G-011

ハッシュ関数を用いた色付き de Bruijn グラフによる DNA シーケンスリードのインデックス
Indexing DNA Sequencing Reads with Colored de Bruijn Graph Using Hash Function

長谷川 望† 清水 佳奈†
Nozomi Hasegawa Kana Shimizu

1. はじめに

色付き de Bruijn グラフは，文字列間のオーバーラップを表した有向ラベル付きグラフである de Bruijn グラフに対して，グラフ上の不正確なノード遷移を防ぐために色という付加情報を与えたものである．色は入力配列に対して特定のものが割り当てられ，各入力配列に対応するノードやエッジに記録される．例えば DNA シーケンスリードのセットについて，リードごとに異なる色を割り当ててノードに記録することで，元のリード情報に準じたノード遷移が可能となるため，ゲノムアセンブリに基づく様々な解析の精度向上が望める．また解析の効率を上げるため，一般的にノード遷移などの操作はグラフ情報を元にした索引を用いて高速化を図る． Díaz-Domínguez ら[1]はこの問題に取り組み，実用的な索引作成アルゴリズムを提案した．

しかし，Díaz-Domínguez らの手法では複数の後継ノードが同じ色をもつ場合があり，途中でノード遷移に失敗するリード（以下，**曖昧なリード**と呼ぶ）が多く存在する問題がある．例えば図 1 (a) 左のように，ノードのラベル長未満の長さのリピートを含むリード α の場合，ノード B の 2 つの後継ノード C, D が同じ色を持ち，どちらに進むのが正しいのか判断する情報がないため，ノード遷移が B で止まってしまう．また図 1 (a) 右のように，リード β の複数のノード $G \rightarrow H \rightarrow I$ を他のリード γ のエッジが短絡している場合も，同様にノード G 以降遷移を行うことができなくなる．この問題の重大な点は，単に特定のリード関して正確なノード遷移ができなくなるというだけではない．ゲノム上の同じ領域に由来するリードはほとんどの場合グラフ上の同じ部分を共有するため，ある特定の領域に関して解析を行うことができなくなってしまう．

そこで本研究では，1 つのリードに対して 1 つの色を割り当てるのではなく，図 1 (b) で示すように，ハッシュ関数を用いてリードの途中で色を変化させていくことで辿るべき後継ノードを一意に定める索引法を提案し，この問題の解決を図る．

2. 提案手法

DNA 配列 R はアルファベット $\Sigma = \{A, C, G, T\}$ 上の文字列である．便宜上 Σ には他のどの文字よりも小さいダミー記号 $を加えている．$R = \{R_1, R_2, \ldots, R_n\}$ を n 個の DNA シーケンスリードのセットとし，$R' = \{R_1, R_{rc1}, \ldots, R_n, R_{rcn}\}$ を R のリードとそれらの逆相補的リードのセットとする．出次数が 2 以上の前置ノードを持つノードをクリティカルノードと呼び，$R_i \in R'$ の長さ k 部分文字列 $R_i[i..i+k-1]$ を $k-mer$ と呼ぶ．実際のグラフ構造は， de Bruijn グラフ

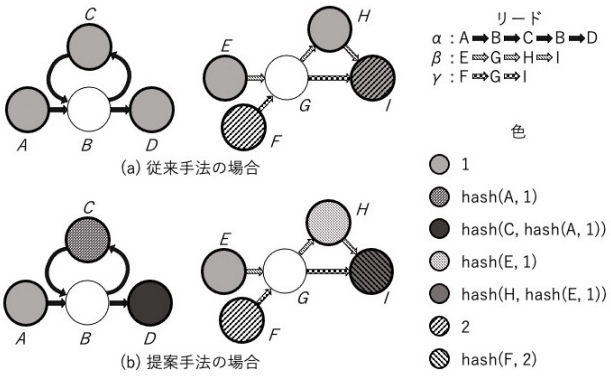

図 1　従来手法と提案手法との比較

の簡潔な表現である，BOSS データ構造[2]を用いている．

2.1 色を付けるノードの選別

リードに準じたノード遷移を行うための色付けについて，以下の定理が成り立つことが知られている[1]

定理. リード $R_i \in R'$ のパスを正確に辿るためには，$ と R_i の $k-2$ 接頭辞を符号化する開始ノード s_i ，R_i の $k-2$ 接尾辞と $ を符号化する終了ノード e_i，およびパス中のクリティカルノードに色をつけなければならない．

索引のサイズを抑えるため，ノード遷移に最低限色付けが必要な s_i ，e_i ，クリティカルノードのみ色を記録する．

2.2 色の割り当て

各ノードに記録する色は 2 つの値 $val1$ と $val2$ を用いて，以下に示すハッシュ関数から求める．このハッシュ関数は C++ライブラリ boost[3]を参考に実装した．

```
uint64_t hash(uint64_t val1, uint_64t val2) {
  uint64_t seed = 0;
  seed ^= val1 + 0x9e3779b9 + (seed << 6) + (seed >> 2);
  seed ^= val2 + 0x9e3779b9 + (seed << 6) + (seed >> 2);
  return seed;
}
```

リード $R_i \in R'$ について，まず s_i のノード ID を $val1$，R_i の処理順番号を $val2$ としてリードの初期色を求める．求めた初期色がすでに s_i に記録された色と被っていないかを確認し，被っていた場合はs_i のノード ID を $val1$，現在の初期色を $val2$ として求め直し， s_i に記録する．また，s_i のノード ID を $val1$，記録した初期色を $val2$ として色を求め，以降のリードの色とする．続いて R_i の $k-1-mer$ に対応するノードを順に確認していく．ノードがクリティカルノードだった場合には，現在のリードの色をノードに

† 早稲田大学大学院基幹理工学研究科 Graduate School of Fundamental Science and Engineering, Waseda University

Algorithm 1　リードへの色の割り当て

for each $R_i \in R'$ **do**
  $R_i \leftarrow \$R_i\$$
  $s_i \leftarrow \mathrm{labelToNode}(R_i[1..k-1])$
  $color \leftarrow \mathrm{hash}(s_i, i)$
  while $color$ is contained in s_i **do**
    $color \leftarrow \mathrm{hash}(s_i, color)$
  for $j = 1$ to $|R_i| - (k-2)$ **do**
    $v \leftarrow \mathrm{labelToNode}(R_i[j..j+k-2])$
    if v needs to be colored **then**
      $\mathrm{storeColor}(v, color)$
      $color \leftarrow \mathrm{hash}(v, color)$

記録し，現在のノードの ID を $val1$，現在のリードの色を $val2$として新たな色を求め，リードの色を更新する．これを e_i まで繰り返す．この色付けアルゴリズムの疑似コードを Algorithm 1 に示す．

ノードへの色の記録は，全ノード数を nとして，ノードの色付けの有無を表す長さ n のビットベクトル N，各ノードの色をノードごとにソートして Δで表現し結合したリスト M，M 中の各ノード最初の要素を表す長さ $|M|$ のビットベクトル F を用いて表現する．ノード v の色は，rank と select 操作を用いて以下のようにして取得できる．

1. 色付きノード内の v のランク$r = \mathrm{rank}_1(N, v)$を求める．
2. $[i,j] = (select_1(F, r), select_1(F, r+1) - 1)$ より，v の M 上の範囲 $[i..j]$ を得る．
3. 範囲 $[i..j]$ で先頭からインクリメンタルに色を再構築．

2.3 曖昧なリードによる下流解析への影響

提案手法は，曖昧なリードを完全になくすことは保証しない．あるノード X を共有し，異なるX の後継ノード Y_1, Y_2 をもつ 2 つのリード R_1, R_2 について，Y_1, Y_2それぞれの 1 つ前のクリティカルノードで同じハッシュ値が生成された場合，Y_1, Y_2に同じ色が記録されるため，X 以降ノード遷移が出来なくなり曖昧なリードとなる．しかし提案手法の場合，ハッシュ値の衝突は特定のリードに対してのみ発生し，グラフ上の同じ場所を通る別のリードは正しく遷移することが期待できるため，従来手法と異なりゲノム上の特定の領域由来のリードが全て曖昧になってしまうわけではない．よって，従来手法に比べて下流解析への影響は小さいと考えられる．またこの問題は，より衝突の起きにくいハッシュ関数を実装することで，軽減することができる．

3. 実験

提案手法の性能確認のため，ヒト 21 番染色体のシミュレーションリードセットを用いて，従来手法[1]との比較実験を行った．また，文字列上の様々な操作を備えた Suffix Tree はゲノムアセンブリに用いることもでき，元のリードをすべて記録するため，提案手法と同様に元のリードに忠実な解析を行うことができる．したがって，本稿では[1]に加えて，Suffix Tree との比較も行った．リードセットには，生成した 100 塩基長のリード 10,000,000 本と，各リードの逆相補鎖を合わせた計 20,000,000 リードが含まれている．一般的に解析はシーケンスエラーを除去した後に行われるため，N のシンボルやシーケンスエラーは含まれていない．また，ノードのラベル長 k の値は $k = 30$ とした．

表 1　索引作成の結果

手法	実行時間 [second]	メモリピーク [MB]	索引サイズ [MB]
Suffix Tree	826	9632.1	4066.2
従来手法	3624	9632.1	487.5
提案手法	2829	9632.1	2652.7

表 2　リードの復元結果

手法	曖昧なリード数
従来手法	122938
提案手法	0

提案手法と Suffix Tree 作成のプログラムは SDSL-lite ライブラリ[4]を用いて C++で実装した．すべての実験は Intel(R) Core(TM) i7-7820X CPU@3.60GHz，125GB の RAM を搭載したマシンを用いて実行した．それぞれの手法について，索引の作成を行い，実行時間，メモリ使用量とデータサイズを計測した．索引作成における着色ステップは 16 スレッドで行った．その結果を表 1 に示す．また，従来手法と提案手法について，どれだけ正確にノード遷移ができるかを確認するため，作成した索引を用いて元になったリードの復元を行ない，復元できない曖昧なリードの数を計測した．その結果を表 2 に示す．

提案手法による索引の構築は，従来手法よりもおよそ 1.3 倍高速であった．また提案手法による索引は Suffix Tree を用いた索引よりもおよそ 1.5 倍良い空間効率を示している．曖昧なリードについては，従来手法が 0.6%のリードを復元できなかったのに対し，提案手法では全てのリードを復元することができた．

4. おわりに

本研究では，色の割り当てを行う際に，ハッシュ関数を用いてリードの途中で色を変化させることで，曖昧なリードの生成を防ぐことのできる，色付き de Bruijn グラフを用いた索引の作成手法を提案した．実験結果より，1 つのリードにつき 1 つの色を割り当てていた従来手法と比較して，曖昧なリードの数を大幅に削減することができた．また，Suffix Tree を用いた索引と比べ，空間効率が良いことが確認できた．この手法により，従来手法では復元が難しかった領域の解析に寄与することが期待できる．

今後，全ゲノムシーケンスデータ等のより大きいデータセットも問題なく扱えるようにするために，衝突耐性の計測を通したハッシュ関数の改善や，色を記録する際の圧縮方法の改善による索引サイズの削減を行なっていく必要がある．

参考文献

[1] D. Díaz-Domínguez, “An Index for Sequencing Reads Based on the Colored de Bruijn Graph.", International Symposium on String Processing and Information Retrieval, Springer, Cham (2019).
[2] A. Bowe, T. Onodera, K. Sadakane, T. Shibuya, "Succinct de Bruijn graphs.", International workshop on algorithms in bioinformatics, Springer, Berlin, Heidelberg, (2012).
[3] https://www.boost.org/
[4] S. Gog, T. Beller, A. Moffat, M. Petri, “From theory to practice: Plug and play with succinct data structures.”. International Symposium on Experimental Algorithms (SEA)., Springer, Cham, (2014)