G-021

大規模ゲノム復元のための de novo アセンブリアルゴリズムの開発

De novo Assembly Algorithm for Huge Genomes with Massively Short Read Sequencing

遠藤 友基† 外山 史† 東海林 健二† 宮道 壽一†

Yuki Endo Fubito Toyama Kenji Shoji Juichi Miyamichi

1 はじめに

ヒトゲノムなどの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元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 [1][2] が提案されている．特に Velvet[2] は消費メモリや計算時間などの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．そこで本研究では，次世代シーケンサから得られた大量のデータを用いて，大規模なゲノムに対してもアセンブリが可能となるように，消費メモリの少ない De novo アセンブリアルゴリズムを提案する．

本研究で提案するアルゴリズムは，Velvet と同様に De Bruijn グラフを用いてアセンブリを行うが，実際に構築するグラフでは，どのノードにエッジがあるかという情報は保持せずに，2.2 節でのべる k-mer 整数を用いて効率的にあるノードに対してどのノードとの間にエッジが存在するかを必要な時に調べる．これにより，大量のエッジ情報を保持する必要が無くなるため，大幅な消費メモリの削減が実現できる．又，入力データの文字列をバイナリデータとして扱うことで更なるメモリ削減を行っている．シュミレーション実験では，*E.coli* K-12 strain MG1655 から得られた，仮想的なリード配列に対して本手法と Velvet とでアセンブリを行い，その結果消費メモリが Velvet と比較して約 1/3 に削減が可能ということが確認できた．

2 提案手法

2.1 提案手法の概要

本手法の全体の流れを図 1 に示す．まず，入力となるリードから k-mer の全てのパターンを登録する．ここで k-mer とは，k 文字の塩基配列のパターンのことを指す．この時，各 k-mer のパターンがリード中に出現する回数も登録しておく．次に，登録した k-mer から De Bruijn グラフを構築する．そして，構築したグラフ内の分岐による曖昧さや閉路を除去し，最後にコンティグを生成する．ただし，前章で既に述べたように，グラフを構成する要素の表現や，利用する情報を厳選し余分な情報をメモリ上に保持しないなどの工夫によって，消費メモリを削減を行っている．

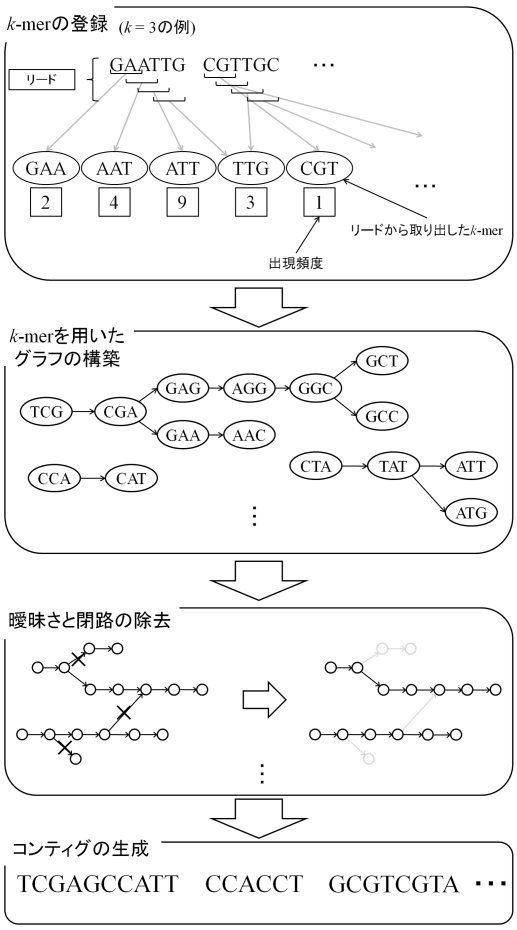

図 1: 全体の処理の流れ

2.2 k-mer の登録

本手法では入力となるすべてのリードからすべての k-mer のパターンを調べ，それらを k-mer 整数としてメモリ上に格納していく．ここで k-mer を整数とは，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 をそれぞれ 0，1，2，3 の数値に対応させて塩基配列を 4 進数として表現し，この 4 進数で表された塩基配列を 10 進数で表した値が k-mer 整数となる．すなわち，各 k-mer は 0 から $4^k - 1$ までの整数に対応している．このような k-mer 整数を用いることで，塩基配列を文字列として処理するよりもメモリ量を削減することができる．更に，一般に計算機では文字の操作よりもバイナリデータの操作の方が高速で行える．そのため，配列同士の比較・探索を行う場合，k 文字の文字列よりも k-mer 整数で扱う方がより高速に行うことができる．本手法では k-mer 整数を登録する際にその k-mer の出現回数も登録する．

2.3 グラフの構築

2.2 節で求めた，全てのリードから取り出した k-mer を用いて，コンティグを求める．具体的には，ある k-mer の 2〜 k 文字目に対して，1〜 $k-1$ 文字目が同じ，すなわち $k-1$ 文

†宇都宮大学大学院 工学研究科

字の重複がある k-mer を探し繋ぎ合わせる処理を繰り返し行うことでコンティグを生成する．ここで，任意の k-mer をグラフのノードとし，あるノードの k-mer を 1 文字分シフトした時に重複するようなノードとの間にエッジを持たせたグラフとして，De Bruijn グラフがある．De Bruijn グラフは，任意のノードが m 種類の文字からなる n 文字列に対応しており，エッジで連結された両ノードの各文字列は $n-1$ 文字の重複を持つという特徴を持つグラフである．従って，k-mer から構築した De Bruijn グラフのパスを探索することにより，コンティグを生成することができる．しかし，k-mer から De Bruijn グラフを構築した場合，グラフ内のパスには曖昧さや閉路が多数含まれているため，始点と終点までのパスが一意の単純パスになることはほとんどない．本手法におけるその解決方法については 2.4 節で述べる．

Velvet などの De Bruijn グラフを用いた手法では，De Bruijn グラフを構築した際に，ノード間のエッジを情報としてメモリ上に保持する必要があるため，多くのメモリを消費してしまう．そこで本手法では，あるノード v の k-mer 整数を左シフトしたものに 0〜 3 を足すことでその k-mer の重複している k-mer を調べ，もし重複している k-mer が存在すればそのノード v' に対して v は有向エッジがあるものとするだけであり，エッジの情報を保持しない．すなわち，グラフ全体を表すために実際にメモリ上に保持するのはノードを表すデータのにみすることで，大幅に消費メモリを削減している．そのため，本手法では有向エッジをあらかじめ調べる必要が無いので，全ての k-mer の登録の完了と共に De Bruijn グラフの構築は完了となる．

2.4 曖昧さと閉路の除去

前節で説明した方法で De Bruijn グラフのパスを求める際，そのグラフには図 2 に示すような分岐や閉路を多数含んでいるため，コンティグを生成する際にどのエッジを選択するかが問題となる．

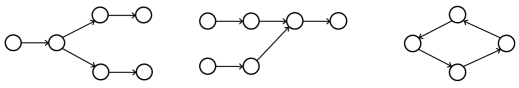

図 2: グラフに現れる分岐の例

図 2(a) は，あるノードから複数の異なったノードへ有向エッジを持っている例である．図 2(b) は，複数の異なったノードからの有向エッジが同一のノードに連結されている例である．図 2(c) は，パスに閉路を含む場合である．実際のグラフでは以上のような分岐のパターンが複雑に組み合わさっており，適切なエッジ及びノードを選択する必要がある．

本手法では，コンティグを生成する前にグラフ全体を走査し，曖昧さの除去を繰り返し行うことで，1 つ以上の連結グラフからなる De Bruijn グラフを構築する．ここで走査とは，まず始点となるノードを選択し，その k-mer の右シフト方向の重複している k-mer を求め，その有無を調べる．もし存在すれば次はそのノードの k-mer のオーバラップを調べ，次々と遷移していく．遷移できるノードが無くなればそこを終点とし，次の始点となるノードを選択し同じ走査を行う．そして通過していないノードが無くなるまで，ノードの走査を繰り返す．ただし，通過したノードには始点のノードに対応する番号をラベルとして与えておく．このラベルを用いて，通過したかどうかの判別や分岐の検出を行う．もし，図 2 に示したような分岐を検出すれば．それぞれに対する処理を行う．具体的には，図 2(a) の場合，複数の分岐先のノードの内 k-mer の出現回数が最も多いものが正しい遷移先とし，他の分岐先のノードを除去する．同様に図 2(b) においても，同一ノードへ有向エッジを持っている複数のノードの内 k-mer の出現回数が最も多いものが正しい遷移元とし，他のノードを除去する．以上のような走査とエッジの選択によって，パスを辿る際の曖昧さを除去し単純パスを得ることができる．又，2(c) については，既に通過したノードを検出した時点でそこを終点とすることで閉路への侵入を回避し，次の走査へ移る．

2.5 コンティグの生成

2.4 節で求めたパスに含まれるノードの k-mer を順に参照すことによりコンティグを生成する．ここで，複数のパスが存在するため，それぞれのパスに対するコンティグを求める．

3 シミュレーション実験

提案手法の性能を確かめるため，シミュレーション実験を行った．実験には *E.coli* K-12 strain MG1655 の全塩基配列からランダムな位置でコピーを行って生成したリードを用いた．又，リードの長さは 35bp とした．ただし，今回はシーケンスエラーを含まない理想的なリードを想定し，リードにエラーを含ませずに実験を行った．今回の実験では k-mer をパラメータ k=21，25，29，33 と変化させ，それに対するメモリ消費量の変化を調べその結果を Velvet と比較した．実験の結果を表 1 に示す．表 1 より，すべての k の場合で消費メモリを削減できていることがわかる．このことから，2.3 節で述べた本手法のグラフ構築方法は有効であると言える．

表 1: 両手法の最大メモリ消費量と実行時間

	既存手法 (Velvet)		提案手法	
k(bp)	最大消費メモリ (MB)	実行時間 (s)	最大消費メモリ (MB)	実行時間 (s)
21	814	160	220	347
25	780	140	220	296
29	780	117	220	227
33	731	82	254	127

一方で，生成されたコンティグについては，最大のコンティグ長と N50 値を比較した結果，ほとんどの場合で Velvet の方が優れていた．具体的には，最大のコンティグ長と N50 値の最大値をそれぞれ比較すると，Velvet では k=21 の時に 73026bp，15624bp となり，提案手法では k=29 の時に 67091bp，20819bp であったが，それ以外の場合ではすべて Velvet が優れた結果であった．これは，グラフにおける曖昧さや閉路の除去方法について，本手法では非常に単純な処理で対応しているのに対し，Velvet では単純に有向エッジを一意に決定するのではなく，より複雑な処理を行うことで対応していることが原因と考えられる．

4 おわりに

本研究では，大規模なゲノムに対してもアセンブリが可能となるように，消費メモリの少ない De novo アセンブリアルゴリズムを提案した．実験の結果，Velvet を用いた場合と比較して，コンティグの精度や計算時間については劣る結果となったが，消費メモリを約 1/3 に削減することが可能ということが確認できた．

今後の課題としては，分岐よる曖昧さや閉路における解決方法を見直し，コンティグの精度を更に高めることが挙げられる．具体的には，k-mer の出現回数だけでなく，リード内の位置やペアエンドなどの情報を用いることで，曖昧さや閉路の除去方法を改善することが考えられる．

参考文献

[1] David Hernandez, Patrice Francois, Laurent Farinelli, et al:“De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer” Genome Res, 2008.

[2] Daniel R. Zerbino and Ewan Birney:“Velvet: Algorithms for de novo short read assembly using de Bruijn graphs” Genome Res, 2008 18: 821-829.