Actes de la Réunion des Actions Spécifiques "Algorithmes et Séquences" et Indexation de Texte et Découverte de Motifs"

Eric Rivals, G. Kucherov, T. Lecroq

Actes de la réunion des Actions Spécifiques "Algorithmes et Séquences " et "Indexation de texte et découverte de motifs" Montpellier, 20 et 21 novembre 2003

Éditeurs :

Éric Rivals (rivals@lirmm.fr)
LIRMM/CNRS
161, rue Ada
34392 Montpellier Cedex 5

Gregory Kucherov (kucherov@loria.fr)
LORIA/INRIA,
615, rue du Jardin Botanique
54602 Villers-les-Nancy Cedex

Thierry Lecroq (Thierry.Lecroq@univ-rouen.fr)
LIFAR/Université de Rouen
place Émile Blondel
76821 Mont-Saint-Aignan

Site internet des actions spécifiques : http ://degas.lirmm.fr/ASIM/Reunion/.

Comité d'organisation local :

Sèverine Bérard (berard@lirmm.fr),
Éric Rivals (rivals@lirmm.fr) et
François Nicolas (nicolas@lirmm.fr).

Soutiens :

CNRS Département STIC,
RTP 41,
CNRS déléguation régionale Languedoc Roussillon,
Équipe-Projet Multi-Laboratoire (EPML) 64 du RTP 41 - Méthodes informatiques pour la biologie moléculaire,
Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier.

Table des matières

1 Méthode de recherche locale pour l'inférence d'arbres de duplication

Auteurs :

Denis Bertrand (LIRMM, Univ. Montpellier II) et

Olivier Gascuel (LIRMM, Montpellier CNRS)

Contact : dbertrand@lirmm.fr

Le problème de la reconstruction d'arbre de duplication (DT) à partir d'un ensemble de séquences répétées en tandem, obtenu par le processus de recombinaison inégale, a été introduit par Fitch (1977). Récemment de nombreuses recherches ont porté sur ce problème. Nous avons adapté, sur l'ensemble des DT, une heuristique, de type améliorations successives, très efficace sur les phylogénies. Pour cela, nous avons défini un ensemble de mouvements qui assurent de ne produire que des DT et permettent de tous les atteindre. De plus, nous avons utilisé une approche stochastique de type bruitage pour améliorer nos résultats. Nous avons testé l'exactitude topologique de notre heuristique à l'aide de jeux de données simulés, et nous l'avons comparée aux méthodes de reconstructions existantes. Les résultats montrent que la technique de recherche locale est plus précise que toutes les autres méthodes que nous avons étudiées.

2 The Transformation Distance Problem Revisited

Auteurs :

Behshad Behzadi (LIX, École Polytechnique, Palaiseau) et

Jean-Marc Steyaert (LIX, École Polytechnique, Palaiseau)

Contact : behzadi@lix.polytechnique.fr

2.1 Introduction

Evolution acts in several ways on biological sequences : either by mutating an element, or by inserting, deleting or copying a segment of the sequence. Varré et al. [1] defined a transformation distance for the sequences, in which the evolutionary operations are copy, reverse copy and insertion of a segment. They also proposed an algorithm to calculate the transformation distance. This algorithm is $O(n^4)$ in time and $O(n^4)$ in space, where n is the size of the sequences. In this work, we propose an improved algorithm which costs $O(n^2)$ in time and $O(n^2)$ in space. Furthermore, we extend the operation set by adding point deletions. We present an algorithm which is $O(n^3)$ in time and $O(n^2)$ in space for this extended case.

2.2 Model and Problem Description

Varré et al. [1, 2] propose a new measure which evaluates segment-based dissimilarity between two strings : the source string S and the target string T. This measure is related to the process of constructing the target string T with *segment operations*[1]. The construction starts with the empty string ϵ and proceeds from left to right by adding segments (concatenation), one segment per operation. The left-to-right generation is not a restriction but a fact that can be formally proved. A list of operations is called a *script*. Three types of segment operations are considered : the *copy* adds segments that are contained in the source string S, the *reverse copy* adds the segments that are contained in S in reverse order, and the *insertion* adds segments that are not necessarily contained in S. Depending on the number of common segments between S and T, there exist several scripts for constructing the target T. Among these scripts, some are more likely ; in order to identify them, we introduce a cost function for each operation. The cost of a script is the sum of the costs of its

[1] We use segment as an equivalent word for substring.

operations. The *minimal scripts* are all scripts of minimum cost and the *transformation distance*[2] *(TD)* is the cost of a minimal script. The problem which we solve in this work is the computation of the transformation distance. It is clear that it is also possible to get a minimal script.

2.3 Algorithm

Our algorithm consists of two parts. The first part is a preprocessing part in which we determine for each substring of target string T, whether it exists in the source string S or not. For this aim, we design an algorithm based on KMP (Knutt-Moris-Pratt) string matching algorithm with some changes. We show that one can search for the presence of the prefixes of a pattern string in the source string, in the same time of searching for the complete pattern, without increasing the complexity of the search. The total complexity of the preprocessing part is $O(n^2)$ in time and $O(n^2)$ in space.

In the second part, which is the core algorithm, we determine the transformation distance with help of the information that we obtained in the preprocessing part. As the scripts construct the target string T from left to right by adding segments, dynamic programming is an ideal tool for computing the transformation distance. Let $C[k]$ be the minimum production cost of $T[1..k]$ using the segments of S. This table is computed and filled recursively by a dynamic programming which costs $O(n^2)$ in time and $O(n)$ in space. So the total complexity of our algorithm (preprocessing + core algorithm) is $O(n^2)$ in time and $O(n^2)$ in space.

2.4 An Additional Operation : Point Deletion

In this section, we extend the set of evolutionary operations by adding the *point deletion* operation. In the real evolution of biological sequences, in several cases after or during the copy operations some bases (symbols) are eliminated. We consider a cost function for the deletion of unique symbols. Depending on the assigned costs, deletions can be used after the copy or reverse copy operations. We define two new operations called *NewCopy* and *NewRevCopy* which are a copy operation or a reverse copy operation, followed by zero or more deletions on the copied segment. Solving the extended transformation distance with the point deletions, amounts to solve the transformation distance with the following three operations : Insertion, NewCopy and NewRevCopy. A substring $T[i..j]$ of the target string can be produced by a unique NewCopy operation if and only if $T[i..j]$ is a subsequence string of source S. In a preprocessing part, the algorithm determines the minimum generation cost by a NewCopy or NewRevCopy operation, for any substring of the target string T. Very similar to the last section algorithm, a dynamic programming algorithm calculates the extended transformation distance in the new core algorithm. The whole complexity of the new algorithm for the calculation of extended transformation distance is $O(n^3)$ in time and $O(n^2)$ in space.

Références

[1] J.-S. Varré, J.P. Delahaye, and É. Rivals. The transformation distance : A dissimilarity measure based on movements of segments. In *German Conference on Bioinformatics, Koel - Germany*, 1998.

[2] J.-S. Varré, J.P. Delahaye, and É. Rivals. Transformation distances : A family of dissimilarity measures based on movements of segments. *Bioinformatics*, 15(3) :194–202, 1999.

3 Analyse de séquences par compression : un algorithme efficace de localisation optimale de régularités

Auteurs :

Olivier Delgrange (LIFL et Univ. de Mons-Hainaut),

[2] Although this measure is not a mathematical distance but we will use the term transformation distance which was introduced by Varré et al. [1, 2].

Eric Rivals (LIRMM, Montpellier CNRS)

Contact : Olivier.Delgrange@umh.ac.be

La compression informatique peut être utilisée efficacement pour analyser de l'information, plus particulièrement des séquences de symboles. En effet, un programme de compression parvient à réduire la taille d'une séquence s'il parvient à exploiter des redondances (des "régularités") dans la séquence.

Habituellement, les méthodes de compression exploitent les redondances d'un bout à l'autre de la séquence à comprimer. Si certaines zones sont effectivement raccourcies de cette façon, d'autres zones sont malheureusement rallongées. Nous proposons un algorithme permettant, une fois une séquence comprimée, de rompre le schéma de compression là où il n'est pas intéressant en recopiant ces morceaux de la séquence tels quels. Le gain global en compression s'en trouve amélioré. Notre algorithme propose une décomposition, en zones à comprimer et zones à recopier, optimale dans le sens où elle maximise le gain en compression résultant. Notre algorithme d'optimisation fonctionne en temps $O(n \log n)$ où n est la longueur de la séquence.

L'optimisation nous permet donc de localiser, de manière optimale, les zones régulières et irrégulières pour le type précis de régularités exploité par la méthode de compression initiale.

Nous détaillons plus spécifiquement une application dans le domaine de l'analyse de séquences génétiques : la localisation de répétitions en tandem approximatives dans une séquence d'ADN (algorithme *STAR*). Notre méthode très générale permet de nombreuses autres applications comme par exemple la localisation de zones localement répétitives dans un texte ou une séquence par optimisation d'une compression de type Lempel-Ziv (LZSS).

4 Estimation de la sensibilité de graines pour la recherche de similarités dans les séquences d'ADN

Auteurs :

Grégory Kucherov (INRIA/LORIA, Nancy),

Laurent Noé (LORIA, Nancy) et

Yann Ponty (LRI, Nancy)

Contact : kucherov@loria.fr

L'identification de régions de similarité dans les séquences d'ADN (alignement local) est un problème fondamental en bioinformatique. La plupart des algorithmes utilisent la technique basée sur la recherche de petites répétitions exactes de taille fixe, appelées *graines*, qui sont ensuite étendues à des répétitions approchées plus larges. BLAST est l'exemple le plus représentatif de cette famille d'algorithmes.

Récemment, de nouvelles idées sont apparues qui permettent d'augmenter l'efficacité de cette approche. Ces idées ont donné lieu à de nouveaux programmes d'alignement local de séquences d'ADN, tels que PatternHunter, BLAT ou YASS. Parmi les nouvelles techniques, l'utilisation de *graines espacées* permet d'accroître la sensibilité de la recherche, sans perdre en sélectivité (et donc en temps d'exécution). Une autre technique indépendante, exploitée dans YASS, consiste à utiliser des *groupes de graines* plutôt que des graines individuelles ou des couples de graines. Enfin, une prise en compte plus fine de propriétés de séquences (telle que la distinction entre différents types de mismatches) permet également d'augmenter la performance de l'algorithme.

L'apparition de ces nouvelles techniques pose de nouvelles questions : comment choisir les graines espacées ? plus généralement, quels sont les critères pour estimer la sensibilité d'un algorithme et pour comparer les différents algorithmes entre eux ?

Dans cet exposé, je donnerai quelques éléments de réponse à ces questions. En particulier, je présenterai une approche basé sur la modélisation d'alignements par les chaînes de Markov, utilisée par quelques auteurs, ainsi qu'une autre approche, basée elle sur les alignements dits *homogènes*. Dans le cadre de cette dernière approche, je présenterai quelques résultats que nous avons obtenus récemment ainsi que des résultats expérimentaux.

5 Utilisation de familles de graines pour la recherche de motifs par filtrage

Auteurs :

Grégory Kucherov (INRIA/LORIA, Nancy) et

Laurent Noé (LORIA, Nancy)

Contact : `laurent.noe@loria.fr`

Nous proposons une méthode de recherche de motifs de taille fixée m avec au plus k substitutions dans le texte. Cette méthode se base sur le *design* et l'utilisation de familles de graines espacées (nommées également Q-grams espacés). Burkhardt et Kärkkaïnen [1] avaient développé et analysé une approche basée sur les graines espacées en n'utilisant qu'une seule graine pour la recherche. Nous proposons ici une extension du filtre en réalisant le *design* simultané de plusieurs graines utilisées de manière disjonctive, permettant ainsi d'améliorer les taux de filtrage d'un facteur 20 sur des problèmes comme la recherche de fragments uniques à k erreurs près pour le design d'oligos.

5.1 Introduction

Les graines espacées (ou Q-grams espacés) ont été proposées en Bioinformatique dans l'algorithme implanté dans `FLASH` (Fast Look-Up Algorithm for String Homology [2]). Il s'agissait d'une méthode dérivée de celles proposées conjointement pour la reconnaisance de formes associée à des problèmes de vision. Le principe du filtrage avec perte (respectivement sans perte) consiste à éliminer de manière rapide des régions qui ont peu de chances (respectivement aucune chance) d'être similaires selon un critère fixé. Malgré son efficacité, le principe n'a pas été développé durant les années 90 pour les problèmes biologiques. Ce n'est que vers le début de la décennie que le concept est réapparu pour résoudre deux problèmes différents :

- la recherche exacte de fragments de taille m fixée ayant au plus k substitutions (que l'on notera par la suite problème (m, k)).
- la recherche heuristique de séquences redondantes, visant à améliorer la sensibilité des algorithmes d'alignement local de type heuristique (`FASTA`,`BLAST`,`Pattern-Hunter`,`YASS`,...).

5.2 Filtrage de texte

Le premier problème (recherche exacte par utilisation d'un filtre sans perte) a été étudié en détail par Burkhardt et Kärkkäinen. Il s'agissait de trouver une graine espacée permettant de résoudre des problèmes (m, k) déterminés, c'est à dire réaliser un filtre sans perte trouvant la totalité des instances de motifs ayant k erreurs de substitutions.

Un extension proposée ici pour cette méthode consiste à utiliser plusieurs graines (une famille de graines). Le but est d'augmenter la selectivité du filtre en utilisant des graines dites de poids élevé. (tout en se limitant à des tailles de familles raisonnables).

Le design des graines par la méthode proposée en [1] n'étant plus applicable en pratique dès que la taille de la famille atteint 3 graines ou plus, une approche heuristique a été adoptée (algorithme génétique en l'occurrence).

5.3 Résultats et Experiences

Sur un alphabet à 4 lettres, la méthode précédemment décrite donne des familles de graines ayant une meilleure sélectivité globale, comparée aux méthodes de filtrage les plus courantes [3]. On mesure la sélectivité δ comme la probabilité que deux positions aléatoires sur deux séquences i.i.d vérifient le critère du filtre.

Références

[1] S. Burkhardt and J. Kärkkäinen. Better Filtering with Gapped q-Grams. *Fundamenta Informaticae*, 23 :1001–1018, 2003.

TAB. 1 – familles de graines trouvées pour le problème $(m = 50, k = 5)$

w	$taille\ famille$	$exemple$	δ
12	1	111010011101001110 1 ou 10101000100000101010001000001010100013	$5.96\ 10^{-8}$
14	2	11110101100111101011 1011001111010110011114	$7.45\ 10^{-9}$
15	3	100110101111110011010 11 1011111100110101111 1 11110011010111111 00 11	$2.79\ 10^{-9}$
16	4	1110110101110011111111 1101011100111111101101 1110011111111011010111 11111110110101110011 1	$9.31\ 10^{-10}$
17	6	1101011001111111011110 1 1011001111111011110 1011 1111111011110101100111 1110111101011001111111 11110101100111111101 11 11001111111101111010110 01	$3.49\ 10^{-10}$

TAB. 2 – familles de graines trouvées pour le problème $(m = 32, k = 5)$

w	$taille\ famille$	δ
7	1	$6.10\ 10^{-5}$
8	2	$3.05\ 10^{-5}$
9	3	$1.14\ 10^{-5}$
10	4	$3.81\ 10^{-6}$
11	6	$1.43\ 10^{-6}$

[2] A. Califano and I. Rigoutsos. Flash : A fast look-up algorithm for string homology. In *Proc. of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 56–64, 1993.

[3] P. Pevzner and M. Waterman. Multiple Filtration and Approximate Pattern Matching. *Algorithmica*, pages 135–154, 1995.

6 Index compacts

Auteur : Maxime Crochemore (IGM, Univ. de Marne-la-Vallée)

Contact : `Maxime.Crochemore@univ-mlv.fr`

L'exposé porte sur les structures de données et algorithmes pour gérer des index complets. Les index complets sont ceux qui contiennent tous les segments d'un texte cible. Ils ont un très grand nombre d'utilisation dans les systèmes de recherche, la compression de données, et l'extraction de motifs, à titre d'exemple. Le problème principal est celui de l'optimisation de l'espace nécessaire au stockage de l'index, sans pénaliser pour autant le temps d'exécution des opérations de base (accès, comptage, localisation). Les structures classiques pour implémenter cette notion sont l'arbre des suffixes, l'automate des suffixes, et le tableau des suffixes. On fera un survol de ces méthodes et de leurs derniers développements.

7 Extraction de mots approchés : les mots primitifs

Auteurs :

Saïd Abdeddaïm (LIFAR/ABISS, Univ. de Rouen),

Joël Alexandre (UMR 6037/ABISS, Univ. de Rouen) et

Johann Pelfrêne (UMR 6037/ABISS, Univ. de Rouen)

Contact `johann.pelfrene@univ-rouen.fr`

Le nombre de mots approchés dans une séquence est exponentiel. Notre travail a consisté en l'amélioration d'une méthode récente introduite par Parida *et al.* [2], qui consistait à ne sélectionner que quelques mots à jokers, plus intéressants dans le sens où ils forment une base pour les autres mots à jokers. Nous avons donc défini les mots primitifs, dont le nombre est inférieur à celui des mots décrits dans [2]. Nous présentons ici cette définition, ainsi que trois algorithmes. Le premier permet de déterminer les mots primitifs apparaissant au moins deux fois. Le second extrait ceux apparaissant au moins q fois à partir de ceux apparaissant au moins $q-1$ fois. Le dernier permet de tester la primitivité d'un mot approché. Nous montrons par ailleurs que le nombre de primitifs est exponentiel lui aussi.

7.1 Introduction

Les mots approchés sont des objets structurés répétés dans un texte. Les algorithmes d'extraction de ce type de mot ont de nombreuses applications en bioinformatique en particulier, et d'en d'autres champs de l'informatique tels que la compression de données.

Nous considérons les mots approchés avec joker, une lettre qui correspond à toutes les autres. Le nombre de mots à joker est exponentiel en la taille de la séquence. Un travail récent [2] présentait une idée qui tendait à réduire le nombre de mots en ne prenant en compte que certains mots, formant une base pour les autres. Nous avons proposé [3, 4, 5] la nouvelle définition de mot primitif, dont le nombre est inférieur à celui des maximaux non redondants de [2]. Les mots q-primitifs sont des mots qui apparaissent au moins q fois. Pour $q = 2$, le nombre de primitif est linéaire en la taille de la séquence, et nous avons proposé un algorithme en $O(n^3)$ permettant de les extraire [4]. Dans [5], nous avons proposé une amélioration de cet algorithme, dont la complexité est $O(|\Sigma|n^2 \log n)$. Nous avons aussi présenté un algorithme incrémental calculant les primitifs pour q à partir des primitifs pour $q-1$. Si le nombre de primitif est inférieur à celui des maximaux non redondants, nous montrons qu'il reste cependant exponentiel.

Très récemment, Pisanti *et al.* [6, 7] ont présenté indépendamment la même notion, un algorithme de même complexité pour $q = 2$, ainsi qu'une discussion sur le nombre des mots primitifs.

Dans la suite, nous présentons brièvement la notion de mot primitif, ainsi que l'algorithme incrémental qui permet de les extraire, puis un algorithme qui permet de décider si un mot est primitif ou non.

7.2 Les mots primitifs

Nous nous intéressons à des mots définis par leur ensemble de positions dans le texte d'intérêt. Si m un est mot approché dans le texte t, nous notons $pos(m, t)$ l'ensemble des positions de m dans t. La notion de maximalité, introduite dans [2], garanti que tout mot maximal ne possède pas de joker superflu, et ne peut être étendu à gauche ou à droite sans modifier l'ensemble des positions. Un mot maximal est primitif si il ne peut être écrit comme l'union d'autres maximaux, à un décalage près : $pos(m, t) = \cup_{i>1}(pos(m_i, t) + \delta_i)$. Un mot non primitif est ainsi en quelque sorte le résultat d'un mélange de facteurs de mot maximaux. Les mots primitifs forment une base, dans le sens où l'ensemble des primitifs est libre, et engendre les autres mots maximaux. Dans le texte $t =$`AAAAAXAAAAA`, les primitifs sont au nombre de cinq : `AAAAA`, `A_AAA_A`, `AA_AA_AA`, `AAA_A_AAA` et `AAAA_AAAA`. Le mot `AAA__AAA` n'est pas primitif, car $pos($`AAA__AAA`$, t) = \{1, 2, 3\}$ et est l'union (ici sans décalage) des mots `AAAA_AAAA` ($\{1, 2\}$) et `AAA_A_AAA` ($\{1, 3\}$).

Le quorum est le nombre minimal d'occurrence que doit avoir un mot pour être pris en compte. En effet, utilisée telle que, cette définition ne produit qu'un seul primitif, le texte lui-même. Pour obtenir les autres, il nous faut ne plus prendre en compte ceux qui apparaissent moins souvent. Pour $q = 2$, le nombre de primitif est inférieur à la taille du texte. Pour les autres quorum, le nombre théorique est de l'ordre de C_{n-1}^{q-1}, où n est la taille de la séquence d'entrée.

7.3 Algorithme d'extraction

Notre algorithme d'extraction suit une approche incrémentale : nous obtenons les primitifs pour le quorum q à partir de ceux pour le quorum $q-1$. L'algorithme fonctionne en deux étapes. La première étape consiste à calculer un ensemble de mots contenant au moins les primitifs, mais

contenant aussi des non primitifs. La seconde étape est un nettoyage de cet ensemble. Ce nettoyage est effectué en deux temps : il faut d'abord calculer les positions des mots, puis appliquer l'algorithme de décision décrit dans le suite.

7.4 Algorithme de décision

Le test de primitivité d'un mot est dérivé de l'algorithme de Fisher et Paterson [1] que nous utilisons par ailleurs pour obtenir les positions d'un mot. Dans une première passe, nous obtenons la liste des jokers pouvant être remplacés au moins dans q positions : cela montre l'existence d'un mot qui couvre, et donc potentiellement participe à rendre m non primitif. Dans une seconde passe, nous créons de faux mots en utilisant toutes les lettres détectées pendant la première passe, puis nous calculons quelles sont les positions réellement couvertes. Si elles le sont toutes, alors le mot n'est pas primitif.

7.5 Conclusion et perspectives

Nous présentons un algorithme qui permet d'extraire les mots approchés que nous avons définis comme des mots primitifs. Nous proposons aussi un algorithme de décision de la primitivité, et nous montrons que le nombre de primitifs reste exponentiel en la taille de la séquence d'entrée.

Nos premiers tests montrent qu'en pratique, le nombre de primitifs est très très grand, mais bien inférieur à ce que prévoit la théorie. Nous avons aussi pu constater qu'un certain nombre de mots sont très peu significatifs, avec beaucoup de jokers et peu de lettres. Une étape de filtrage serait ainsi nécessaire pour réduire et rendre utilisable cet ensemble de mots primitifs.

Références

[1] M. J. Fischer and M. S. Paterson. String matching and other products. *SIAM-AMS proceedings*, pages 113–125, 1974.

[2] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao. Pattern discovery on character sets and real-valued data : linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proceedings of the 11th Symposium on Discrete Algorithms*, pages 297–308, 2000.

[3] J. Pelfrêne. Indexation de motifs approchés. rapport de DÉA, september 2000.

[4] J. Pelfrêne, S. Abdeddaïm, and J. Alexandre. Un algorithme d'indexation de motifs approchés (poster and short talk). In *Journées Ouvertes Biologie Informatique Mathématiques, Saint-Malo*, pages 263–264, June 2002.

[5] J. Pelfrêne, S. Abdeddaïm, and J. Alexandre. Extracting approximate patterns. In Ricardo Baez-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Proceedings of the 14th annual symposium on Combinatorial Pattern Matching (CPM 2003)*, pages 328–347. Springer, June 2003.

[6] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. Bases of motifs for generating repeated patterns with don't cares. Technical report, Università di Pisa, february 2003.

[7] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. A basis of tiling motifs for generating repeated patterns and its complexity for higher quorum. In *proceedings of Mathematical Foundations of Computer Science (MFCS)*, volume 2747, pages 622–632. Springer LNCS, 2003.

8 Statistiques de q-grams et modèles d'urnes

Auteur : Pierre Nicodème (LIX, École Polytechnique, Palaiseau)

Contact `nicodeme@lix.polytechnique.fr`

Utilisant une terminologie courante, nous désignons par q-gram un mot de taille q.

Nous considérons des séquences aléatoires générées selon un modèle Bernoulli non-uniforme et les statistiques suivantes :

– (I) nombre Q_I de q-grams répétés dans une séquence (sans compter les répétitions),

– (II) nombre Q_{II} de q-grams communs à deux séquences (sans tenir compte des répétitions),
– (III) nombre Q_{III} de q-grams communs à deux séquences (en tenant compte des répétitions dans une séquence "requête").

À titre d'exemple, considérons le cas $q = 2$ et les deux séquences :

$$\begin{array}{rcl} S_1 = aaabaaab & \Rightarrow & 2-\text{grams} = \{\{aa, aa, ab, ba, aa, aa, ab\}\} \\ S_2 = aaacaaaa & \Rightarrow & 2-\text{grams} = \{\{aa, aa, ac, ca, aa, aa, aa\}\} \end{array}$$

Nous avons :

– $Q_I = |\{aa, ab\}| = 2$,
– $Q_{II} = |\{aa\}| = 1$,
– $Q_{III} = |\{\{aa, aa, aa, aa\}\}| = 4$, quand c'est la séquence S_1 qui est prise comme "requête" (où l'on compte les répétitions).

La manière de compter utilisée dans le troisième cas est celle du lemme de Jokinen et Ukkonen, qui donne une borne inférieure du nombre de q-grams communs à deux séquences fonction de la distance d'édition séparant ces deux séquences.

8.1 Méthode

Nous allons considérer un modèle approché, où chaque q-gram est tiré indépendement des autres q-grams ; (dans le modèle réel, aux effets de bords près, un q-gram recouvre partiellement les $q-1$ q-grams précédants et les $q-1$ suivants). Le modèle approché est équivalent à un modèle d'urnes où l'on associe une urne à chaque q-gram. Tirer au hasard un q-gram est équivalent à jeter au hasard une boule dans le système d'urnes. Si l'on compare deux séquences, on associera à chaque séquence une couleur, et on aura un système d'urnes et de boules de deux couleurs. Nous effectuerons une analyse probabiliste de ce modèle, et nous le comparerons aux résultats obtenus par simulation dans le modèle réel.

8.2 Définitions

Nous considérons un alphabet Δ de taille d et les q-grams des séquences de taille $n+q-1$. Dans le modèle approché, nous tirons aléatoirement n q-grams. Nous noterons p_i la probabilité de jeter une boule dans l'urne i. Pour le q-gram $w_1 w_2 \dots w_q$, la probabilité de l'urne associée sera $p = \pi(w_1) \times \pi(w_2) \times \cdots \times \pi(w_q)$ où $\pi(w_j)$ est la probabilité de la lettre w_j $(j = 1..q)$.

8.3 Analyse du modèle indépendant

Nous considérons les séries génératrices

$$G(z,u) = \sum_{n\geq 0, k\geq 0} g_{nk} u^k \frac{z^n}{n!} \quad \text{et} \quad F_X(z,t,u) = \sum_{r\geq 0, s\geq 0, k\geq 0} f_{X,rsk} u^k \frac{z^r t^s}{r!s!}$$

où g_{nk} est la probabilité d'avoir k q-grams répétés dans une séquence de taille n et $f_{X,rsk}$ est la probabilité d'avoir k q-grams communs à deux séquences de longueur r et s dans le modèle X (II ou III). Ces séries génératrices peuvent être calculées par Poissonisation-Dépoissonisation. On obtient (avec $m = d^q$, le nombre d'urnes) :

$$\begin{aligned} G(z,u) &= \prod_{0\leq i\leq m-1} (e^{p_i z} + (u-1)(1+p_i z)) \,, \\ F_{II}(z,t,u) &= \prod_{0\leq i\leq m-1} \left(e^{p_i(z+t)} + (u-1)(e^{p_i z} + e^{p_i t} - 1)\right) \,, \\ F_{III}(z,t,u) &= \prod_{0\leq i\leq m-1} \left(e^{p_i(uz+t)} - e^{p_i uz} + e^{p_i z}\right) \,. \end{aligned}$$

$\Delta = \{0, 1\}$ $d = 2$ $q = 10$ $m = d^q = 1024$ $n = 300$ $p_0 = \Pr(0)$
traits pleins : espérance μ_n et écart-type σ_n théoriques pour le modèle indépendant ; pointillés : simulations

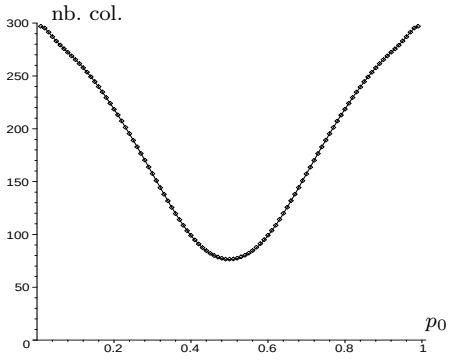


Figure 1 : Modèle indépendant : espérance

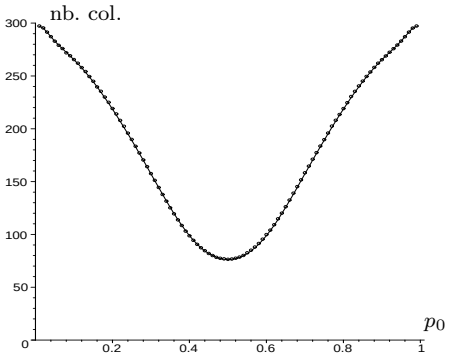


Figure 2 : Modèle dépendant : espérance

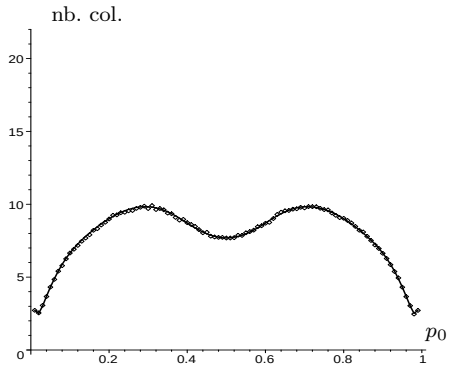


Figure 3 : Modèle indépendant : écart-type

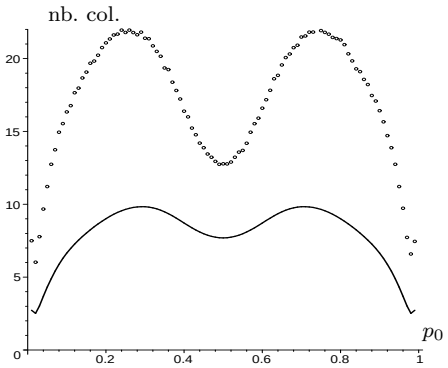


Figure 4 : Modèle dépendant : écart-type

Pour le modèle III, on obtient asymptotiquement pour deux séquences de taille $r = s = n$, avec $n \times p_i \to \theta_i$,

$$\text{espérance} \qquad \mu_n \approx \sum_i \kappa_i \quad \text{avec} \quad \kappa_i = \sum_i \theta_i \left(1 - e^{-\theta_i}\left(1 - \frac{\theta_i^2}{2n}\right)\right),$$

$$\text{variance} \qquad \sigma_n^2 \approx \sum_i \kappa_i(\theta_i - \kappa_i) - \frac{1}{n}\left(\left(\sum_i \theta_i(1 - e^{-\theta_i})\right)^2 + \left(\sum_i \theta_i^2 e^{-\theta_i}\right)\right),$$

où la sommation sur i est faite de 0 à $d^q - 1$. Les figures 1 à 4 comparent ce résultat théorique (modèle III indépendant) avec les simulations faites pour le modèle III exact. Ceci montre que le modèle indépendent est une excellente approximation au premier ordre. (Similairement, l'espérance du nombre d'occurrences de mots dans des textes aléatoires est indépendante de l'auto-corrélation, contrairement à l'écart-type).

Références

[1] P. Nicodème. q-gram analysis and urn models. In *Discrete Random Walks 2003*, pages 243–257. DMTCS, 2003. Paris.

[2] S. Rahmann and É. Rivals. Exact and Efficient Computation of the Expected Number of Missing and Common Words in Random Texts. In R. Giancarlo and D. Sankoff, editors, *Proc. of the 11th Symposium on Combinatorial Pattern Matching*, volume 1848 of *Lecture Notes in Computer Science*, pages 375–387. Springer-Verlag, Berlin, 2000.

[3] S. Rahmann and É. Rivals. The number of missing words in random texts. *Combinatorics, Probability and Computing*, 12 :73–87, 2003.

9 Détection de structures conservées dans les génomes de Procaryotes

Auteurs :

Jean-Paul Delahaye (LIFL, Univ. de Lille),

Martin Figeac (LIFL, Univ. de Lille) et

Jean-Stéphane Varré (LIFL, Univ. de Lille)

Contact : `martin.figeac@lifl.fr`

Les méthodes de détection de motifs conservés au niveau de l'ordre des gènes détectent de moins en moins de clusters quand on augmente l'information disponible. En augmentant le nombre de génomes comparés, le nombre de contraintes augmente et moins de résultats sont obtenus ; même les adjacences conservées ne seront pas identifiées. Nous présentons les *méta-clusters*, un nouveau type de motif, et leur méthode de détection en temps polynomial autorisant les gènes paralogues. Les *méta-clusters* sont tels que, plus on ajoute d'informations, plus ceux-ci contiennent de gènes. Ajouter une espèce augmente les possibilités d'adjacences ou augmente leur nombre d'occurrences. Nous présentons aussi une variante des *méta-clusters* qui fournit pour la première fois une méthode exacte de détection des *über-operons*.

9.1 Introduction

Soit $\hat{\Sigma}$ l'ensemble des gènes communs à au moins deux génomes parmi n. Un génome est représenté par une liste ordonnée sur $\Sigma = \hat{\Sigma} \cup \{*\}$, appelée permutation, où $*$ représente les gènes n'appartenant pas à $\hat{\Sigma}$. Soit $\Pi = \{\pi_i\}$ l'ensemble des permutations représentant les n génomes. Une permutation $\pi =$`(1 * 7 4 1 2 * 9)` est telle que les entiers représentent des gènes de $\hat{\Sigma}$, présents dans au moins un autre génome.

Deux gènes g_1 et g_2 de $\hat{\Sigma}$ sont *voisins* sur une permutation π s'il existe une position de g_1 et une position de g_2 dans π notées $\text{POS}_\pi(g_1)$ et $\text{POS}_\pi(g_2)$ telles que $|\text{POS}_\pi(g_1) - \text{POS}_\pi(g_2)| - 1 = 0$ est vérifiée. Dans l'exemple précédent les gènes 1 et 2 sont voisins.

Lorsque l'on compare l'ordre des gènes, le motif le plus simple conservé entre deux génomes est la paire de gènes adjacents. Deux gènes forment une *paire de gènes adjacents*, s'ils sont voisins dans au moins deux permutations. Si on étend ce motif à plus de deux génomes, on s'attend à trouver des paires de gènes adjacents conservées à k permutations. Or plus on augmente le nombre de génomes, plus le nombre d'adjacences conservées diminue [1]. Il en est de même pour les motifs plus élaborés comme les intervalles communs [3] et les Gene Teams [2]. Pour un ensemble de k permutations sans étoile et sans doublon [2] $\tilde{\Pi} = \{\tilde{\pi}_i\}$, un intervalle commun est un ensemble d'éléments de $\tilde{\Sigma}$ tel qu'il forme une sous permutation dans chaque permutation $\tilde{\pi}_i$. Les Gene Teams reprennent le même principe mais autorisent les étoiles. Ces définitions sont trop strictes pour qu'il existe des motifs conservés à beaucoup d'espèces. De plus ces motifs n'autorisent pas la présence de gènes dupliqués dans les génomes. Nous proposons ici le concept des *méta-clusters* qui est inspiré des Connected Gene Neighborhoods [6]. Un *méta-cluster* est un ensemble de gènes tel que pour suffisamment de génomes, chaque gène forme une paire adjacente avec un autre gène de cet ensemble. Nous allons donner une définition formelle à l'aide de graphes.

9.2 Présentation et détection des méta-clusters

Nous généralisons la définition de gènes voisins. Soit δ est un entier naturel ou $+\infty$. Deux gènes g_1 et g_2 de $\hat{\Sigma}$ sont *voisins* s'il existe une position de g_1 et une position de g_2 telles qu'entre ces deux positions il n'y ait que des étoiles et en nombre inférieur à δ.

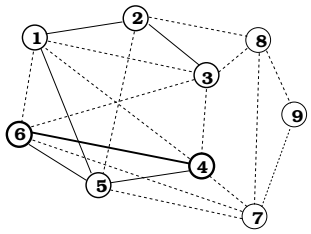
Nous définissons le graphe non orienté pondéré $G = (S, A, w)$ appelé graphe de voisinage. S est l'ensemble des gènes de $\hat{\Sigma}$, A l'ensemble des arcs de G et w une fonction de poids. Deux noeuds sont reliés par un arc a de poids $w(a) = k$ si les noeuds associés à l'arc a sont voisins exactement k

[2] On se restreint à l'ensemble des gènes communs à tous les génomes et sans duplication noté $\tilde{\Sigma}$.

fois. Soit α un entier positif, on définit le graphe G_α identique au graphe G dans lequel on supprime les arcs de poids inférieur à α. **Les composantes connexes du graphe $\mathbf{G}_\alpha$ seront appelées les α-clusters**.

La présence de bruit dans les génomes nous conduit à une nouvelle généralisation de la notion de voisins. Un chemin d'une permutation π sur Σ est une suite $g_1 g_2 \dots g_n$ de gènes de $\hat{\Sigma}$ pris dans π, tel que pour $1 \leq j < n$ on ait $|\text{POS}_\pi(g_j) - \text{POS}_\pi(g_{j+1})| - 1 \leq \delta$. Soit γ un entier ou $+\infty$. Nous définissons le graphe non orienté pondéré $G_\gamma = (S, A_\gamma, w_\gamma)$ appelé graphe des chemins. Deux noeuds de S sont reliés par un arc a de poids $w_\gamma(a) = k$ si les noeuds associés à l'arc a sont dans exactement k chemins de longueur au plus γ. Sur le même principe que précédemment, on supprime les arcs de G_γ de poids inférieur à α pour obtenir le graphe $G_{\alpha,\gamma}$. **Les composantes connexes du graphe $\mathbf{G}_{\alpha,\gamma}$ seront appelées des α-γ-clusters ou méta-clusters**.

Prenons l'exemple suivant avec trois permutations : $\pi_1 = (6 * 5 * 4\ 7 * ... * 1\ 3 * 2)$, $\pi_2 = (5 * 4\ 1\ 6 * ... * 2\ 3\ 8)$ et $\pi_3 = (2\ 1\ 5 * ... * 4\ 6 * 3 * ... * 7\ 8\ 9)$.



Ci-dessus est donné le graphe des chemins pour $\gamma = \infty$ et $\delta = 1$. Les arcs en pointillés sont détectés une seule fois, les arcs pleins deux fois et les arcs en gras trois fois. L'arc $(1, 2)$ est identifié deux fois car il existe un chemin de 1 vers 3 dans π_1 et un autre dans π_3. On obtient le *méta-cluster* {1 2 3 4 5 6} si $\alpha = 2$ et {4 6} si $\alpha = 3$. Les Gene Teams ne donnent aucun cluster bien qu'une structure conservée existe.

Utiliser les chemins plutôt que les voisinages permet de prendre en compte des génomes plus mélangés et où les ensembles de gènes de chaque génome sont assez différents.

9.3 Variante pour la détection d'über-operons

Un opéron est une suite de gènes transcrits en une seule fois ; ils ont souvent une fonction commune ou apparentée [5]. Un gène peut appartenir à tel opéron dans un génome alors qu'un de ses orthologues, dans un autre génome, appartient à un autre opéron. [4] présentent des exemples de tels gènes et les nomment des *über-operons*. Ils proposent la définition suivante : les *über-operons* sont détectés et définis en assemblant de façon itérative les gènes voisins d'un même opéron. Cette méthode n'est pas assez souple pour détecter tous les *über-operons* attendus.

Basé sur les idées des paragraphes précédents, nous définissons le graphe non orienté pondéré $G^o = (S, A^o, w^o)$ appelé graphe des opérons. Deux noeuds de S sont reliés par un arc a de poids $w^o(a) = k$ si les gènes associés aux noeuds sont exactement k fois dans le même opéron. On construit le graphe G^o_α en supprimant du graphe G^o les arcs de poids inférieur à α. **Les über-operons sont les composantes connexes de $\mathbf{G}^\mathbf{o}_\alpha$.**

9.4 Conclusion

Nous avons présenté un concept plus souple de clusters de gènes communs à plusieurs génomes. L'utilisation des paramètres permet d'adapter notre méthode à diverses situations de complexité différente. Nous pouvons traiter les cas où un grand nombre de génomes sont simultanément pris en compte car nous n'imposons pas aux motifs d'être présents dans tous les génomes considérés. L'adaptation aux *über-operons* permet d'identifier des structures abstraites mais fonctionnelles comme l'*über-operon* de la flagelle chez les bactéries.

Références

[1] Anne Bergeron. Similarity vs. distance in whole genome comparison. *Mathematics of Evolution and Phylogeny Workshop*, 2003.

[2] Anne Bergeron, Sylvie Corteel, and Mathieu Raffinot. The algorithmic of gene teams. *Bioinformatics*, 2002.

[3] Steffen Heber and Jens Stoye. Finding all common intervals of k permutations. *Lecture Notes in Computer Science*, 2001.

[4] Warren C. Lathe, Berend Snel, and Peer Bork. Gene context conservation of higher order than operons. *Trends in Biochemical Sciences*, 2000.

[5] Ross Overbeek, Michael Fonstein, Mark D'Souza, Gordon D. Pusch, and Natalia Maltsev. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA*, 1999.

[6] Rogozin, Makarova, Murvai, Czabarka, Wolf, Tatusov, Szekely, and Koonin. Connected gene neighborhoods in prokaryotic genomes. *Nucleic Acids Res.*, 2002.

10 Calcul linéaire de toutes les périodes locales d'un mot

Auteurs :

Jean-Pierre Duval (LIFAR, Univ. de Rouen),

Roman Kolpakov (Department of Computer Science, Univ. of Liverpool),

Gregory Kucherov (INRIA/LORIA, Nancy),

Thierry Lecroq (LIFAR/ABISS, Univ. de Rouen) et

Arnaud Lefebvre (UMR 6037/ABISS, Univ. de Rouen)

Contact `Thierry.Lecroq@univ-rouen.fr`

Nous présentons un algorithme qui calcule *toutes* les périodes locales d'un mot en temps et espace linéaire. Une période locale en une position i sur un mot w est définie comme étant la longueur du plus court carré centré sur la position i (avec possibilité de débordement en début et/ou en fin du mot w). Notre algorithme calcule, dans une première phase, les longueurs des plus courts carrés internes (ceux qui ne débordent pas) en chaque position de w en utilisant la s-factorisation du mot et des fonctions d'extension. Dans une seconde phase, il calcule les longueurs des plus courts carrés externes (ceux qui débordent) en utilisant une version simplifiée de la fonction de décalage de l'algorithme de recherche de mot de Boyer-Moore. Ce résultat permet de retrouver à la fois la période globale et les factorisations critiques du mot w.

11 Conception d'amorces pour la PCR multiplexe

Auteurs :

François Nicolas (LIRMM, Univ. Montpellier II) et

Eric Rivals (LIRMM, Montpellier CNRS)

Contact : `nicolas@lirmm.fr`

Nous considérons le problème de minimisation suivant :

MINIMUM FACTOR COVER	
ENTRÉE :	Un ensemble fini S de séquences et un entier naturel L.
SOLUTION :	Tout ensemble fini C de séquences de longueurs L tel que tout mot de S admette un facteur dans C.
MESURE :	Le cardinal de C.

Ce problème constitue une modélisation naturelle du problème de la conception d'amorce pour la PCR multiplexe [4].

Il est équivalent de plusieurs point de vues (paramétrique, approximation, ...) au problème MINIMUM SET COVER ce qui n'est pas une bonne nouvelle [3, 1, 2]. Plus précisément, nous montrerons

que MINIMUM FACTOR COVER est NP-complet, W[2]-complet pour le paramètre $\#C$ et inapproximable à un facteur constant. La seule bonne nouvelle est l'existence d'une heuristique gloutonne réalisant en temps polynomial une approximation de MINIMUM FACTOR COVER de borne $1 + \ln \#S$ et c'est le mieux que l'on peut espérer.

Références

[1] M. Bellare, S. Goldwasser, C. Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the Twenty-Fifth Annual A.C.M. Symposium on Theory of Computing*, pages 294–304, 1993.

[2] Marco Cesati. Compendium of parameterized problems.

[3] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the A.C.M.*, 45(4) :634–652, 1998.

[4] W. R. Pearson, G. Robins, D. E. Wrege, and T. Zhang. A new approach to primer selection in polymerase chain reaction experiments. *Discrete and Applied Mathematics*, 71(1–3) :231–246, 1996.