Vol.2018-BIO-53 No.3
2018/3/9

Short *k*-mer データベースを用いた *k*-mer カウント処理についての検討

加治原翔太[†1]　外山史[†1]　森博志[†1]　東海林健二[†1]

概要：大規模ゲノムの解読は，生命システムの解明だけでなく，医療科学，薬学，農学などの多様な応用が考えられ，様々な種を対象にゲノム解読の研究が行われている．ゲノム解読はシーケンサから得られたデータを元に行われるが，近年，次世代シーケンサの登場によりシーケンシングのコストパフォーマンスは飛躍的に向上し，短時間で大量のデータを生成できるようになった．次世代シーケンサは対象の塩基配列の短い断片(リード) を大量に出力するため，それを正しく並べ替えて元の塩基配列の決定するためのアルゴリズムが必要になる．そのようなアルゴリズムは *de novo* アセンブリアルゴリズムと呼ばれ，様々な手法が提案されている．特に Velvet は消費メモリや計算時間などについてパフォーマンスに優れており，コンティグの精度も比較的高いため，最も普及しているアセンブリ手法の一つとなっている．しかし，リードの総量が数十 Gbp(bp：塩基対) を越える大規模なデータのアセンブリを行う場合，Velvet を用いても実行時に要求されるメモリが非常に膨大になってしまいメモリ不足となってしまう．

この問題を解決する手法として、遠藤らによる消費メモリの少ない *de novo* アセンブリアルゴリズムがある．遠藤らの手法は，1 つの *k*-mer に対するデータ量を小さくすることで最大メモリ使用量を大幅に削減した手法である．*k*-mer とは，リード内に含まれる *k* 文字の塩基配列を指す．一方，遠藤らの *de novo* アセンブリにおいて *k*-mer カウント処理が最もメモリを使用する．そこで，本研究では *k*-mer カウント処理に焦点を当てて遠藤らの手法を改良する．実験では，ヒト 14 番染色体から得られたリードに対して，本手法と遠藤らの手法及び DSK との比較を行った．

1. はじめに

DNA の全塩基配列を決定することは，生物化学の分野だけでなく医学や薬学など多種多様な方面で活用ができるため重要視されている．現在，DNA の全塩基配列の決定には，次世代シーケンサが用いられる．次世代シーケンサでは，DNA の短い塩基配列の断片であるリードを短時間で大量に得ることが可能である．また，塩基配列の決定には取り出したリードを繋ぎ合わせる必要がある．この繋ぎ合わせの処理は，*de novo* アセンブリと呼ばれる．リードを繋ぎ合わせる際，*k*-mer と呼ばれるリードに含まれる *k* 文字の塩基配列を取り出し，各 *k*-mer に対して全リード中に同じ *k*-mer がいくつあるかをカウントする必要がある．この処理を本研究では *k*-mer カウント処理と呼ぶ．ここで，次世代シーケンサは，大量のリードを安価に取り出すことができるが，*k*-mer カウント処理に非常に多くのメモリを消費してしまうという問題点がある．例えば，ヒトゲノムのような大規模ゲノムにおいては，*k*-mer カウント処理に数テラから数十テラバイトのメモリが必要となる．

この問題を解決するため，これまでに遠藤らによる手法[1]が提案されている．遠藤らによる手法は，1 つの *k*-mer に対するデータ量を小さくすることで，アセンブリにおける最大メモリ使用量を大幅に削減する手法である．また，*k*-mer カウント処理のメモリ削減手法として，DSK[2]が提案されている．DSK は，取り出したリードをメインメモリ上に保存するのではなく，ハードディスク上に保存することで，大幅なメモリ削減を実現した手法である．しかし，DSK では消費メモリは少ないが全ての *k*-mer をハードディスク上に保存しなければカウントを行うことができない．そのため，大規模ゲノムを実行する場合，ハードディスク上に膨大な量の情報をすべて書き込まなければならなく，主記憶装置と補助記憶装置間のアクセスが増えるという問題点がある．

そこで本研究では，遠藤らの手法における *k*-mer カウント処理を改良した手法を提案する．*de novo* アセンブリには，出現回数が著しく少ない *k*-mer は，繋ぎ合わせの処理に使用しないという特徴がある．本手法では出現回数が著しく少ない *k*-mer の一部を登録しないことで，最大メモリ使用量を更に削減した．

実験では，ヒト 14 番染色体から得られたリードに対して，本手法と遠藤らによる手法及び DSK との比較を行った．その結果，遠藤らの手法に比べ実行時間が約 13%減少し，最大メモリ使用量が約 67%減少した．また，DSK との比較では実行時間が約 15％減少した．

2. *k*-mer カウント処理

2.1 *k*-mer

DNA の塩基配列を決定する際，全てのリード内に含まれている *k*-mer と呼ばれる *k* 文字の塩基配列を取り出す必要がある．例えば，ATTCGGC という 7bp(base pair:塩基対)のリードがある．ここから，5-mer(5 文字の塩基配列)を取り出すとすると，ATTCG，TTCGG，TCGGC の 3 種類の 5-mer を得ることができる．実際の *de novo* アセンブリでは，各 *k*-mer が全リード中にいくつあるかをカウントし，出現回数が多い *k*-mer のみを用いて繋ぎ合わせの処理が行われる．したがって，出現回数の少ない *k*-mer は以降の処理では必要ない．しかし，出現回数をカウントするためには全てのリードに含まれる *k*-mer を調べる必要があるため，一般的にはリードに含まれる *k*-mer は全てメインメモリ上に保存される．

†1 宇都宮大学
Utsunomiya University

2.2 *k*-mer 整数

k-mer 整数とは, 各 *k*-mer に一対一で対応づけた整数である．具体的には，各塩基 A，C，G，T に対しそれぞれ 0，1，2，3 の数値を割り当てて塩基配列を 4 進数で表現し，その 4 進数を 10 進数で表した値が *k*-mer 整数となる．したがって，*k*-mer 整数は「0～4^k-1」までの整数に対応している．*k*-mer 整数を用いる利点として, 塩基配列を文字列として扱う場合よりメモリ量が少ないという点がある．また，一般的に計算機で処理を行う場合には，文字の操作よりもバイナリデータの操作の方が高速に処理を行うことができる．したがって，塩基配列の比較や探索を行う場合，*k*-mer 整数を利用することで高速に処理を行うことができる．提案手法においても，この *k*-mer 整数を用いてカウント処理を行っている．

3. 従来手法

本章では，提案手法のベースとなっている遠藤らによる消費メモリの少ない *de novo* アセンブリアルゴリズムの手法について述べる．Velvet[3]や SOAPdenovo[4]といった代表的な *de novo* アセンブリアルゴリズムでは，*k*-mer を登録する際に出現回数だけでなく，どのリードのどの部分から出現したか等の位置情報も登録している．そのため，1 つの *k*-mer に対するデータ量が多くなってしまう．そこで，遠藤らはこの 1 つの *k*-mer に対するデータ量を小さくすることで，アセンブリにおける最大メモリ使用量を削減する手法を提案している．この手法では位置情報は登録せず，出現回数のみの情報でカウント処理から繋ぎ合わせ処理までを行う手法であり，大幅なメモリ削減を実現している．

遠藤らの手法を以下に述べる．この手法では，次世代シーケンサを用いて取り出したリードをメインメモリ上に保存する際に，*k*-mer として保存する．そのため，まずリードから *k*-mer を取り出し，取り出した *k*-mer を *k*-mer 整数に変換する．次に，変換した *k*-mer 整数を，ハッシュ関数を用いてメインメモリ上のハッシュテーブルに登録する．このとき，メインメモリ上のハッシュテーブルには，*k*-mer 整数と出現回数が登録されているレコードへのアドレスが書き込まれている. 最後に, *k*-mer の出現回数の情報を基に de Bruijn グラフを構築し，塩基配列を復元する．以上が，遠藤らによる *de novo* アセンブリアルゴリズムの一連の流れである．

4. 提案手法

4.1 Short *k*-mer データベース

遠藤らの手法において，最もメインメモリを消費する部分は, *k*-mer カウント処理の部分であるため, この処理におけるメモリ使用量を削減することで全体の最大メモリ使用量を削減することができる．本手法は，この *k*-mer カウント処理の部分を改良した手法である．

本手法では，*k*-mer をメインメモリに登録していく過程で，出現回数の少ない *k*-mer を登録せずにカウント処理を行う．*k*-mer の繋ぎ合わせの処理では，3 回以上出現した *k*-mer を使用することが一般的であるが，実際に出現した *k*-mer の中で 3 回以上出現している *k*-mer の種類数は 10～20%である．したがって，出現回数が 2 回以下の *k*-mer の一部を登録しないことで，消費メモリ量を削減することができる．

そこで，各 *k*-mer の出現回数が 3 回以上か否かを判定するために，本手法では，Short *k*-mer データベースを使用している．Short *k*-mer データベースとは，求めたい *k*-mer よりも短い *k*-mer のデータベースであり，全種類の Short *k*-mer の出現回数が格納されている配列である．Short *k*-mer データベースは *k*-mer よりも短い塩基配列のデータベースであるため, 直接 *k*-mer のデータベースを作成するよりも，少ないメモリ量で実装することが可能である．

このデータベースにおいて，出現回数が 2 回以下であった Short *k*-mer が存在した場合，それを含んでいる *k*-mer は 3 回以上出現しないため，ハッシュテーブルに登録する必要が無いことがわかる．この特徴を利用して，出現回数が 2 回以下の *k*-mer の一部を登録しない手法を実現している．

4.2 提案手法

本研究における *k*-mer カウント処理の流れを以下に示す. また，ここで，実験では 3 回以上出現した *k*-mer を登録しているが，以下では説明を簡略化するため，2 回以上出現した *k*-mer を登録するアルゴリズムの説明を行う．また以降では Short *k*-mer を「S*k*-mer」と記述する．

1. S*k*-mer データベースの配列を，メインメモリ上に用意する．
2. リードを読み込む.
3. リードから S*k*-mer を取り出し，*k*-mer 整数に変換する．
4. *k*-mer 整数に対応する S*k*-mer データベース上のカウントを変数「*Y*」に保存した後，S*k*-mer データベース上のカウントを +1 する．
5. 「*Y*>=1」が *k*－S*k*+1 回以上続いていたら 6.を行い，それ以外の場合は 7.を行う．
6. 取り出していた S*k*-mer が末尾となる *k*-mer を，ハッシュ関数を通し登録する．
7. リードから 1 文字右にずらした S*k*-mer を新たに取り出し, *k*-mer 整数に変換し 4.を行う．リードの終端の S*k*-mer の場合には, 次のリードを読み取り 3.を行う．
8. 3.～7.の処理を全てのリードに対して実行することで，*k*-mer カウント処理を行う．

本手法では，S*k*-mer データベースに登録する前に，S*k*-

mer データベースを参照する．ここで，S*k*-mer がデータベースに登録されていなかった場合，S*k*-mer をデータベースに登録後，その S*k*-mer を含む *k*-mer は 1 回しか出現しないと判断し，*k*-mer の登録を行わない．S*k*-mer がデータベースに登録されていた場合，その S*k*-mer を含む *k*-mer の登録のスキップを行わない．これにより，カウント数が 1 減ることになるが，リードの読み込みが 1 回のみで，1 回しか出現しない *k*-mer の大部分を登録せずにスキップすることができる．5. の処理では，取り出している *k*-mer に含まれている全ての S*k*-mer が，それ以前に出現したかどうかを調べている．それによって，「出現回数が 1 回である S*k*-mer を含んでいる *k*-mer を登録しない」という処理を実現している．

具体例を，図 1 を用いて説明する．例では，S*k*-mer を 3-mer，*k*-mer を 6-mer とした．したがって，5. における *k*-S*k*+1 は「6-3+1=4」となるため，5. ～7. は「出現回数 1 回以上の 3-mer を 4 回以上連続して取り出した場合，その 3-mer が末尾となる 6-mer を，ハッシュ関数を通し登録する．」ということになる．また，例では「ATT」は *n* 番目のリードまで 1 度も出現しておらず，その他の 3-mer は全て出現しているとする．

まず，「ACG」「CGC」「GCG」を取り出した段階では *k*-mer（6-mer）はまだ登録されず，「CGA」を取り出したときに初めて *k*-mer が登録される．これは，このリードに「ACG」「CGC」「GCG」を末尾とする *k*-mer が存在しないからである．次に，「GAA」「AAA」「AAT」を取り出しそれぞれを末尾とした *k*-mer を登録する．その後「ATT」～「GGA」を取り出すが，これらを末尾とした *k*-mer には「ATT」が含まれているため *k*-mer の登録は行わない．最後に「GAC」を取り出し，*k*-mer を登録する．このリードに対する処理はここまでであり，次のリードを読み取った後にこの処理を繰り返す．以上が本研究における *k*-mer カウント処理である．

ACGCGAAATTGGAC（*n* 番目のリード）

取り出すSk-mer	カウント数 Y	1回以上が続いた回数	登録するk-mer	登録しないk-mer
ACG	1	1		
CGC	1	2		
GCG	1	3		
CGA	1	4	ACGCGA	
GAA	1	5	CGCGAA	
AAA	1	6	GCGAAA	
AAT	1	7	CGAAAT	
ATT	0	0		GAAATT
TTG	1	1		AAATTC
TGG	1	2		AATTCG
GGA	1	3		ATTCGA
GAC	1	4	TTGGAC	

図 1：本研究における *k*-mer カウント処理の具体例
（Sk-mer を 3-mer，k-mer を 6-mer とする）

4.3 提案手法における *k*-mer カウント処理の精度

4.2 で述べたように，本手法では取り出している *k*-mer に含まれている全ての S*k*-mer が，それ以前に出現したかどうかによって，*k*-mer の登録を行うかの判定を 4. で行っている．つまり，登録される各 *k*-mer の中で最初の *k*-mer は，その後何回出現したかに関わらずカウントされない．それによって，各 *k*-mer のカウント数が実際の出現回数より 1 回少なくなってしまっている．そこで解決策として，各 *k*-mer を初めて登録するとき，つまり各 *k*-mer が 2 回目に出現したときに *k*-mer のカウント数を+2 することが考えられるが，図 2 のような例が多数あるためこの方法を用いたとしても解決できない．

具体例を，図 2 を用いて説明する．まず，最初のリード「ACGCGAAATTGGAC」から「AAA」という S*k*-mer が登録されたとする．この場合，S*k*-mer データベースの「AAA」のカウント数は「1」となる．その後「AAAAAA」という *k*-mer が出現したとする．このとき，本来であれば「AAAAAA」という *k*-mer はまだ出現していないため登録されないが，「AAAAAA」が含んでいる S*k*-mer はすべて「AAA」であり，この「AAA」のカウント数は既に「1」となっているため，「AAAAAA」は登録されてしまう．そのためこの問題は，各 *k*-mer が 2 回目に出現したときに *k*-mer のカウント数を+2 することでは解決できない．よって本手法ではカウント数に最大「-1」の誤差が生じる．

しかし，*k*-mer カウント処理の本来の目的は，各 *k*-mer の大まかな出現回数を調べることで，*de novo* アセンブリアルゴリズムを用いて塩基配列を決定する際の，各 *k*-mer の接合優先順位を決めることであり，詳細な *k*-mer のカウント数を求めることではないため，この誤差は重要ではない．

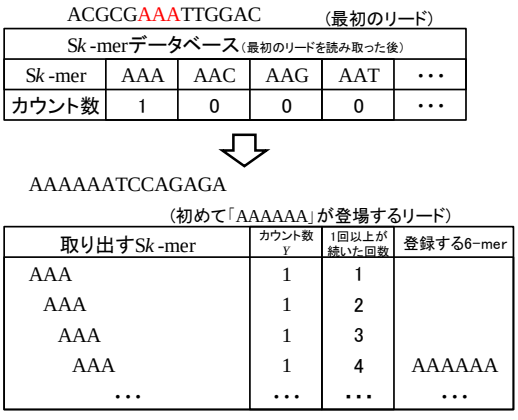

図 2：初登場の *k*-mer が登録されてしまう例
（Sk-mer を 3-mer，k-mer を 6-mer とする）

5. 実験

本実験では，提案手法の有効性を確認するため，遠藤らの手法及び DSK と比較実験を行った． DSK は公開サイトからダウンロードしたものを用い，メモリ使用量と HDD 使

用量のパラメータをデフォルト値に設定し実験を行った．実験では，次世代シーケンサによりヒト 14 番染色体から得られたリード(101bp) を用いた．このヒト 14 番染色体のリードは, GAGE で公開されているものから 100bp 以下のリードを削除したものである．GAGE とは，ゲノムアセンブリを行う際に使用するリードデータを公開しているサイトである．今回使用するリードの数は 60,722,881 である．また *k*-mer は 51-mer，S*k*-mer は 16-mer とし、出現回数が 3 回以上の *k*-mer を登録した．全ての実験は 256GByte のメモリを搭載した Intel Xeon E5-2660 2.20GHz 上で行った．

実験結果を表 1 に示す．表 1 において，最大メモリ使用量は，*k*-mer カウント処理における最大のメモリ使用量を表す．HDD 使用量は *k*-mer カウント処理中で用いた HDD の使用量である．また，HDD は DSK のみ *k*-mer カウント処理の過程において使用するので，提案手法と遠藤らの手法の HDD 使用量は 0 である．

表 1　実験結果

手法	最大メモリ使用量	HDD 使用量	実行時間
提案手法	8.19Gb	0	7 分 6 秒
遠藤らの手法	24.72Gb	0	8 分 8 秒
DSK	3.51Gb	1.83Tb	8 分 17 秒

6. 考察

6.1 遠藤らの手法との比較

本手法では，遠藤らの手法に比べ最大メモリ使用量が約 67 %減少した．これは，4.1 で述べたように出現回数の少ない *k*-mer を登録しなかったからである．

また，本手法では遠藤らの手法に比べ実行時間が約 13%減少した. 本手法では S*k*-mer データベースへの参照や登録を行うため，遠藤らの手法に比べ手順が多くなっている．しかし，本手法では遠藤らの手法に比べ登録を行っている *k*-mer の総数が少ないため，*k*-mer の登録に用いられる時間は短い．よって実行時間が短くなったと考えられる．

6.2 DSK との比較

本手法では，DSK に比べ実行時間が約 15 %減少した．本手法と DSK では使用しているアルゴリズムが全く異なっているため，詳細な比較はできないが，DSK はハードディスクを用いた手法であるためアクセスに時間がかかることに対し，本手法はメインメモリのみを使用した手法であるため，アクセス時間が少ないことが実行時間の差であると考えられる. DSK をメインメモリのみで動かすことができれば，実行時間は DSK のほうが短くなることが考えられるが，その場合大容量のメインメモリを確保する必要があるため，現実的ではない．DSK は，全ての *k*-mer を登録してから *k*-mer のカウントを行い，最後にカウント数の少ない *k*-mer を消去する．そのため，一時的に大容量の記憶領域が必要となっている．それに対し，本手法ではカウント数が少ないと判定された *k*-mer を登録しないため, *k*-mer カウント処理を終えたときに, メモリ使用量が最大となる．そのため，少ない最大メモリ使用量で実行することが可能となっている．

7. おわりに

本研究では，大規模なゲノムのアセンブリを可能にするため，*k*-mer カウント処理における実行時間と最大メモリ使用量の削減方法を提案した．本手法は，Short *k*-mer データベースを用いることで，出現回数が著しく少ない *k*-mer の一部を登録しない手法である．実験では，*k*-mer の値を 51-mer として遠藤らの手法及び DSK と比較して提案手法の有効性を確認した．遠藤らの手法との比較では，実行時間が約 13 %減少し，最大メモリ使用量が約 67 %減少した．また，DSK との比較では実行時間が約 15 %減少した．この結果より本手法は有用と言える．

今後の課題としては，プログラムの並列化による高速化，様々なリードデータに対する考察などが挙げられる．

参考文献

[1] Yuki Endo, Fubito Toyama, Chikafumi Chiba, Hiroshi Mori and Kenji Shoji，“A Memory Efficient Short Read De Novo Assembly Algorithm”，IPSJ Transactions on Bioinfomatics Vol.8 pp.2-8，2015.

[2] Guillaume Rizk， Dominique Lavenier and Rayan Chikhi，“DSK: *k*-mer counting with very low memory usage”，BIOINFORMATICS，pp.652-653，2013.

[3] Zerbino, D. and Birney, E.，“Velvet: Algorithms for de novo short read assembly using de Bruijn graphs,” Genome Res., 18，pp.821-829，2008.

[4] Li, R., Zhu, H., Ruan, J. et al，“De novo assembly of human genomes with massively parallel short read sequencing”，Genome Res., 20, pp.265-272, 2010.