pour obtenir c.

Lemme 2.12

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

Reste maintenant à calculer b de manière efficace. Le lemme suivant montre que pour chaque p, $b(M,p,i,\beta,\delta)$ peut être calculé par programmation dynamique et ce de manière paresseuse à partir de valeurs plus petites de i, β et δ.

Lemme 2.13

$$\begin{cases} b(M,p,i,0,0) &=& 1, \textit{ si } i < p \\ b(M,p,i,\beta,\delta) &=& 0, \textit{ si } i < p, \textit{ } \beta \neq 0 \textit{ ou } \delta \neq 0 \\ b(M,p,i,\beta,\delta) &=& \vee_{x\in\Sigma}\;\; b(M,p,i-1,\beta - M(i,x), \delta - M(i-p,x)) \textit{ sinon} \end{cases}$$

Il est également intéressant de remarquer que pour le calcul de c il n'est pas nécessaire de disposer de l'ensemble des valeurs de b.

Lemme 2.14 *$c(M,p,\alpha)$ est vrai si, et seulement si, pour chaque valeur i de p à $m-1$, il existe β et δ tels que :*

1. *$b(M,p,i,\beta,\delta) = 1$, et*
2. *$\beta \geq \texttt{BI}(M,i,\alpha) - \texttt{ScoreMax}(M[0..p-1])$, et*
3. *$\delta \geq \texttt{BI}(M, i-p, \alpha)$.*

Nous pouvons maintenant donner un algorithme de calcul de c (Algorithme 7). Étant donnés i et p, soit $\mathcal{S}_i^p$ l'ensemble des paires de scores (β,δ) satisfaisant les conditions du lemme 2.14 : $c(M,p,\alpha)$ est vrai si, et seulement si, $\mathcal{S}_{m-1}^p$ n'est pas vide. Le calcul de `mpNext` s'obtient naturellement à partir de c : $\texttt{mpNext}[i] = i - p$ où p est la plus petite valeur telle que $c(M,p,\alpha)$ est vrai.

Utilisation pour le calcul des occurrences chevauchantes. Nous ne détaillons pas cette partie mais nous pouvons, en calculant la probabilité de l'ensemble des mots u vérifiant les conditions de $b(M,p,i,\beta,\delta)$, calculer la probabilité d'observer deux occurrences chevauchantes de la matrice M. De la même manière, on peut calculer la moyenne, la variance et la covariance.

2.5.2 Extension de Knuth-Morris-Pratt

La différence entre l'algorithme de Morris-Pratt et l'algorithme de Knuth-Morris-Pratt [67] est que si on veut éviter un échec immédiat après un décalage donné par la table `mpNext`, le caractère suivant le préfixe dans le motif doit être différent de la lettre à la position courante. La notion de bord est donc étendue à la notion de bord étiqueté qui prend en compte cette contrainte supplémentaire.

Soit $u_0 \dots u_{p-1}$ un préfixe de u, considérons le mot v de longueur ℓ bord de $u_0 \dots u_{p-1}$, v est un *bord étiqueté* si la lettre u_p est différente de u_ℓ. $\texttt{kmpNext}[i]$ est défini comme la longueur du plus long bord étiqueté de $u_0 \dots u_{i-1}$. L'expression de cette condition sur les matrices est un peu plus complexe puisque l'on veut assurer que le score avec la lettre suivante sera supérieur au score minimum à atteindre à la position courante.

Entrée : une matrice M de longueur m, une position p de M, un score seuil α
Sortie : $c(M, p, \alpha)$
  $\mathcal{S}^p_{p-1} \leftarrow \{(0,0)\}$
  pour i allant de $p-1$ à $m-2$ **faire**
    $\mathcal{S}^p_{i+1} \leftarrow \emptyset$
    pour tout (β, δ) in $\mathcal{S}^p_i$ **faire**
      pour tout $x \in \Sigma$ **faire**
        si $\beta + M(i,x) \geq \texttt{BI}(M,i,\alpha) - \texttt{ScoreMax}(M[0..p-1])$ **alors**
          si $\delta \geq \texttt{BI}(M, i-p, \alpha)$ { *Definition 2.9* } **alors**
            $\mathcal{S}^p_{i+1} \leftarrow \mathcal{S}^p_{i+1} \cup \{(\beta + M(i,x), \delta + M(i-p,x))\}$
          fin si
        fin si
      fin pour
    fin pour
  fin pour
  $c(M,p,\alpha) \leftarrow \mathcal{S}^p_{m-1} \neq \emptyset$ { *Lemme 2.14* }

Algorithme 7 : Calcul des valeurs de c

Définition 2.10 *Soit M une matrice, i une position dans M et α un score seuil. u est un* bord étiqueté *pour M, i et α, si u est un bord de $M[0..i]$ et $\texttt{BI}(M,i,\alpha)$, et il existe une lettre x de Σ tel que*

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

où ℓ est la longueur de u.

Nous noterons $\texttt{BordEtiqueté}(M,i,\alpha)$ la longueur du plus long bord étiqueté de M, i et α. Par convention, $\texttt{BordEtiqueté}(M,i,\alpha)$ vaut -1 si un tel mot u n'existe pas. La table `kmpNext` est définie ainsi :

$$\begin{aligned} \texttt{kmpNext}[0] &= -1 \\ \texttt{kmpNext}[i] &= \texttt{BordEtiqueté}(M, i-1, \alpha), \quad 1 \leq i \leq m \end{aligned}$$

Nous introduisons un nouveau prédicat afin de calculer `kmpNext`, qui est une extension de c pour prendre en compte la notion de bord étiqueté.

Définition 2.11 *On définit $c'(M,p,i,\alpha)$ à vrai si, et seulement si, il existe un bord étiqueté de longueur $i-p$ pour M, $i-1$ et α.*

$\texttt{kmpNext}[i]$ vaut $i-p$, avec p la plus petite valeur telle que $c'(M,p,i-1,\alpha)$ est vrai. La calcul effectif de c' est réalisé grâce au lemme suivant, montrant que b peut être utilisé pour calculer c'.

Lemme 2.15 *$c'(M,j,i,\alpha)$ est vrai si, et seulement si, il existe deux scores β, δ et une lettre x de Σ tels que*

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

2.5.3 Table de décalage à Σ entrées

Nous terminons ces extensions de Morris-Pratt et Knuth-Morris-Pratt en proposant une variation où le décalage calculé dépend de la lettre (du texte) sur laquelle l'échec a eu lieu. Au lieu de calculer le plus long bord étiqueté, nous calculons, pour chaque lettre x de l'alphabet, le plus long bord qui n'est pas suivi par x.

Définition 2.12 *Soit M une matrice, i une position de M, α un score seuil et x une lettre de Σ. u est un x-*bord étiqueté *pour M, i, et α, si u est un bord de $M[0..i]$ et $\texttt{BI}(M, i, \alpha)$ tel que*

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

où ℓ est la longueur de u.

On notera $\texttt{BordEtiqueté}_\Sigma(M, i, \alpha, x)$ la longueur du plus long x-bord étiqueté pour M, i, et α. La table de décalage est donc maintenant à deux entrées :

$$\begin{aligned} \texttt{kmpNext}_\Sigma[0][x] &= -1 \\ \texttt{kmpNext}_\Sigma[i][x] &= \texttt{BordEtiqueté}_\Sigma(M, i, \alpha, x), \qquad 1 \leq i \leq m \end{aligned}$$

On remarquera que $\texttt{kmpNext}[i] = \min_{x \in \Sigma} \texttt{kmpNext}_\Sigma[i][x]$ pour toute position i d'une matrice M. On retrouve comme précédemment la définition d'un prédicat et son calcul à partir de b.

Définition 2.13 *On définit le prédicat $c''(M, p, i, \alpha, x)$ vrai si, et seulement si, il existe un x-bord étiqueté de longueur $i-p$ pour M, $i-1$, α et x.*

$\texttt{kmpNext}_\Sigma[i][x]$ vaut $i-p$, où p est la plus petite valeur telle que $c''(M, p, i-1, \alpha, x)$ est vrai.

Lemme 2.16 *$c''(M, p, i, \alpha, x)$ est vrai si, et seulement si, il existe deux scores β, δ tels que :*

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

2.5.4 Résultats expérimentaux

Nous avons testé la capacité de l'algorithme utilisant la table $\texttt{kmpNext}_\Sigma$ à effectuer des sauts dans le texte, c'est-à-dire à ne pas tester certaines positions. Les résultats sont reportés Figure 2.15. On constate que pour une grande majorité des cas, l'algorithme permet efectivement des sauts. Deux exemples typiques de matrices pour laquelle a) les décalages sont efficaces (MA0090) et b) les décalages sont inefficaces (MA0115) sont donnés Figure 2.16. Bien sûr on observe que la matrice MA0115 est auto-chevauchante. Les résultats en terme de temps de calcul sont plus décevants car très proches de ceux obtenus juste avec l'élagage. Il y a deux raisons à cela. D'une part le re-calcul du score du préfixe après décalage est responsable de la perte du temps gagné par le non test d'une position. D'autre part si la position n'a pas à être testée c'est qu'elle amène à un échec, ce qui est donc favorable à l'élagage.

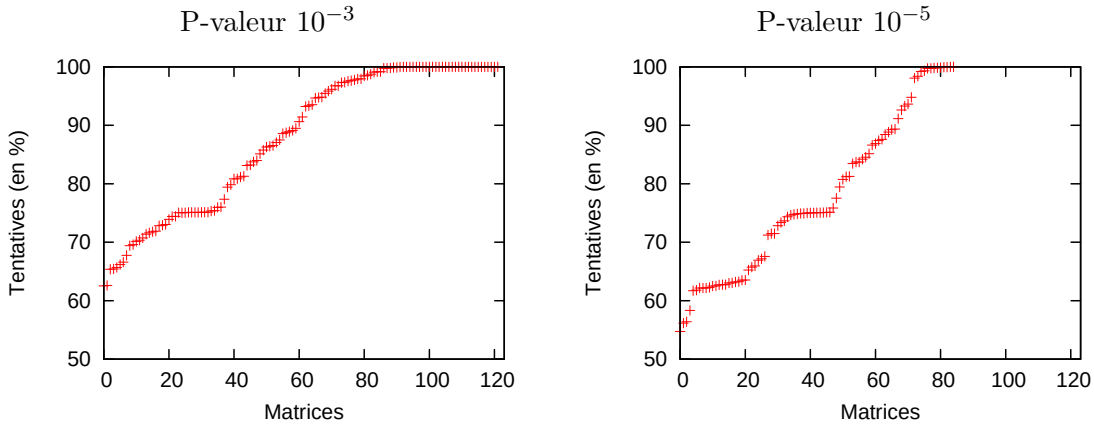


FIG. 2.15 – Pourcentage des tentatives.

Nous avons représenté pour des P-valeurs de 10^{-3} et 10^{-5} le pourcentage de tentatives réalisées avec l'algorithme utilisant la table $\texttt{kmpNext}_{\Sigma}$ sur l'ensemble des matrices de Jaspar pour une séquence de longueur 10^6. Pour la P-valeur 10^{-5} les matrices trop courtes pour avoir une occurrence ont été supprimées. On constate un gain d'au moins 20% pour environ 30% des matrices avec une P-valeur 10^{-3} et pour environ 50% des matrices avec une P-valeur 10^{-5}.

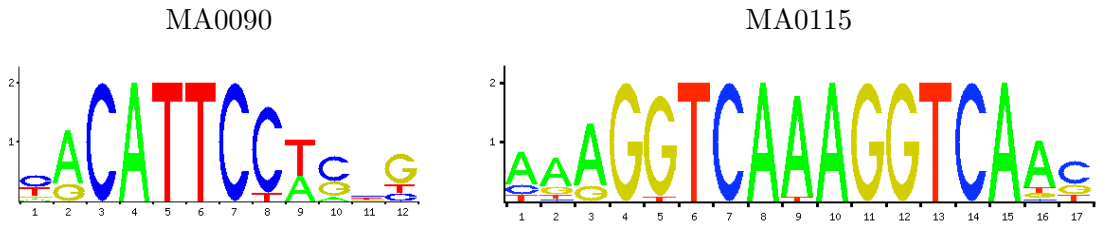


FIG. 2.16 – Matrices MA0090 et MA0115 de Jaspar.

Avec la matrice MA0090, on gagne 17% des positions alors qu'avec la matrice MA0115 on ne gagne que 0,2%. Le caractère auto-chevauchant de la matrice MA0115 est responsable des mauvais décalages obtenus.

2.5.5 Re-visitons le découpage de matrices

Il est également possible de combiner le découpage d'une matrice présenté Section 2.3 avec les extensions présentées ici.

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

Comme pour le découpage présenté Section 2.3, on définit des ensembles de mots semblables aux ensembles des mots candidats (définition 2.3). Le calcul de la probabilité de ces ensembles est réalisé à partir de la matrice de programmation dynamique permettant le calcul de la probabilité des ensembles de mots candidats. On calcule ensuite le saut moyen associé à une tranche grâce à cette probabilité et on en déduit l'expression d'un découpage optimal minimisant le saut moyen total.

2.6 Perspectives

Les développements effectués dans cette partie se sont focalisés sur des algorithmes de recherche de motifs modélisés par des matrices score-position. Nous avons développé plusieurs stratégies, la plus efficace d'un point de vue pratique étant sans doute celle du découpage des matrices.

D'un point de vue théorique il est intéressant de poursuivre dans cette voie car l'adaptation d'autres techniques issues de la recherche de motifs peut fournir des extensions intéressantes au cas des matrices. L'intérêt pour ce type de motif ne se limite d'ailleurs pas à l'étude des sites de fixation de facteurs de transcription ou même des objets biologiques et mérite d'être étudié plus largement. Nous avons vu que l'algorithme de Knuth-Morris-Pratt permettait un gain limité. Pizzi et *al.* [89] ont montré que la construction d'un automate de Aho-Corasick (qui est une généralisation de l'algorithme de Morris-Pratt) permettait de bonnes accélérations au détriment d'un espace mémoire nécessaire trop grand, empêchant de considérer de longues matrices. Comment trouver un intermédiaire ? Nous pensons que des approches type Boyer-Moore [28] ou des approches utilisant des structures telles qu'un automate des suffixes devraient être tentées.

D'un point de vue pratique, il ne semble pas que l'on puisse attendre des accélérations importantes, au moins pour les matrices représentant des sites de fixation de facteurs de transcription. Dans d'autres cas, motif moins dégénéré, motif avec redondance, on doit pourvoir obtenir également des accélérations en pratique. Par contre, ces accélérations en pratique peuvent être obtenues d'une autre façon, en utilisant le calcul parallèle. Nous avons développé le problème de recherche de plusieurs matrices sur GPU[7] dans le cadre d'un projet de master 1 avec Mathieu Giraud. Les résultats sont prometteurs et c'est une piste que nous développons actuellement.

[7] Graphical Processor Unit

Une poursuite naturelle des travaux engagées autour de la recherche de sites de fixations est de s'orienter vers l'identification et la localisation de modules. On désigne par module un ensemble de facteurs de transcriptions qui coopèrent. La localisation de modules consiste donc à rechercher des régions où il existe des occurrences de différents sites de fixations. La plupart du temps l'ordre et la distance qui séparent les sites sont importants. Cependant, si l'on dispose d'une desciption d'un module, peu de méthodes existent pour retrouver ceux-ci à l'échelle d'un génome eucaryote [102].

Il est enfin nécessaire de se confronter à la réalité de l'ADN qui ne peut, lorsqu'il s'agit de phénomènes complexes de liaison, être restreint à un texte écrit sur un alphabet. Il a par exemple été démontré que les régions de régulation actives étaient le plus souvent dépourvues de nucléosomes [71]. Les nucléosomes sont des structures responsables de la compaction de l'ADN. On commence à avoir des modèles du positionnenment des nucléosomes le long d'une séquence [104, 116, 52]. Cela ouvre la voie à la création de méthodes combinant la recherche de motifs avec des matrices score-position (tel que nous l'avons développée ici) et la recherche de motifs dans des textes pondérés [63] afin de mieux prédire les occurrences potentielles des sites de fixation des facteurs de transcription [81].

Bibliographie

[Darracq et *al.* 08] Aude Darracq, Jean-Stéphane Varré, Adeline Courseaux, Laurence Maréchal-Drouard, and Pascal Touzet. Evolution of the mitochondrial genome in beet. A comparative genomic study at the intra-specific level, Poster in XX International Congress of Genetics, 2008.

[Figeac et Varré 04] Martin Figeac and Jean-Stéphane Varré. Sorting by Reversals with Common Intervals. In *Proceedings of the 4th Workshop on Algorithms In Bioinformatics*, volume 3240 of *Lectures Notes in Computer Science*, pages 26–37, 2004.

[Liefooghe et *al.* 06] Aude Liefooghe, Hélène Touzet, and Jean-Stéphane Varré. Large Scale Matching for Position Weight Matrices. In *Proceedings of the 17th Symposium on Combinatorial Pattern Matching*, volume 4009 of *Lecture Notes in Computer Science*, pages 401–412, 2006.

[Liefooghe et *al.* , 09] Aude Liefooghe, Hélène Touzet, and Jean-Stéphane Varré. Self-overlapping occurrences and Knuth-Morris-Pratt algorithm for weighted matching. Submitted to LATA'09.

[Touzet et Varré 07] Hélène Touzet and Jean-Stéphane Varré. Efficient and accurate P-value computation for Position Weight Matrices. *Algorithms in Molecular Biology*, 2(15), 2007.

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

[2] A. Aho and M. Corasick. Efficient string matching : An aid to bibliographic search. *Communications of the ACM*, 18 :333–340, 1975.

[3] Y. Ajana, J. Lefebvre, E. Tillier, and N. El-Mabrouk. Exploring the Set of All Minimal Sequences of Reversals-An Application to Test the Replication-Directed Reversal Hypothesis. In *Proceedings of WABI'01*, volume 2452 of *Lecture Notes in Computer Science*, pages 300–315, 2002.

[4] J. O. Allen, C. M. Fauron, P. Minx, L. Roark, S. Oddiraju, G. N. Lin, L. Meyer, H. Sun, K. Kim, C. Wang, F. Du, D. Xu, M. Gibson, J. Cifrese, S. W. Clifton, and K. J. Newton. Comparisons among two fertile and three male-sterile mitochondrial genomes of maize. *Genetics*, 177(2) :1173–1192, 2007.

[5] A. Amir, L. G. asieniec, and R. Shalom. Improved approximate common interval. *Information Processing Letters*, 103 :142–149, 2007.

[6] D. Bader, B. Moret, and M. Yan. A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 8(5) :483–491, 2001.

[7] R. Baeza-Yates and G. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10) :77–82, 1992.

[8] V. Bafna and P. Pevzner. Sorting by Reversals : Genome Rearrangements in Plant Organelles and Evolutionary History of X Chromosome. *Molecular Biology and Evolution*, 12(2) :239–245, 1995.

[9] M.-P. Béal, A. Bergeron, S. Corteel, and M. Raffinot. An algorithmic view of gene teams. *Theoritical Computer Science*, 320(2-3) :395–418, 2004.

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

[11] R. Beigel. Finding Maximum Independent Sets in Sparse and General Graphs. *Proceedings of SODA'99*, pages 856–857, 1999.

[12] S. Bérard, A. Bergeron, C. Chauve, and C. Paul. Perfect Sorting by Reversals Is Not Always Difficult. *IEEE/ACM transactions on computational biology and bioinformatics*, 4(1) :4–16, 2005.

[13] S. Bérard, C. Chauve, and C. Paul. A more efficient algorithm for perfect sorting by reversals. *Information Processing Letters*, 2008(106) :90–95, 2008.

[14] A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve. Reconstructing Ancestral Gene Orders Using Conserved Intervals. In *Proceedings of WABI'04*, volume 3240 of *Lecture Notes in Computer Science*, pages 14–25, 2004.

[15] A. Bergeron, C. Chauve, F. de Montgolfier, and M. Raffinot. Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. In *Proceedings of ESA'05*, volume 3669 of *Lecture Notes in Computer Science*, pages 779–790, 2005.

[16] A. Bergeron, C. Chauve, T. Hartman, and K. St-Onge. On the properties of sequences of reversals that sort a signed permutation. In *Proceedings of JOBIM'02*, pages 99–108, 2002.

[17] A. Bergeron, S. Heber, and J. Stoye. Common intervals and sorting by reversals : a marriage of necessity. *Bioinformatics*, 18 Suppl 2 :S54–S63, 2002.

[18] A. Bergeron, J. Mixtacki, and J. Stoye. Reversal Distance without Hurdles and Fortresses. In *Proceedings of CPM'04*, volume 3240 of *Lecture Notes in Computer Science*, pages 388–399, 2004.

[19] A. Bergeron and J. Stoye. On the Similarity of Sets of Permutations and Its Applications to Genome Comparison. *Journal of Computational Biology*, 13(7) :68–79, 2003.

[20] M. Bernt, D. Merkle, and M. Middendorf. A Fast and Exact Algorithm for the Perfect Reversal Median Problem. In *Proceedings of ISBRA'07*, volume 4463 of *Lecture Notes in Computer Science*, pages 305–316, 2007.

[21] M. Bernt, D. Merkle, K. Ramsch, G. Fritzsch, M. Perseke, D. Bernhard, M. Schlegel, P. Stadler, and M. Middendorf. CREx : inferring genomic rearrangements based on common intervals. *Bioinformatics*, 23(21) :2957–2958, 2007.

[22] M. Blanchette, T. Kunisawa, and D. Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2) :193–203, 1999.

[23] G. Blin, A. Chateau, C. Chauve, and Y. Gingras. Inferring Positional Homologs with Common Intervals of Sequences. In *Proceedings of RECOMB Comparative Genomics 2005*, volume 4205 of *Lecture Notes in Computer Science*, pages 24–38, 2006.

[24] G. Blin, C. Chauve, G. Fertin, R. Rizzi, and S. Vialette. Comparing Genomes with Duplications : a Computational Complexity Point of View. *IEEE/ACM transactions on computational biology and bioinformatics*, 4(4) :523–534, 2007.

[25] G. Blin and R. Rizzi. Conserved Interval Distance Computation Between Non-trivial Genomes. In *Proceedings of COCOON'05*, volume 3595 of *Lecture Notes in Computer Science*, pages 22–31, 2005.

[26] J. Boore and W. Brown. Big trees from little genomes : mitochondrial gene order as a phylogenetic tool. *Current Opition in Genetics and Development*, 8 :668–674, 1998.

[27] G. Bourque, P. A. Pevzner, and G. Tesler. Reconstructing the genomic architecture of ancestral mammals : lessons from human, mouse, and rat genomes. *Genome Research*, 14(4) :507–516, 2004.

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

[29] S. Bérard, A. Bergeron, and C. Chauve. Conservation of combinatorial structures in evolution scenarios. In *Proceedings of RECOMB Comparative Genomics 2004*, volume 3388 of *Lecture Notes in Computer Science*, pages 1–14, 2005.

[30] A. Caprara. Sorting by reversals is difficult. In *Proceedings of RECOMB'97*, pages 75–83. ACM, 1997.

[31] K. Cartharius, K. Frech, K. Grote, and B. Klocke. MatInspector and beyond : promoter analysis based on transcription factor binding sites. *Bioinformatics*, 21(13) :2933–2942, 2005.

[32] K. Chaudhuri, K. Chen, R. Mihaescu, and S. Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of SODA'06*. ACM, 2006.

[33] C. Chauve, Y. Diekmann, S. Heber, and J. Mixtacki. On Common Intervals with Errors. Technical report, Bielefeld University, 2006.

[34] D. Che, G. Li, F. Mao, H. Wu, and Y. Xu. Detecting uber-operons in prokaryotic genomes. *Nucleic Acids Research*, 34(8) :2418–2427, 2006.

[35] D. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4) :165–169, 1996.

[36] J. Claverie and S. Audic. The statistical significance of nucleotide position-weight matrix matches. *Bioinformatics*, 12(5) :431–439, 1996.

[37] P. da Fonseca, C. Gautier, K. Guimaraes, and M. Sagot. Efficient representation and P-value computation for high-order Markov motifs. In *Bioinformatics*, volume 24 of *Proceedings of ECCB'08*, pages 160–166, 2008.

[38] M. Defrance. *Algorithmes pour l'analyse de régions régulatrices dans le génome d'eucaryotes supérieurs*. PhD thesis, Université des Sciences et Technologies de Lille, 2006.

[39] G. Didier, T. Schmidt, J. Stoye, and D. Tsur. Character sets of strings. *Journal of Discrete Algorithms*, 5(2007) :330–340, 2006.

[40] Y. Diekmann, M.-F. Sagot, and E. Tannier. Evolution under reversals : parsimony and conservation of common intervals. *IEEE/ACM transactions on computational biology and bioinformatics*, 4(2) :301–9, 2007.

[41] T. Dobzhansky and A. Sturtevant. Inversions in the chromosomes of *Drosophila Pseudoobscura*. *Genetics*, 23(1) :29–64, 1938.

[42] B. Dorohonceanu and C. G. Nevill-Manning. Accelerating Protein Classification Using Suffix Trees. In *Proceedings of ISMB'00*, pages 128–133. AAAI Press, 2000.

[43] N. El-Mabrouk. Sorting signed permutations by reversals and insertions/deletions of contiguous segments. *Journal of Discrete Algorithms*, 1(1) :105–122, 2001.

[44] R. Eres, G. Landau, and L. Parida. A Combinatorial Approach to Automatic Discovery of Cluster-Patterns. In *Proceedings of WABI'03*, volume 2812 of *Lecture Notes in Computer Science*, pages 139–150, 2003.

[45] C. Fauron, M. Havlik, and R. Brettell. The mitochondrial genome organization of a maize fertile cmst revertant line is generated through recombination between two sets of repeats. *Genetics*, 124 :423–428, 1990.

[46] S. Fénart, P. Touzet, J.-F. Arnaud, and J. Cuguen. Emergence of gynodioecy in wild beet (Beta vulgaris ssp. maritima L.) : a genealogical approach using chloroplastic nucleotide sequences. *Proceedings of the Royal Society*, 273(1592) :1391–8, 2006.

[47] M. Figeac. *Étude de l'ordre des gènes : clusters de gènes et algorithmique des réarrangements*. PhD thesis, Université des Sciences et Technologies de Lille, 2005.

[48] M. Figeac and J.-S. Varré. Sorting by Reversals with Common Intervals. In *Proceedings of WABI'04*, volume 3240 of *Lecture Notes in Computer Science*, 2004.

[49] V. Freschi and A. Bogliolo. Using sequence compression to speedup probabilistic profile matching. *Bioinformatics*, 21(10) :2225–9, 2005.

[50] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[51] S. Gupta, J. Stamatoyannopoulos, T. Bailey, and W. Noble. Quantifying similarity between motifs. *Genome Biology*, 8(2), 2007.

[52] S. Gupta1, J. Dennis, R. E. Thurman, R. Kingston, J. A. Stamatoyannopoulos, and W. S. Noble. Predicting Human Nucleosome Occupancy from Primary Sequence. *PLoS Computational Biology*, 4(8), 2008.

[53] S. Halford and J. Marko. How do site-specific DNA-binding proteins find their targets ? *Nucleic Acids Research*, 32, 2004.

[54] S. Hannenhalli, C. Chappey, E. Koonin, and P. Pevzner. Genome Sequence Comparison and Scenarios for Gene Rearrangements : A Test Case. *Genomics*, 30 :299–311, 1995.

[55] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip : polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of STOC'95*. ACM, 1995.

[56] S. Hannenhalli and P. Pevzner. To cut... or not to cut (applications of comparative physical maps in molecular evolution). In *Proceedings of SODA'96*, pages 304–313. Society for Industrial and Applied Mathematics, 1996.

[57] X. He and M. Goldwasser. Identifying Conserved Gene Clusters in the Presence of Homology Families. *Journal of Computational Biology*, 12(6) :638–656, 2005.

[58] S. Heber and J. Stoye. Algorithms for Finding Gene Clusters. In *Proceedings of WABI'01*, volume 2149 of *Lecture Notes in Computer Science*, pages 252–263, 2001.

[59] S. Heber and J. Stoye. Finding All Common Intervals of k Permutations. In *Proceedings of CPM'01*, volume 2089 of *Lecture Notes in Computer Science*, pages 207–218, 2001.

[60] G. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7-8) :563–577, 1999.

[61] R. Hoberman and D. Durand. The Incompatible Desiderata of Gene Cluster Properties. In *Proceedings of RECOMB Comparative Genomics 2005*, volume 3678 of *Lecture Notes in Computer Science*, pages 73–87, 2005.

[62] M. D. Hoon, S. Imoto, K. Kobayashi, N. Ogasawara, and S. Miyano. Predicting the operon structure of *Bacillus subtilis* using operon length, intergene distance, and gene expression information. In *Proceedings of PSB'04*, pages 276–287, 2004.

[63] C. S. Iliopoulos, C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. The Weighted Suffix Tree : An Efficient Data Structure for Handling Molecular Weighted Sequences and its Applications. *Fundamenta Informaticae*, 71 :259–277, 2006.

[64] A. Kel, E. Gossling, I. Reuter, and E. Cheremushkin. MATCHTM : a tool for searching transcription factor binding sites in DNA sequences. *Nucleic Acids Research*, 31(13) :3576–3579, 2003.

[65] S. M. Kielbasa, D. Gonze, and H. Herzel. Measuring similarities between transcription factor binding sites. *BMC Bioinformatics*, 6 :237, 2005.

[66] S. Kim, J. Choi, and J. Yang. Gene Teams with Relaxed Proximity Constraint. In *Proceedings of IEEE Computational Systems Bioinformatics Conference*, pages 44–55. IEEE Computer Society, 2005.

[67] D. Knuth, J. Morris, and V. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2) :323–350, 1977.

[68] T. Kubo, S. Nishizawa, A. Sugawara, N. Itchoda, A. Estiati, and T. Mikami. The complete nucleotide sequence of the mitochondrial genome of sugar beet (Beta vulgaris L.) reveals a novel gene for tRNA(Cys)(GCA). *Nucleic Acids Research*, 28(13) :2571–6, 2000.

[69] G. Landau, L. Parida, and O. Weimann. Gene Proximity Analysis across Whole Genomes via PQ Trees. *Journal of Computational Biology*, 12(10) :1289–1306, 2005.

[70] W. Lathe, B. Snel, and P. Bork. Gene context conservation of a higher order than operons. *Trends in Biochemical Sciences*, 25(10) :474–9, 2000.

[71] C. Lee, Y. Shibata, B. Rao, B. Strahl, and J. Lieb. Evidence for nucleosome depletion at active regulatory regions genome-wide. *Nature Genetics*, 36(8) :900–905, 2004.

[72] J. Lefebvre, E. Tillier, A. Maler, and N. El-Mabrouk. The distribution of inversion lengths in bacteria. In *Proceedings of the RECOMB Comparative Genomics 2004*, volume 3388 of *Lecture Notes in Computer Science*, pages 97–108, 2004.

[73] M. Lercher, T. Blumenthal, and L. D. Hurst. Coexpression of Neighboring Genes in *Caenorhabditis Elegans* Is Mostly Due to Operons and Duplicate Genes. *Genome Research*, 13 :238–243, 2003.

[74] A. Liefooghe. *Matrices score-position, algorithmes et propriétés.* PhD thesis, Université des Sciences et Technologies de Lille, 2008.

[75] N. Luc, J.-L. Risler, A. Bergeron, and M. Raffinot. Gene teams : a new formalization of gene clusters for comparative genomics. *Computational biology and chemistry*, 27(1) :59–67, 2002.

[76] G. Moreno-Hagelsieb and J. Collado-Vides. A powerful non-homology method for the prediction of operons in prokaryotes. *Bioinformatics*, 18 :S329–S336, 2002.

[77] B. Moret, A. Siepel, J. Tang, and T. Liu. Inversion Medians Outperform Breakpoint Medians in Phylogeny Reconstruction from Gene-Order Data. In *Proceedings of WABI'02*, volume 2452 of *Lecture Notes in Computer Science*, pages 521–536, 2002.

[78] B. Moret, J. Tang, L. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *Journal of Computer and System Sciences*, 65(3) :508–525, 2002.

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

[80] R. Mueller and J. Boore. Molecular Mechanisms of Extensive Mitochondrial Gene Rearrangement in Plethodontid Salamanders. *Molecular Biology and Evolution*, 22(10) :2104–2112, 2005.

[81] L. Narlikar, R. Gordân, and A. Hartemink. A Nucleosome-Guided Map of Transcription Factor Binding Sites in Yeast. *PLoS Computational Biology*, 3(11), 2007.

[82] H. Ogata, S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa. KEGG : Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 27(1) :29–34, 1999.

[83] R. Overbeek, M. Fonstein, M. D'Souza, and G. Pusch. The use of gene clusters to infer functional coupling. *Proceedings of the National Academy of Sciences*, 96(6) :2896–2901, 1999.

[84] J. Palmer. Intraspecific Variation and Multicircularity in Brassica Mitochondrial DNAs. *Genetics*, 118 :341–351, 1988.

[85] U. Pape, S. Rahmann, and M. Vingron. Natural Similarity Measures between Position Frequency Matrices with an Application to Clustering. *Bioinformatics*, 24(3) :350–357, 2008.

[86] S. Pasek, A. Bergeron, J.-L. Risler, A. Louis, E. Ollivier, and M. Raffinot. Identification of genomic features using microsyntenies of domains : Domain teams. *Genome Research*, 15 :867–874, 2005.

[87] M. Perseke, G. Fritzsch, K. Ramsch, M. Bernt, D. Merkle, M. Middendorf, D. Bernhard, P. F. Stadler, and M. Schlegel. Evolution of mitochondrial gene orders in echinoderms. *Molecular Phylogenetics and Evolution*, 47(2008) :855–864, 2007.

[88] R. Pinter and S. Skiena. Genomic Sorting with Length-Weighted Reversals. *Genome Informatics*, 13 :103–111, 2002.

[89] C. Pizzi, P. Rastas, and E. Ukkonen. Fast search algorithms for position specific scoring matrices. In *Proceedings of BIRD'07*, volume 4414 of *Lecture Notes in Computer Science*, pages 239–250, 2007.

[90] S. Rahmann. Dynamic Programming Algorithms for Two Statistical Problems in Computational Biology. In *Proceedings of WABI'03*, volume 2812 of *Lecture Notes in Computer Science*, pages 151–164, 2003.

[91] I. Rogozin, K. Makarova, J. Murvai, and E. Czabarka. Connected gene neighborhoods in prokaryotic genomes. *Nucleic Acids Research*, 30(10) :2212–2223, 2002.

[92] M. Sagot and E. Tannier. Perfect Sorting by Reversals. In *Proceedings of COCOON'05*, volume 3595 of *Lecture Notes in Computer Science*, pages 42–51, 2005.

[93] H. Salgado, G. Moreno-Hagelsieb, T. Smith, and J. Collado-Vides. Operons in *Escherichia coli* : genomic analyses and predictions. *Proceedings of the National Academy of Sciences*, 97(12) :6652–6659, 2000.

[94] L. Salmela and J. Tarhio. Algorithms for Weighted Matching. In *Proceedings of SPIRE'07*, volume 4726 of *Lecture Notes in Computer Science*, pages 276–286, 2007.

[95] A. Sandelin, W. Alkema, P. Engström, and W. Wasserman. JASPAR : an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, 32 :D91–D94, 2004.

[96] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 5(11) :909–917, 1999.

[97] D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. In *Proceedings of COCOON'97*, volume 1276 of *Lecture Notes in Computer Science*, pages 251–263, 1997.

[98] D. Sankoff and M. Blanchette. Multiple Genome Rearrangement and Breakpoint Phylogeny. *Journal of Computational Biology*, 5(3) :555–570, 1998.

[99] M. Satoh, T. Kubo, S. Nishizawa, A. Estiati, N. Itchoda, and T. Mikami. The cytoplasmic male-sterile type and normal type mitochondrial genomes of sugar beet share the same complement of genes of known function but differ in the content of expressed ORFs. *Molecular Genetics and Genomics*, 272(3) :247–256, 2004.

[100] T. Schenider and R. Stephens. Sequence Logos : A New Way to Display Consensus Sequences. *Nucleic Acids Research*, 18 :6097–6100, 1990.

[101] T. Schmidt and J. Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Proceedings of CPM'04*, volume 3109 of *Lecture Notes in Computer Science*, pages 347–358, 2004.

[102] D. Schones, A. Smith, and M. Zhang. Statistical significance of cis-regulatory modules. *BMC Bioinformatics*, 8(19), 2007.

[103] D. E. Schones, P. Sumazin, and M. Q. Zhang. Similarity of position frequency matrices for transcription factor binding sites. *Bioinformatics*, 21(3) :307–13, 2004.

[104] E. Segal, Y. Fondufe-Mittendorf, L. Chen, and A. Thastrom. A genomic code for nucleosome positioning. *Nature*, 442(7104) :772–780, 2006.

[105] J.-C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.

[106] A. Siepel. An algorithm to find all sorting reversals. In *Proceedings of RECOMB'02*, pages 281–290. ACM, 2002.

[107] B. Snel, P. Bork, and M. Huynen. Genome phylogeny based on gene content. *Nature Genetics*, 21 :108–110, 1999.

[108] B. Snel, P. Bork, and M. Huynen. The identification of functional modules from the genomic association of genes. *Proceedings of the National Academy of Sciences*, 99(9) :5890–5895, 2002.

[109] B. Snel, V. van Noort, and M. Huynen. Gene co-regulation is highly conserved in the evolution of eucaryotes and prokaryotes. *Nucleic Acids Research*, 32(16) :4725–4731, 2004.

[110] R. Staden. Methods for calculating the probabilities of finding patterns in sequences. *Computer Applications in Biosciences*, 5(2) :89–96, 1989.

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

[112] M. Suyama and P. Bork. Evolution of prokaryotic gene order : genome rearrangements in closely related species. *Trends in Genetics*, 17(1) :10–3, 2001.

[113] K. Swenson, M. Marron, J. Earnest-Deyoung, and B. Moret. Approximating the true evolutionary distance between two genomes. *Journal of Experimental Algorithmics*, 12 :1–17, 2008.

[114] E. Tannier, A. Bergeron, and M.-F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7) :881–888, 2007.

[115] R. L. Tatusov, E. V. Koonin, and D. J. Lipman. A Genomic Perspective on Protein Families. *Science*, 278(5338) :631–638, 1997.

[116] M. Y. Tolstorukov, V. Choudhary, W. K. Olson, V. B. Zhurkin, and P. J. Park. nuScore : a web-interface for nucleosome positioning predictions. *Bioinformatics*, 24(12) :1456–1458, 2008.

[117] Z. Trachtulec. Eukaryotic operon genes can define highly conserved synteny. *Folia Biologica (Praha)*, 50(1) :1–6, 2004.

[118] Z. Trachtulec and J. Forejt. Syntenies of unrelated genes conserved in mammals and non-vertebrates (a review). In D. Sankoff and J.-H. Nadeau, editors, *Comparative Genomics : Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, pages 357–366. Kluwer Academic Publishers, 2000.

[119] T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2) :290–309, 2000.

[120] S. Virtanen. Clustering the Chilean Web. In *Proceedings of LA-WEB'03*, pages 229–231, 2003.

[121] L. Wang, T. Warnow, B. Moret, R. Jansen, and L. Raubeson. Distance-Based Genome Rearrangement Phylogeny. *Journal of Molecular Evolution*, 63 :473–483, 2006.

[122] L.-S. Wang, R. K. Jansen, B. M. E. Moret, L. A. Raubeson, and T. Warnow. Fast phylogenetic methods for the analysis of genome rearrangement data : an empirical study. In *Proceedings of PSB'02*, pages 524–535, 2002.

[123] E. Willimas and D. Bowles. Coexpression of Neighboring Genes in the Genome of *Arabidopsis Thaliana*. *Genome Research*, 14 :1060–1067, 2004.

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

[125] T. D. Wu, C. G. Nevill-Manning, and D. L. Brutlag. Fast probabilistic analysis of sequence function using scoring matrices. *Bioinformatics*, 16(3) :233–44, 2000.

[126] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16) :3340–3346, 2005.

[127] J. Zhang, B. Jiang, M. Li, J. Tromp, X. Zhang, and M. Zhang. Computing exact p-values for DNA motifs (Part I). *Bioinformatics*, 23(5) :531–537, 2007.

Annexe administrative

Curriculum vitæ

Jean-Stéphane Varré

Laboratoire d'Informatique Fondamentale de Lille, équipe SEQUOIA
UFR IEEA - Bât M3
59655 Villeneuve d'Ascq Cedex
jean-stephane.varre@lifl.fr

2000 Doctorat en informatique
Université des Sciences et Technologies de Lille - Mention très honorable « Concepts et algorithmes pour la comparaison de séquences génétiques : une approche informationnelle » sous la direction de Jean-Paul Delahaye et Éric Rivals

2000 Ingénieur de recherche en bio-informatique au Centre Intégré de Bio-informatique de la génopole de Lille

2001 Maître de conférences, UFR IEEA, Université des Sciences et Technologies de Lille

Encadrements d'étudiants

Troisième cycle

2000 co-encadrement (25%) avec Jean-Paul Delahaye et Maude Pupin du stage de DEA de Mickaël Delauttre, « Alignement de séquences génétiques et compression de données »

2001 co-encadrement (50%) avec Jean-Paul Delahaye du stage de DEA puis de la thèse de Martin Figeac, « Étude de l'ordre des gènes : clusters de gènes et algorithmique des réarrangements », soutenue en 2005

2004 co-encadrement (50%) avec Hélène Touzet du stage de DEA puis de la thèse de Aude Liefooghe, « Matrices score-position, algorithmes et propriétés », soutenue en 2008

2005 encadrement du stage de DEA de Christophe Colomb « Détection de structures conservées chez les eucaryotes »

2006 co-encadrement (50%) avec Pascal Touzet (Laboratoire de Génétique et Évolution des Populations Végétales, Université des Sciences et Technologies de Lille) de la thèse de Aude Darracq, « Réarrangements du génome mitochondrial chez *Beta vulgaris* »

2008 encadrement du stage de Master Recherche de Sébastien Le Maguer, « Algorithmes de recherche de modules cis-régulateur »

2008 encadrement du stage de Mastère Recherche de Imen Hammami, « Extension de TFM-Explorer à la détection de modules »

Deuxième cycle

2006 Guillermo Mataz, Stage de master 1 informatique, Réalisation d'un logiciel de clustering de matrices poids-position.

2007 Arnaud Jacquemin, Stage de 2^{e} année IUP GMI, Réalisation d'un logiciel de représentation d'une phylogénie des réarrangements.

2007 Hanna Bhouri, Stage de master 1 informatique, Réalisation d'une plate-forme d'étude des scénarios de réarrangement.

2008 Hakim Boukhatem et Arnaud Ysmal, Stage de master 1 informatique, Implémentation de la recherche multiple de matrices score-position sur GPGPU (à 50% avec Mathieu Giraud).

Vie collective

- membre élu de la Comission de Spécialistes de la section 27 de l'Université des Sciences et Technologies de Lille de 2003 à 2008
- membre élu du Comité de Centre du centre de recherche INRIA Lille - Nord Europe depuis 2008
- responsable avec Ralf Blossey (IRI) d'une session de séminaires communs LIFL-IRI de 2006 à 2007
- responsable de la gestion matérielle et financière des Formations en Informatique de Lille de 2002 à 2007
- animateur du Groupe de Travail de Génomique Comparative depuis 2006 avec Sèverine Bérard, Éric Tannier, Hugues Roest-Crollius, Guillaume Fertin
- animation du groupe « Alignement » de l'action spécifique « Algorithmes pour la bioinformatique » du CNRS en 2001-2002
- membre du comité d'organisation de JOBIM 2008
- membre du comité d'organisation de RECOMB Comparative Genomics 2008
- membre du comité d'organisation de CPM 2009

Logiciels mis à disposition

Les logiciels sont téléchargeables à partir de `http ://bioinfo.lifl.fr`.

- TFM-scan : implémente les méthodes de filtrage et multi-index pour la recherche de matrices score-position
- TFM-pvalue : implémente les méthodes pour le calcul des associations score/P-valeur

– HUGO : implémente la détection d'über-operons
– TD : implémente la mesure de distance par copie de segments (travail de thèse)

Bibliographie (depuis la thèse)

Articles dans des revues internationales avec comité de lecture

1. Hélène Touzet and Jean-Stéphane Varré. Efficient and accurate P-value computation for Position Weight Matrices. *Algorithms in Molecular Biology*, 2(15), 2007.
2. Alexandra Coppin, Jean-Stéphane Varré, Luc Lienard, David Dauvillée, Yann Guérardel, Marie-Odile Soyer-Gobillard, Alain Buléon, Steven Ball, and Stanislas Tomavo. Evolution of Plant-Like Crystalline Storage Polysaccharide in the Protozoan Parasite Toxoplasma gondii Argues for a Red Alga Ancestry. *Journal of Molecular Evolution*, 60(2) :257–267, 2005.

Articles dans des conférences internationales avec comité de lecture

3. Aude Liefooghe, Hélène Touzet, and Jean-Stéphane Varré. Large Scale Matching for Position Weight Matrices. In *Proceedings of the 17th Symposium on Combinatorial Pattern Matching*, volume 4009 of *Lecture Notes in Computer Science*, pages 401–412, 2006.
4. Martin Figeac and Jean-Stéphane Varré. Sorting by Reversals with Common Intervals. In *Proceedings of the 4th Workshop on Algorithms In Bioinformatics*, volume 3240 of *Lectures Notes in Computer Science*, pages 26–37, 2004.

Posters, exposés et rapports techniques

5. Aude Darracq, Jean-Stéphane Varré, Adeline Courseaux, Laurence Maréchal-Drouard, and Pascal Touzet. Evolution of the mitochondrial genome in beet. A comparative genomic study at the intra-specific level, Poster in XX International Congress of Genetics, 2008.
6. Jean-Paul Delahaye, Martin Figeac, and Jean-Stéphane Varré. Détection de structures conservées dans les génomes de procaryotes. Technical Report 2004-1, LIFL, 2004.
7. Martin Figeac and Jean-Stéphane Varré. Detecting über-operons in prokaryotics genomes. Technical Report 2004-2, LIFL, 2004.
8. Hélène Touzet and Jean-Stéphane Varré. Calcul de P-valeur efficace et exact pour un motif PWM. Journées algorithmique, combinatoire du texte et applications en bio-informatique, 2007.
9. Jean-Stéphane Varré. Studying tumor genome architectures using genome rearrangement theory on End-Sequence Profiling data . ACI VicAnne/ARC MOCA workshop, 2006.
10. Aude Liefooghe, Hélène Touzet, and Jean-Stéphane Varré. Self-overlapping occurrences and Knuth-Morris-Pratt algorithm for weighted matching. Submitted to LATA'09.
11. Mathieu Giraud and Jean-Stéphane Varré. Parallel Algorithms for Position Weight Matrices. Submitted to HiComb'09.