Indexer un ensemble de séquences ADN annotées

Tatiana Rocher, Mathieu Giraud, Mikael Salson

Indexer un ensemble de séquences ADN annotées

Tatiana Rocher [*1], Mikael Salson [†1], Mathieu Giraud [‡1]

[1] Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL) – INRIA – Université Lille 1, Bâtiment M3 extension Avenue Carl Gauss, F-59 655 Villeneuve d'Ascq, France

Session séquences nucléiques
mardi 28 16h30
Salle des thèses

Les structures d'indexation sont largement utilisées en bioinformatique, par exemple pour rechercher de courtes séquences dans un génome (BWA, Bowtie). Mais comment prendre en compte des annotations lors de cette recherche ? Nous souhaitons indexer un ensemble de séquences ADN annotées, telles que les séquences recombinées codant pour les récepteurs lymphoïdes.

Cet article présente une méthode d'indexation compressée de séquences ADN annotées permettant une recherche rapide d'informations grâce à la transformée de Burrows-Wheeler ainsi qu'un Wavelet Tree.

Introduction : recherches de séquences annotées

L'immunité adaptative est le mécanisme par lequel notre corps se défend de manière spécifique aux infections. Les lymphocytes B et T jouent un rôle clé dans cette immunité. À plusieurs endroits de leur ADN, ces cellules effectuent un mécanisme de recombinaison V(D)J, mécanisme donnant des milliards de possibilités différentes à partir d'un répertoire de quelques gènes. Un réarrangement V(D)J est par exemple V402 4/ACTG/0 J112. Cela signifie que la séquence ADN est composée :

— du gène V402 dont les 4 dernières lettres ont été tronquées,
— de la séquence ACTG,
— et du gène J112.

La diversité des réarrangements V(D)J implique la diversité des récepteurs des lymphocytes T tout comme des anticorps produits par les lymphocytes B.

Aujourd'hui, grâce au séquençage à haut-débit couplé à une analyse bioinformatique, il est possible d'avoir une vue précise d'une population de lymphocytes à un instant donné. Plusieurs logiciels, comme IMGT HighV-QUEST, IgBlast, Vidjil ou MiXCR peuvent quantifier les populations de lympocytes suivant leurs recombinaisons V(D)J, en immunologie fondamentale tout comme en hématologie des leucémies. En effet, le diagnostic et le suivi des leucémies aigües demandent d'analyser lymphocytes et lymphoblastes suivant leur recombinaisons V(D)J.

Pour mieux analyser la population d'un patient ou comparer des populations de lymphocytes de plusieurs patients, nous avons besoin d'une vue globale de l'ensemble des réarrangements VDJ du patient ainsi que d'une fonction de recherche des séquences très rapide. Est-il possible de réaliser un index stockant des séquences d'ADN (telles que ATGCGAT…CGATCGA, de taille 96) munies d'annotations (telles que V402 aux positions 1-64 et J112 aux positions 68-96) ?

Parmi les structures de données déjà existantes, une structure de type tableau des suffixes [9] ou arbre des suffixes [8], est trop volumineuse sur les données utilisées. La transformée de Burrow-Wheeler [3] et la compression LZ-77 [7] sont de très bons outils de compression de fichiers, mais l'entropie du texte augmente lorsque nous y ajoutons les annotations et sa compression devient moins bonne. Dans le cas des séquences de recombinaison VDJ, les séquences de gènes sont trop différentes pour pouvoir utiliser un FM-index for similar string [6].

*. Intervenant
†. Corresponding author : mikael.salson@univ-lille1.fr
‡. Corresponding author : mathieu.giraud@univ-lille1.fr

Nous proposons ici un nouvel index, inspiré d'un FM-index [5], permettant de stocker de telles séquences avec leurs annotations et de répondre en temps linéaire à des requêtes d'association annotation/séquence.

Méthodes

La structure

Notre structure utilise une transformée de Burrows-Wheeler (BWT) pour sauvegarder les séquences, un Wavelet Tree (WT) pour leur associer leurs annotations, et un vecteur de bits pour lier les deux.

La transformée de Burrows-Wheeler (BWT) [3] est un algorithme permettant de réorganiser les lettres d'un texte. Le texte transformé est la concaténation de la dernière lettre de chacune des rotations du texte, triées dans l'ordre lexicographique. Cette transformée sans perte d'information permet une lecture du texte aussi rapide que sur le texte original, et le texte transformé peut mieux se compresser.

Le Wavelet Tree (WT) [4] est un arbre. Les feuilles stockent une information. Les nœuds internes sont composés d'un vecteur de bits. Les bits 0 indiquent que la feuille correspondant à l'information de cette position se trouve dans le sous-arbre gauche, les bits 1, dans le sous-arbre droit. Cet arbre permet de rassembler dans une feuille toutes les occurences d'une information et de faire des recherches facilement.

Nous appelons annotation l'information que nous souhaitons ajouter comme commentaire sur une partie du texte. L'annotation doit être associée à une portion des lettres du texte (par exemple : annotation 1 allant des lettres 12 à 54). De plus, ces annotations peuvent être hiérarchisées. Par exemple, pour les recombinaisons VDJ, une annotation peut être une famille de gènes (V), un gène (V2), ou un allèle (V201).

Soit T, le texte de taille n incluant l'ensemble des séquences que nous souhaitons stocker. Nous transformons T avec la transformée de Burrow-Wheeler : BWT(T). Puis nous ajoutons un vecteur d'annotations A correspondant aux annotations des lettres de BWT(T) : $A[i] = a_{\{}\text{BWT}(T)[i]\}$, avec i étant une position dans A et a_j étant l'annotation de la lettre j. (Le vecteur A n'est utilisé que lors de la construction). Une même annotation est souvent sur un même facteur, donc après application de la tranformée, plusieurs ensembles d'annotations sont consecutifs. Nous associons au texte transformé un vecteur de bit B de taille n avec les conditions suivantes : $B[i] = 0$ si $A[i] = A[i-1]$, $B = 1$ sinon, et $B[0] = 1$. Chaque annotation correspondant à un bit 1 est mise dans la racine du WT. Nous ajoutons aux feuilles de notre WT un entier indiquant le nombre d'apparitions de l'annotation dans le texte.

Optimisation du WT

Nous mettons deux contraintes sur le WT : optimiser la structure de l'arbre pour avoir des requêtes plus rapides, et hiérarchiser les annotations. Afin d'optimiser le temps de réponse à une requête, nous utilisons la forme d'un arbre de Huffman [2]. Les feuilles des annotations les plus fréquentes sont placées plus haut dans l'arbre.

La hiérarchie des annotations des recombinaisons VDJ se représente par un arbre k-aire. Les annotations ayant la même précision, par exemple V, D et J, sont à la même profondeur de l'arbre. Le sous-arbre descendant du nœud V ne contient que des nœuds de la catégorie V ayant une meilleure précision : V1, V2*01…

Nous transformons l'arbre k-aire (muni des fréquences d'apparition des nœuds) en WT, arbre binaire, en lui appliquant au maximum une structure de Huffman. Pour cela, nous parcourons les différentes profondeurs de l'arbre, de la plus profonde vers la moins profonde, pour organiser les fils du nœud courant en utilisant une structure d'un arbre de Huffman.

Tous les vecteurs de bits (le vecteur lien et les vecteurs du WT) sont associés aux fonctions rank et select. La fonction rank_b(V, i) renvoie le nombre de fois où le bit b apparait dans le préfixe [1, i] du vecteur de bits V. La fonction select_b(V, j) renvoie la position i du jième bit b dans le vecteur de bits V. Toutes deux s'effectuent en temps constant dans le vecteur [10].

Taille de la structure

Soient n, la taille de l'ensemble des séquences, l, le nombre d'annotations présentes dans l'ensemble des séquences analysées, et z, le nombre d'annotations différentes de l'annotation précédente dans la BWT.

La BWT produit un texte dont la compression varie en fonction de l'entropie du texte T. La taille du texte compressé est donc n * H_k(T) avec H_k(T) étant l'entropie d'ordre k du texte T. L'arbre a une taille totale de z * log(l). La taille totale de la structure est alors : O(n*H_k(T)+ n + z * log(l)) Dans le pire des cas, l'ensemble des annotations est utilisé et, après transformation du texte, deux annotations adjacentes ne sont jamais identiques, donc z=n. Dans ce cas là, la racine du WT est de taille n.

En pratique, le répertoire immunologique d'un individu présente souvent une ou des recombinaisons VDJ présentes en plusieurs exemplaires : entre 1 % et 5 % chez un individu sain, et entre 5 % et 80 % chez une personne atteinte de leucémie. La taille théorique est donc largement surestimée. Par exemple, pour un texte constitué de 10^6 lettres, la structure aura une taille maximum de 13×10^6 bits (2 bits par lettres pour la BWT, 1 bit par lettre pour le vecteur lien et un WT maximum : 10 fois la taille du vecteur lien), mais nous pouvons espérer une taille de 10^6 bits (une compression de 50 % lors de la BWT, des annotations économisant 9 lettres sur 10 et certaines annotations très présentes donnant un WT très déséquilibré).

Requêtes possibles

Cette structure permet de répondre efficacement à plusieurs requêtes liant séquences et annotations.

Soit les abréviations suivantes :
— r : taille d'une séquence,
— l : nombre total de feuilles/annotations,
— b_f : nombre de bits sur la feuille correspondant à l'annotation f (au plus z).

1) Le nombre de séquences possédant une annotation : ce nombre est connu par un simple accès à l'entier stocké dans la feuille possédant cette annotation. Cette recherche se fait en temps $O(\log(l))$. Cette requête permet d'avoir une vue d'ensemble des proportions de présence des annotations dans l'ensemble des séquences, comme la proportion de séquences présentant le gène V4.

2) L'ensemble des annotations associées à une séquence S appartenantà T : pour chaque lettre de S, nous cherchons l'annotation correspondante dans le WT. La complexité totale est : $O(r \times log(l))$. Nous pouvons éviter certains parcours de l'arbre en recherchant l'annotation d'une lettre toutes les x lettres, et ainsi réduire la complexité. De plus, nous pouvons arrêter la recherche de l'annotation d'une séquence dans un nœud interne de l'arbre lorsque nous aurons obtenu le niveau de précision voulu de l'annotation.

3) La liste des séquences possédant une annotation f : nous récupérons tous les bits de la feuille associée à l'annotation f. Puis nous remontons l'arbre pour chacun de ces bits pour trouver les séquences associées à ce bit. Enfin, nous parcourons la séquence dans la BWT pour trouver son identifiant. La complexité totale est : $O(b_f * log(l) + (r * y))$, avec y étant l'ensemble des lettres possédant l'annotation f. Cette fonction a une complexité assez grande principalement à cause de la taille de b_f et à la redondance des séquences trouvées à lire. Cette requête permet par exemple de récupérer toutes les séquences ayant le gène V4, pour les aligner et les étudier.

Perspectives

Nous avons proposé une structure permettant de stocker des séquences ADN ainsi que des annotations associées à certaines positions. Appliquée aux recombinaisons VDJ, cette structure permet d'avoir une vision d'ensemble du répertoire immunitaire d'une personne, en connaissant l'ensemble des séquences lymphoïdes ainsi que les gènes qui les composent. Nous allons à présent procéder à l'implémentation de la structure de données, à l'évaluation de celle-ci sur des données réelles, et optimiser certains algorithmes de la structure pour les rendre spécifiques aux données VDJ, en particulier pour pouvoir rechercher toutes les séquences qui ont une annotation spécifique telle que V102/D3/J402. Nous souhaitons ensuite donner une estimation de la qualité des données à l'utilisateur. Nous voulons pouvoir lui indiquer les différences entre les gènes présents sur les séquences analysées et ceux de référence proposés par IMGT/GENE-DB. Couplée au logiciel Vidjil [1], cette structure de donnée sera utilisée pour faire des comparaisons de répertoire immunologique intra- et inter-patients.

Références

[1] M. Giraud, M. Salson, et al.. Fast multiclonal clusterization of V(D)J recombinations from high-throughput sequencing. *BMC Genomics*, vol 15, 2014.

[2] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 1952.

[3] M. Burrows, D. J. Wheeler. A block-sorting lossless data compression algorithm. *Digital SRC Research*, 1994.

[4] R. Grossi, A. Gupta et J.S. Vitter. High-order entropy-compressed text indexes. *Symposium on Discrete Algorithms (SODA 2003)*, 2003.

[5] P. Ferragina et G. Manzini. Opportunistic Data Structures with Applications. *Symposium on Foundations of Computer Science (FOCS 2000)*, 2000.

[6] J.C. Na, H. Kim, H. Park, T. Lecroq, M. Léonard, L. Mouchard, K. Park. FM-index of alignment: A compressed index for similar strings. *Theoretical Computer Science*, 2015.

[7] J. Ziv et A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, volume 23, numéro 3, pages 337-343, 1977.

[8] P. Weiner. Linear pattern matching algorithms. *IEEE Symposium on Switching and Automata Theory*, 1973.

[9] U. Manber et G. Myers. Suffix arrays: a new method for on-line string searches. *Symposium on Discrete Algorithms (SODA 1990)*, 1990.

[10] G. Jacobson. Space-efficient static trees and graphs. *Symposium on Foundations of Computer Science (FOCS 1989)*, 1989, pages 549–554.

[11] IMGT. Base de données IMGT des gènes VDJ.`http://www.imgt.org/genedb`, Accessed: 2016-04-22.

Mots clefs : indexation, transformée de Burrows, Wheeler, Wavelet Tree, immunologie, hématologie, recombinaison VDJ, répertoire immunitaire