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に関する研究

2015年3月

遠藤友基

宇都宮大学大学院工学研究科
システム創成工学専攻

内容梗概

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．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に対して，大規模なゲノムのアセンブリが可能となる，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．実験では，*E. coli* K-12 strain MG1655 及びヒトの 14 番染色体から得られたリードに対して，本手法と Velvet 及び SOAPdenovo とでアセンブリを行った．その結果，本手法は E. coli に対しては他手法の約 20%, ヒト 14 番染色体に対しては他手法の約 60%の消費メモリ量で *de novo* アセンブリが可能であることを確認した．

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

Yuki Endo

Abstract

Sequencing the whole genome of various species has many applications, not only in understanding biological systems, but also in medicine, pharmacy, and agriculture. In recent years, the emergence of high-throughput next generation sequencing technologies has dramatically reduced the time and costs for whole genome sequencing. These new technologies provide ultrahigh throughput with a lower per-unit data cost. However, the data are generated from very short fragments of DNA. Thus, it is very important to develop algorithms for merging these fragments. One method of merging these fragments without using a reference dataset is called de novo assembly. Many algorithms for de novo assembly have been proposed in recent years. Velvet and SOAPdenovo2 are well-known assembly algorithms, which have good performance in terms of memory and time consumption. However, memory consumption increases dramatically when the size of input fragments is larger. Therefore, it is necessary to develop an alternative algorithm with low memory usage. In this paper, we propose an algorithm for de novo assembly with lower memory. In our experiments using the E.coli K-12 strain MG 1655, the average maximum memory consumption of the proposed method was approximately 20% of that of the popular assemblers. Moreover, in the experiments using human chromosome 14, the average amount of memory of our method was approximately 60% of that of the popular assemblers.

発表論文

- 学・協会誌発表論文

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

- 国際会議等発表論文

1. Yuki Endo, Fubito Toyama, Chikafumi Chiba, Hiroshi Mori and Kenji Shoji, “De Novo Short Read Assembly Algorithm with Low Memory Usage,” BIOINFORMATICS 2014 - Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms, pp.215-220, 2014.

- 学会・研究会等講演論文・ポスター発表

1. 遠藤友基，外山史，東海林健二，宮道壽一，“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，” FIT2011，第 2 分冊，pp.569-570，2011.
2. 遠藤友基，“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，” NGS 現場の会第二回研究会，2012 年 5 月 24 25 日，ホテル阪急エキスポパーク.
3. 遠藤友基，“大規模ゲノム解読のための de novo アセンブリアルゴリズムの開発，” 生命情報科学若手の会第四回研究会，2013 年 3 月 1 3 日，自然科学研究機構岡崎コンファレンスセンター.

目 次

内容梗概 **i**

Abstract **ii**

目 次 **v**

1 まえがき **1**

2 DNA とゲノム **7**

2.1 DNA と塩基配列 7

2.2 ゲノム 9

3 塩基配列の決定方法 **13**

3.1 シーケンシング 13

3.2 *de novo* アセンブリ 19

3.3 リードのフォーマット 22

4 de Bruijn グラフを用いた *de novo* アセンブリアルゴリズム **25**

4.1 de Bruijn グラフを用いたアセンブリの概要 25

4.2 k-mer 整数 26

4.3 ハッシング 28

4.4 de Bruijn グラフの構築 30

4.5 エッジの除去とコンティグの生成 34

4.6 de Bruijn グラフを用いた *de novo* アセンブラ 36

5 提案手法 **39**

5.1 k-mer 整数の登録とグラフの構築 41

5.2 分岐と閉路の除去 45

5.3 部分グラフの連結とコンティグの生成 48

6 実験 **55**

6.1 *E.coli* の *de novo* アセンブリ 56

6.2 ヒトの 14 番染色体の *de novo* アセンブリ 58

6.3 考察 61

7 むすび **63**

7.1 本研究のまとめ 63

7.2 今後の課題・展望 64

謝 辞 **67**

参考文献 **69**

第1章　まえがき

近年多くの生物を対象に実施されているゲノムプロジェクト，そしてそれに伴うに解析技術の発展によって，遺伝子やタンパク質の構造といった生命の持つ ”情報 ”を大量に得ることが可能となってきた．一方，それらの大量の情報から生物学的な意味を抽出することが困難となる場面も増加してきた．このような課題に取り組むために，バイオインフォマティクス (bioinformatics) と呼ばれる分野の重要性が注目されてきている．バイオインフォマティクスとは，生物学 (biology) と情報学 (information science) が融合した新たな学問・技術であり，実験等で得られる膨大な生命が持つデータからコンピュータを用いて生物学的な発見を行う分野である．バイオインフォマティクスの研究対象分野は様々であるが，その中でも重要な分野の一つとして生物の持つゲノムの解読がある．ここでゲノム（genome）とは，”gene（遺伝子）”と集合をあらわす”-ome”を組み合わせた言葉であり，一般に遺伝情報全体のことを指す．これを解読することは遺伝子の役割を解明するための重要なステップとなる．ゲノムを解読するためにはDNAの全塩基配列を解読することが必要であるが，この塩基配列は非常に長く，その並び方を決定するために必要なデータも膨大な量となるため，現在のコンピュータの処理能力をはるかに超えてしまう非常に困難な問題となる．そのような問題を解決するためのアルゴリズムの開発がバイオインフォマティクスの主要な研究分野であり、その成果は塩基配列の決定に大きく貢献してきた．　特にヒトゲノムのような大規模なゲノムの解読は，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るため，生命に関する多くの分野において重要な要素となっている．

ゲノムを解読するために塩基配列を決定することはシーケンシングと呼

ばれる．シーケンシングの最も基本的な手法として，マクサム-ギルバート法 (化学分解法) [1]，サンガー法 (酵素法) [2] がある．マクサム-ギルバート法は，特定の塩基の位置で DNA 分子を切断する化学的な処理を行う方法であるが，この方法は充分な量の DNA と，取り扱いに注意を要するヒドラジンなどを必要とするので，現在，全塩基配列の決定にはサンガー法を用いるのが一般的になってきている．サンガー法では，一本鎖 DNA から酵素 (DNA ポリメラーゼ) を用いて相補的なポリヌクレオチド鎖を形成し，特定の塩基の位置で酵素反応を停止させることによって，短い塩基配列の断片 (リード) を得る．このリードを繋ぎ合わせることで，元の全体の塩基配列を再現する. このサンガー法は様々な改良が加えられ，2000 年代前半には全自動で比較的多くのリードを生成できる装置が登場した．シーケンシングを行う装置はシーケンサと呼ばれ，リードを生成する処理は並列化によってより高速化されていき，読み取ることのできるリードの長さは 1000bp(base pair: 塩基対) 程度まで増加し読み取り精度も向上した．それに伴い，それまで手作業で行われていたリードの重複部分からの元の全塩基配列の決定をコンピュータで行う技術が必要となってきた．この技術はアセンブリアルゴリズムと呼ばれ，そのうちとくに未知である塩基配列を決定するものは *de novo* アセンブリアルゴリズムと呼ばれる．*de novo* アセンブリアルゴリズムはバイオインフォマティクスの分野の中でも重要なテーマの一つとなっている．*de novo* アセンブリでは，シーケンサで生成されるリードの長さが長ければ長い程，数が多ければ多いほど再現率は高くなるが，その分要求されるコンピュータ資源は大きくなる．また，シーケンサの読み取り誤差への対応や，反復配列と呼ばれる同じ配列が反復して (特に数回以上) 出現する領域への対応が必要である．特により短いリードでアセンブリを行おうとした場合，リードの比較・探索が高速にできる反面，アセンブリの位置を決める段階で反復配列が含まれていると，正しい並び方を得ることが困難になる．シーケンサが出力するリードの長さ・正確性を重視した改良が進むにつれてそのコスト・解析速度が大幅に増大し，大規模な DNA 鎖の塩基配列

決定のためにはより低コストにシーケンシングを行う必要が出てきた．そのため近年では，読み取るリードの長さを犠牲にし，より低コストで短時間で大量のリードを得る方法が主流となってきた．生成されるリードが短いほど，リードを読み取る処理の並列数を大幅に増やすことができ，且つ読み取りに必要な試薬を削減できる．すなわち，短いリードの生成であれば，シーケンシングのスループットを飛躍的に高めることできる．この短いリードを短時間に大量に生成することを目的とした装置は次世代シーケンサと呼ばれ，2005 年以降，様々な次世代シーケンサが登場した．次世代シーケンサの出力するリードの長さは数十〜数百 bp であり，従来のシーケンサのリードの長さと比較すると遥かに短く，その読み取り精度も比較的低い．しかし，短時間で極めて多くのリードを出力することができる．そのため，次世代シーケンサによって得られる大量の，しかし精度の低いデジタルデータを用いた塩基配列の決定を行うためのアセンブリアルゴリズムが必要となってきた．

次世代シーケンサに対応した *de novo* アセンブリアルゴリズムとして，Edena [3]，SSAKE [4]，VCAKE [5]，Velvet [6]，ABySS [7]，SOAPdenovo [8] などが挙げられる．これらのアセンブリアルゴリズムはそのアルゴリズムの特徴によって大きく 2 種類に分類される．一つは Overlap-Layout-Consensus(OLC) 法を用いたものであり，Edena，SSAKE，VCAKE がこれに分類される．OLC 法は比較的早期に提案された手法で，各リードをノードとして表現し，重複する塩基配列があるノード同士を辺で結んだグラフを作成し，全てのノードを通るパスを探索する．そしてパスを辿りながらノードの文字列を連結していくことで長い塩基配列 (コンティグ) を求める．ここで，コンティグとはリードに対してアセンブリを行うことで得られるある程度長い塩基配列であり，最も理想的な場合では 1 本のコンティグが元の DNA の塩基配列となる．この手法はノード間の重複する塩基配列の長さがある程度以上必要となるため，比較的長いリードのアセンブリに用いられる．もう一つのグループとして de Bruijn グラフ [9] を用いたものがあり，Velvet,

ABySS，SOAPdenovo がこれに分類されている．de Bruijn グラフとは，任意のノードが k 文字の文字列に対応し，辺で連結された両ノードの各文字列は $k-1$ 文字だけ重複するという特徴を持つグラフである．これらの手法では，まず全てのリードから k 文字の塩基配列を抽出してノードとし，次に $k-1$ 文字の重複があるものを辺で結ぶことで de Bruijn グラフを構築する．そして，得られた de Bruijn グラフ内のパスからコンティグを求めている．de Bruijn グラフを用いた手法は短いリードのアセンブリに向いており，次世代シーケンサの出力するリードにより適している．

de Bruijn グラフを用いたアルゴリズムのうち，Velvet や SOAPdenovo が比較的少ない消費メモリ量で高精度のコンティグが得られるので，現在では広く普及している．しかし，数 Gbp を越える大規模な全塩基配列を決定する場合，これらのアルゴリズムを用いても，実行時に要求される消費メモリ量が非常に膨大でメモリ不足となりやすい [10]．空きメモリ量が不足した場合，プログラムが強制終了されてしまう可能性もあるが，ほとんど場合はスワップと呼ばれる OS の機能が利用される．スワップとは，スワップ領域と呼ばれる専用の保存領域をメインメモリの代替領域として使用する機能である．一般的なコンピュータの OS は，このスワップを行うことでプログラムの続行を試みる．しかし，スワップ領域はハードディスク等の補助記憶の一部であり，メインメモリと比較してアクセス速度は極めて遅い．その結果，コンピュータの動作速度が著しく低下し，特に大規模な *de novo* アセンブリでは実行時間が大幅に増加してしまう．そのため，*de novo* アセンブリではその消費メモリ量が重要な課題となっている．

本研究では，de Bruijn グラフを用いた *de novo* アセンブリアルゴリズムの特徴と課題を明らかにし，次世代シーケンサから得られた大量のリードを用いて，大規模なゲノムに対しても *de novo* アセンブリが可能となるように，消費メモリ量の少ない *de novo* アセンブリアルゴリズムを提案する．本論文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，どのノードに辺があ

るかという情報は保持せずに，あるノードに対してどのノードとの間に辺が存在するかを必要な時に 4.2 節で述べる k-mer 整数を用いて効率的に調べる．これにより，大量の辺情報を保持する必要が無くなるため，大幅なメモリ消費量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表現することで更なるメモリ削減を行っている．実験では，*E. coli* K-12 strain MG1655 及びヒトの 14 番染色体から得られたリードに対して，本手法，Velvet 及び SOAPdenovo でアセンブリを行い，消費メモリ量を比較した．その結果，本手法は E. coli に対しては他手法の約 20%，ヒト 14 番染色体に対しては他手法の約 60%の消費メモリ量で *de novo* アセンブリが可能であることを確認した．

以下，第 2 章では DNA と塩基配列，ゲノムの役割とそれらの関係について，第 3 章では一般的な塩基配列の決定方法について述べる．第 4 章では de Bruijn グラフを用いた *de novo* アセンブリアルゴリズムについて説明する．第 5 章で提案手法のアルゴリズムについて説明し，第 6 章では実験とそれに対する考察を述べる．最後に第 7 章で本研究についてまとめる．

第2章　DNAとゲノム

ゲノムとは膨大な量の遺伝情報であり，それらの情報には遺伝子や遺伝子の発現の制御に関わるものなど，その生物に必要な全ての情報が全て含まれている．遺伝子の発現とは，遺伝子の持つ情報がタンパク質の合成を通じて細胞の構造や機能へと変換される過程のことを指す．ゲノムを解読するためには，DNA と呼ばれる物質を構成する分子の並び方 (全塩基配列) の決定が必要となる．本章では，まず DNA の構造と塩基配列について述べ，ゲノムの役割と塩基配列との関係について論ずる．

2.1　DNAと塩基配列

DNA(Deoxyribo Nucleic Acid) とは，デオキシリボ核酸と呼ばれる核酸の一種であり，高分子生体物質である．真正細菌・古細菌においては細胞質に環状 DNA として存在する．人間のような真核生物においては通常は細胞核内に存在し，細胞分裂の際にヒストンと呼ばれるタンパク質に巻きつき，更に幾重にも折りたたまれ，染色体として現れる．DNA は，ヌクレオチド (Nucleotide) と呼ばれる物質によって構成されている．ヌクレオチドが鎖のように連なったものはポリヌクレオチド (polynucleotide) と呼ばれ，DNA はこのポリヌクレオチドが二本平行に結合しており，二重らせん構造と呼ばれるらせん状の構造をとっている．ヌクレオチドには塩基と呼ばれる物質が含まれており，その種類によって図 1 に示すようなアデニン (adenine)，シトシン (aytosine)，グアニン (guanine)，チミン (thymine) の 4 種類の塩基があり，各塩基はそれぞれ A，C，G，T と略される．各ポリヌクレオチドの塩基が水素結合で結ばれることによって，DNA の二重らせん構造を形成

している．塩基は一方の鎖の塩基が決まればもう一方の鎖に対応する塩基も決まるという性質 (塩基の相補性) を持っており，A と T，C と G という組み合わせで結ばれている．図 2 に相補的に結合された二本鎖の模式図を示す．ヌクレオチドは含まれている塩基によってその種類が決まるため，DNA の (ポリヌクレオチド) の長さは bp(base pair:塩基対) という単位で表記される．生物の種によってその長さは大きく異なり，短いものでは数 kbp から数 Mbp，特に真核生物などの DNA は数 Gbp という膨大な長さである．例えば，ヒトの持つ DNA の長さは約 3Gbp である．また細胞分裂時には，DNA はヒストンと呼ばれるタンパク質に巻き複雑に折りたたまれ，複数に分割される．この構造は染色体と呼ばれる．ヒトの場合は，二本鎖の DNA が 23 組の染色体へと分割される．DNA のヌクレオチドの結合順を塩基の種類に注目して記述したものを塩基配列と呼び，DNA の塩基配列は”AACCCGT...” のように，一文字の略称で記述される．詳細は後述となるが，生物の持つ遺伝情報はこの塩基配列の形で保持されており，上記のような塩基配列の並び方を求めることが遺伝情報の解析において非常に基本的かつ重要な作業となる．

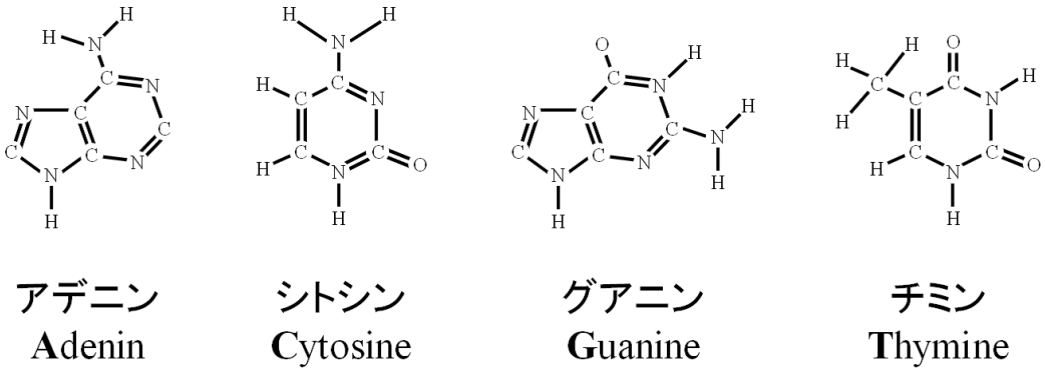


図 1. 4 種類の塩基

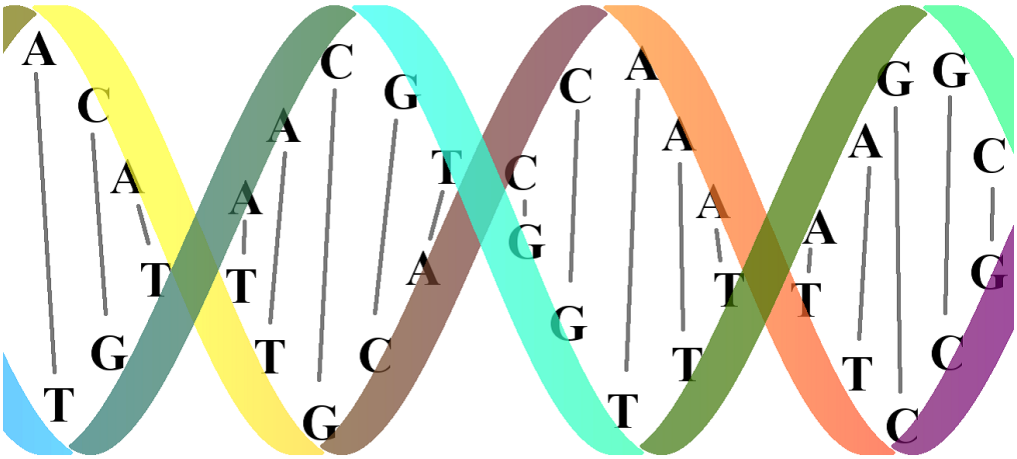


図 2. DNA の相補的二本鎖構造

2.2 ゲノム

2.1 節で述べたように，塩基配列の並び方には，生物の持つ全ての遺伝情報が含まれている．遺伝情報とは，遺伝現象によって親から子に伝わる情報であり，生物を構築する細胞の形質発現や複製に関わる情報のことを指す．遺伝情報の最小単位は遺伝子と呼ばれ，ヒトは約 2 万 3 千個の遺伝子を持つと言われている．そして，生物の持つ全ての遺伝情報 (遺伝子) のことをまとめてゲノム (genome) と呼ぶ．つまり，ゲノムを解読することによって，様々な遺伝子が生物の肉体の構築にどのように関与しているのかを明らかにすることができる．遺伝情報は様々な種類があり，代表的なものとして生物の肉体を構成する各種のタンパク質の構造や，その生成の制御に関与するものがある．タンパク質は約 20 種類あるアミノ酸が鎖上に結合したものであり，生物の神経細胞や骨、臓器といったものはすべてタンパク質から構成されている．塩基配列には，発現するタンパク質の種類を決定するアミノ酸配列や，発現するタイミング等の全ての情報が含まれている．例えば，塩基配列のコーディング領域と呼ばれる領域では，トリプレット (triplet) と呼ばれる 3 つの塩基が 1 種類のアミノ酸を意味しており，この並

び方によってアミノ酸の結合順序が決まる．アミノ酸に対応するトリプレットはコドン (codon) と呼ばれ，表 1 に示すように 20 種のアミノ酸とコドンが対応している．また，表 2 は表 1 のアミノ酸名と表記方法と対応を示している．このコドンの並び方 (アミノ酸配列) から生物に必要なタンパク質が合成される．2.1 節では DNA の塩基は A，C，G，T で表現されると述べたが，実際は DNA から RNA(RiboNucleic Asid) と呼ばれる生体物質 (DNA と同様の核酸の一種) へ塩基配列がコピー (転写) が行われ、RNA からタンパク質が合成される．RNA では塩基 T の代わりに U(uracil) が用いられるため，表 1 では T の代わりに U が対応している．RNA は DNA と同様核酸の一種ではあるが，DNA は遺伝情報の全てが保存されたものであるのに対し，RNA はその情報の一時的な処理・運搬・翻訳等を担う物質である．また，RNA からタンパク質を作ることは翻訳と呼ばれ，DNA から RNA への転写，RNA からタンパク質への翻訳という一連の流れはセントラルドグマ (central dogma) と呼ばれる．この他にも塩基配列には様々な遺伝情報がコードされており，タンパク質の合成に関与するものの他にも様々な情報が含まれている．例えば，いつ，どのような条件になったら転写を開始するかといった，転写の活性を調節する特殊な塩基配列などがある．このように，塩基配列は様々な情報を持っており，その並び方を調べることでゲノムを解析することができ，その生物が持つ様々な性質を知ることができる．また，塩基配列の並び方は一生変わることがないため，それによって決定される各個体の性質は一生変わることがはない．塩基配列の全てがタンパク質合成に関わるものではなく，そもそも未だ解明されていない領域も多いため，塩基配列の並び方を決定しただけではゲノムを完全解読することはできない．しかし，全塩基配列の決定は，生物の持つ遺伝情報全体，生物一個体を形成するのに必要な情報を明らかにするの非常に重要なステップである．

表 1. コドンとアミノ酸の対応

	2 塩基目				
1 塩基目	T	C	A	G	3 塩基目
U	UUU:Phe	UCU:Ser	UAU:Tyr	UGU:Cys	U
	UUC:Phe	UCC:Ser	UAC:Tyr	UGC:Cys	C
	UUA:Leu	UCA:Ser	UAA:終止	UGA:終止	A
	UUG:Leu	UCG:Ser	UAG:終止	UGG:Trp	G
C	CUU:Leu	CCU:Pro	CAU:His	CGU:Arg	U
	CUC:Leu	CCC:Pro	CAC:His	CGC:Arg	C
	CUA:Leu	CCA:Pro	CAA:Gln	CGA:Arg	A
	CUG:Leu	CCG:Pro	CAG:Gln	CGG:Arg	G
A	GUU:Ile	GCU:Thr	GAU:Asn	GGU:Ser	U
	GUC:Ile	GCC:Thr	GAC:Asn	GGC:Ser	C
	GUA:Ile	GCA:Thr	GAA:Lys	GGA:Arg	A
	GUG:Met	GCG:Thr	GAG:Lys	GGG:Arg	G
G	AUU:Val	ACU:Ala	AAU:Asp	AGU:Gly	U
	AUC:Val	ACC:Ala	AAC:Asp	AGC:Gly	C
	AUA:Val	ACA:Ala	AAA:Glu	AGA:Gly	A
	AUG:Val	ACG:Ala	AAG:Glu	AGG:Gly	G

表 2. アミノ酸の表記方法

アミノ酸名	3 文字表記
アラニン (Alanine)	Ala
アルギニン (Arginine)	Arg
アスパラギン (Asparagine)	Asn
アスパラギン酸 (Asparastic acid)	Asp
システイン (Cysteine)	Cys
グルタミン (Glutamine)	Gln
グルタミン酸 (Glutamic acid)	Glu
グリシン (Glycine)	Gly
ヒスチジン (Histidine)	His
イソロイシン (Isoleucine)	Ile
ロイシン (Leucine)	Leu
リジン (Lysine)	Lys
メチオニン (Methionine)	Met
フェニルアラニン (Phenylalanine)	Phe
プロリン (Proline)	Pro
セリン (Serine)	Ser
スレオニン (Threonine)	Thr
トリプトファン (Tryptophan)	Trp
チロシン (Tyrosine)	Tyr
バリン (Valine)	Val

第3章　塩基配列の決定方法

ゲノムを解読するためには，まず全塩基配列を決定することが必要となる．一般的に，塩基配列を決定するには幾つかの手順が必要となる．DNAの一般的な塩基配列決定の流れを図3に示す．まず未知のDNAから次世代シーケンサを用いてシーケンシングを行うことで，断片的な塩基配列の情報(リード)を得る．次世代シーケンサでは非常に大量のリードを出力することができるが，その長さはDNAに対して非常に短いため，そのまま遺伝子解析等に利用することはできない．そこで，リード間の重複関係を利用することでリード同士をつなぎ合わせていき，より長い塩基配列(コンティグ)を生成する．理想的には1本のコンティグが元のDNAの塩基配列となることであるが，ほとんどの場合は複数本のコンティグとなる．複数のコンティグ間には隙間(ギャップ)があるため，必要に応じてPCR法と呼ばれる手法によってDNAの増幅を行うここでギャップを埋める．このように，リードから複数のコンティグを生成することで元のDNAの塩基配列を再構成する．本章では，一般的な塩基配列の決定の各手順について述べる．

3.1　シーケンシング

シーケンシングとは，本来は塩基配列の並び方を決定すること自体を指すが，次世代シーケンサを利用した塩基配列の決定では，リードを生成する(断片的な塩基配列の決定)のことを意味することが多い．そのため，本論文でも断片的な塩基配列の決定をシーケンシングと呼ぶ．シーケンシングの基本的な原理としてはマクサム-ギルバート法とサンガー法が一般的であるが，ここではより一般的なサンガー法について説明し，近年主流となっ

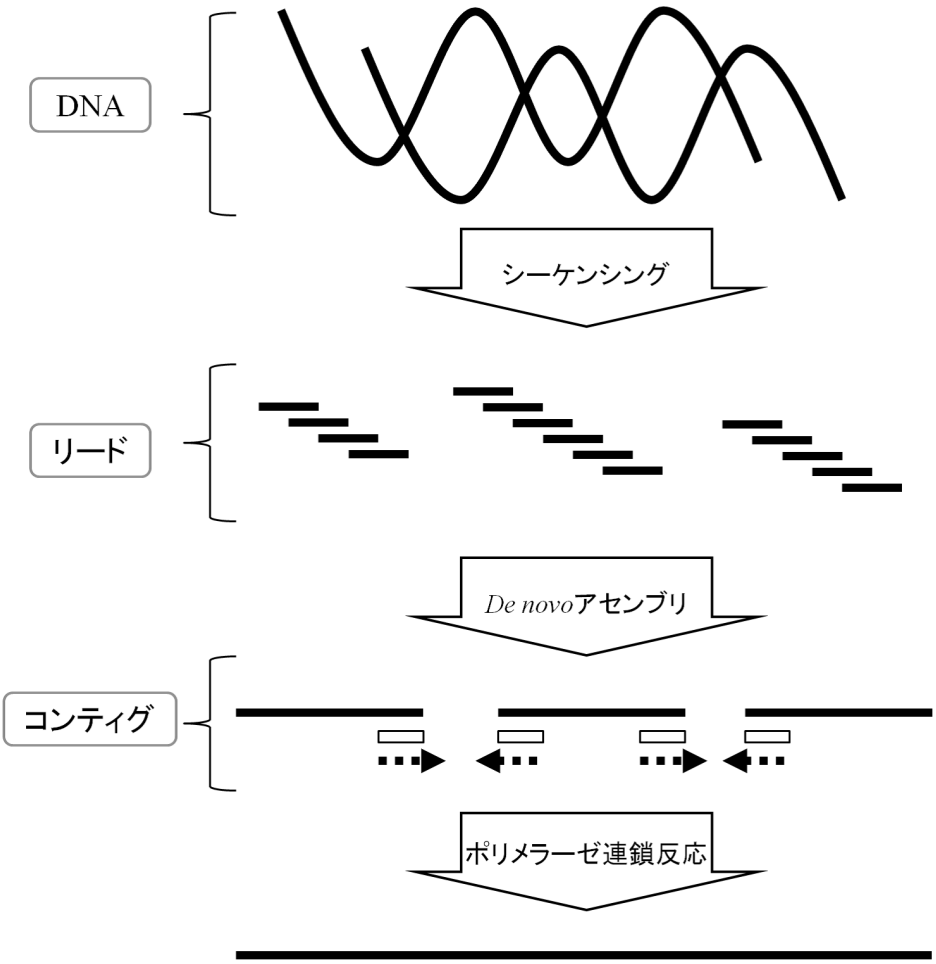


図 3. 一般的な塩基配列決定の流れ

てきている次世代シーケンサについて解説する．

まず，図 4 に示すように，解読したい DNA(鋳型 DNA) とその一部と相補的な短い DNA(プライマ DNA) を，DNA 合成することができる酵素 (DNA ポリメラーゼ) と DNA を構成するデオキシヌクレオチド (dNTP) の共存させることで酵素反応を起こさせる．この酵素反応により，鋳型 DNA に相補的なデオキシヌクレオチドがプライマ DNA に次々に結合し，二本鎖 DNA の合成が行われる．しかし，デオキシヌクレオチドの特定の原子を本来とは異なる原子で置換したデオキシヌクレオチド (ジデオキシヌクレオチドと呼ぶ) が結合されると，酵素反応が止まる．ここで，特定の原子とはヌクレオチドの連結に必要な原子である．例えばアデニンではヒドロキシル基が連結に本来必要であり，これを水素原子で置換している．各デオキシヌクレオチドとジデオキシヌクレオチドの共存下で上記の合成反応をさせると，デオキシヌクレオチドが結合されれば合成は進行し，ジデオキシヌクレオチドが結合されれば合成が停止する．両者はランダムで結合されるため，様々な長さの塩基配列の集団が得られることになる．この性質を利用し，ジデオキシヌクレオチド (ddNTP) をターミネータ (鎖停止ヌクレオチド) として一種類のみ加えることで，相補的な配列のポリヌクレオチド鎖を合成し，特定の塩基の位置でその合成反応を停止させることができる．このとき，デオキシヌクレオチド自体，またはターミネータを蛍光標識しておく．その後，電気泳動によって二本鎖を分離する．分離の際に短い塩基配列ほど速く移動するため，図 5 に示すように塩基配列の長さの短い順に流れてくる．これらにレーザー光を当ててターミネータの蛍光標識の発色を次々に読み取っていき，塩基配列の並びを得る．

第 1 章で述べたように，サンガー法は多数の研究により改良が重ねられ，生成できる断片の長さは 1000bp まで増加したが，そのコスト・必要な時間も増大し，大規模な DNA の塩基配列の決定に用いるのは困難となった．そのため，読み取るリードの長さを数十〜数百 bp と短くすることで，読み取りの並列数を大幅に増加させたシーケンサが主流となっていった．このよう

なシーケンサは次世代シーケンサと呼ばれ，従来型と比較して高速・低コストでリードを読み取ることが可能である．従って，短時間で大量のリードを低コストで生成することができるという特徴を持つ．代表的な次世代シーケンサにはイルミナ社の Genome Analyzer(GA) がある．GA は従来型のシーケンサと比較して飛躍的にスループットが向上している．例えば，GA を用いてヒトゲノムからリードを生成するとき，従来型のシーケンサと比較して必要な時間・コストはそれぞれ 1/160，1/52000 に削減でき，従来型のシーケンサでは 1 ランあたり 300kbp の塩基を読み取るのに対し，次世代シーケンサでは 27Gbp もの塩基を読み取ることができる．近年では，このような次世代シーケンサで短い大量のリードを生成することが主流となってきており，それに対応するアセンブリアルゴリズムが必要となってきている．更に，低コストでより多くのリードを生成することを重視した結果，ほとんどの次世代シーケンサでは読み取り精度が従来型と比較して低下し，生成するリードにはシーケンスエラーが多く含まれている．ここでシーケンスエラーとは，シーケンサから出力されたリードに含まれる読み取りミスである．例えば，”AACCGGTTAACCGGTT”という塩基配列をシーケンサによって処理した時，”AACCGCTT”や”CCGGNTAA”(’N’ は読み取りに失敗している) のようなリードが出力されてしまう．シーケンサ自身の対策として，読み取り精度の向上を目指した改良や，出力できるリード長を伸ばすことで上記二つの問題に対応したシーケンサもある．また，単純に同じ領域に対して読み取り・複製を何度も行うことで，シーケンスエラーをある程度減少させることも可能である．しかし，最新のシーケンサでもその読み取り精度は 100%ではなく，完全に対策されているとは言えない．以上のことから，アセンブリアルゴリズムは大量のリードを効率的に扱うことに加えて，これらの問題に対する処理をどのように行うかが重要になる．

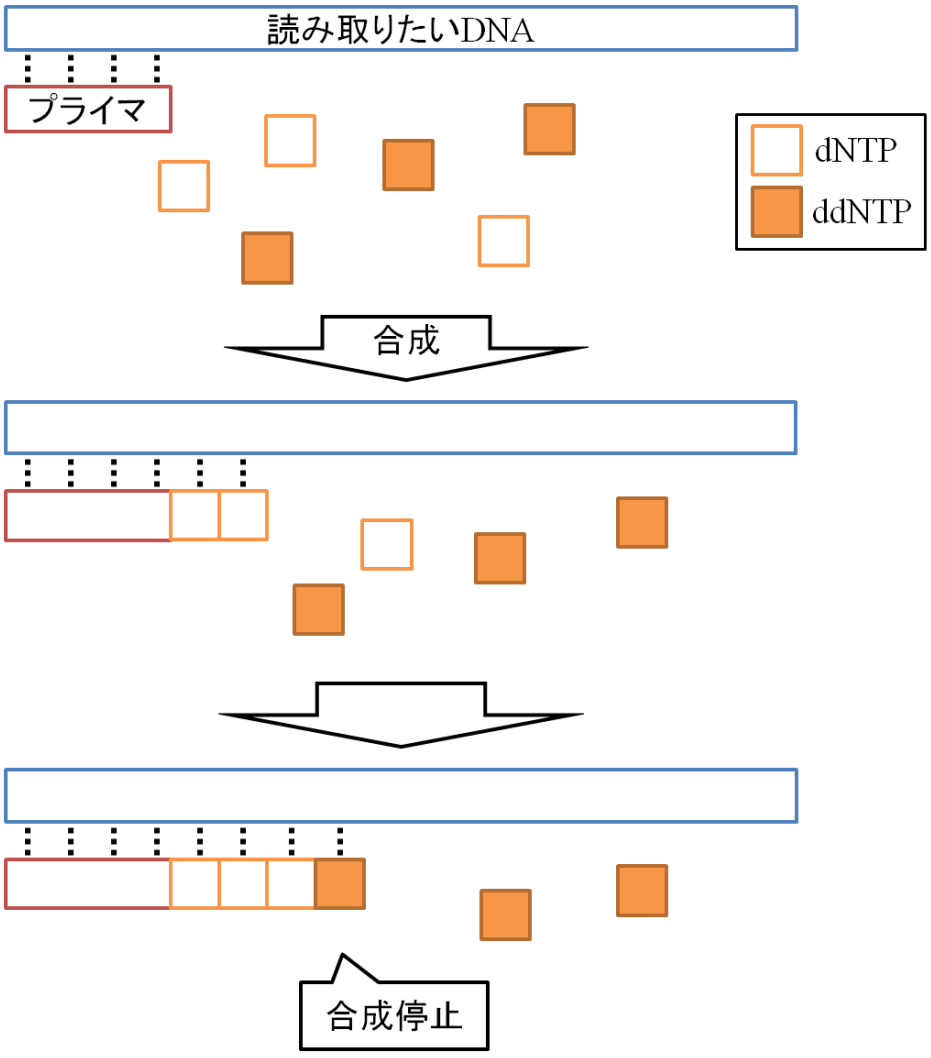


図 4. 二本鎖 DNA の合成の例

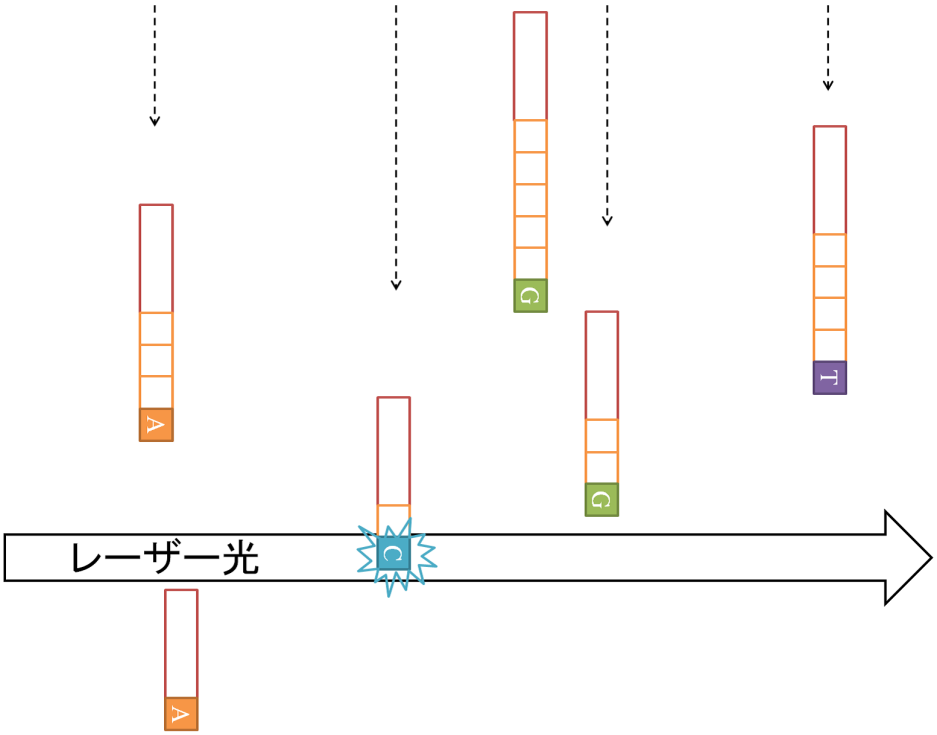


図 5. 発色による塩基配列読み取りの模式図

3.2 *de novo* アセンブリ

現在では次世代シーケンサと呼ばれる装置の登場によって大量のリードを短時間で取得できるようになった．しかし，ここで得られるリードは元の全塩基配列の” どこか” の領域であり，得られたリードが実際の元の配列のどの領域を表しているのかは知ることができない．そのため，複数のリード間で配列の重複関係を調べ，リードを連結していくことでより長い塩基配列 (コンティグ) を生成する．図 6 に示すように，このリードからコンティグを得る技術をアセンブリと呼び，とくにそれまで未知であった DNA の塩基配列の決定を行う技術を *de novo* アセンブリと呼ぶ．現在，次世代シーケンサの登場と改良に伴い様々なアルゴリズムが提案されており，それを利用したプログラム・ソフトウェアは *de novo* アセンブラと呼ばれる．一方，既に全塩基配列が決定されている DNA の塩基配列に対して，シーケンサによって得られたリードがこの既知の DNA 鎖のどの一部分になるかを決定する，マッピングアセンブリと呼ばれる手法がある．この手法は，類似する種族の生物や，同種の生物の個体差を解析するのに有効とされる．本研究では未知の全塩基配列の決定が目的であるため，本稿では前者の *de novo* アセンブリアルゴリズムについての検討を行う．

アセンブリする上で反復配列への対応やシーケンスエラーの発生という問題は避けられず，使用するリード長によっては完全に元の DNA を再現することはできない [11]．ここで，反復配列とは，塩基配列上において同じ配列が反復して出現することを言い，主に真核生物など特に進化した動植物に多く見られる．例えば，”ACGT” という塩基配列が”ACGTACGTACGT...” のように何度も連続して出現したり、”AAA...” のような一文字の繰り返しや，更に数十 bp の配列が反復して現れることもある．例えば，反復配列がリード長より長い場合，繰り返されている配列が何回出現しているのかをリードから判断することはできない．ただし，そのような場合でもある程度長いコンティグを決定することはできる．コンティグに対しアセンブリで決定できなかった領域，すなわち元の全塩基配列の内のコンティグ以外

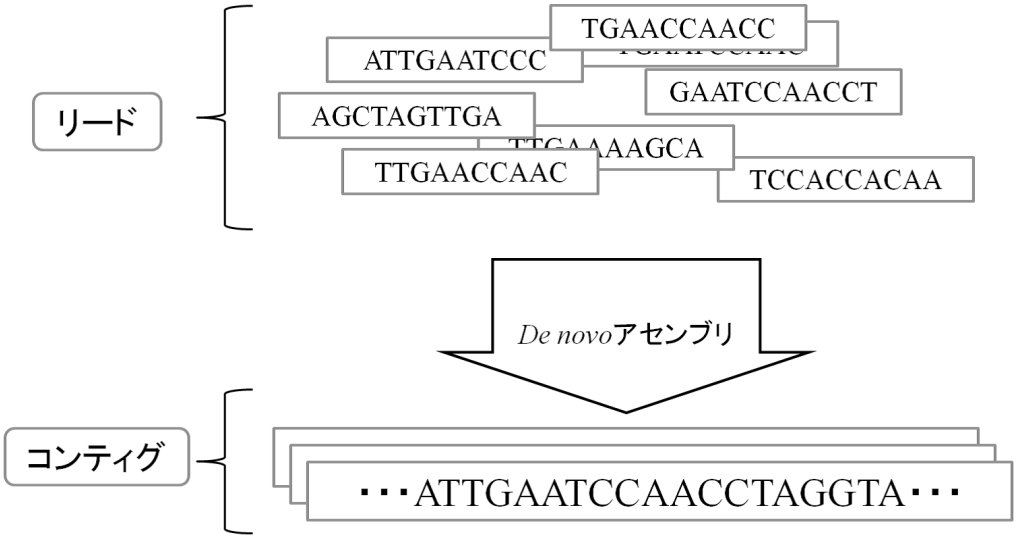


図 6. リードからアセンブリを行う流れ

の部分はギャップと呼ばれる．

ギャップ部分の配列が決定できない限り，全塩基配列の完全な再現は不可能である．しかし，PCR 法 (Polymerase Chain Reaction，ポリメラーゼ連鎖反応) [12] と呼ばれる手法によって，数十 bp の短いギャップならその部分の配列を決定することができる．PCR 法は，酵素法に似た DNA 増幅の手法であり，DNA 上で特定の領域だけを選択的に増幅させることができる．DNA を構成するポリヌクレオチドは非常に不安定な結合であるため，温度変化によって簡単に切断・再結合が可能である．PCR 法はそれを利用し，3 段階の温度変化を何度も繰り返すことで DNA の一部分だけを選択的に増幅させる．PCR 法に必要な試薬等は微量であり，増幅に要する時間も短く，サーマルサイクラーと呼ばれる全自動の卓上用装置を用いることで比較的容易に増幅することができる．そのため，*de novo* アセンブリ等によってある程度の長さのコンティグを生成することができれば，この技術を利用することで多少のギャップは埋めることができ，最終的に元の塩基配列のより多くの領域を再現できる．また，コンティグのようにギャップが残っていた

り並べ方が不確定である配列はドラフト配列 (ドラフトゲノム) とも呼ばれる．ある程度の長さのコンティグを持ったドラフト配列であれば，その情報からおおよそのゲノムを明らかにすることができる．実際にヒトのドラフト配列が決定されたことにより，ヒトの持つ遺伝子の数や構造などのヒトゲノムの輪郭が明らかにされている [13] [14]．更に，ドラフト配列とその生物の近縁種の遺伝子情報用いることで，新たな遺伝子の構造や遺伝子機能の予測にも利用することができる．ドラフト配列の情報は様々なゲノム解析に対して有効に利用することができる．以上のことから，*de novo* アセンブリの結果が複数のコンティグであっても，その情報はゲノム解析に対して様々な面から貢献できる．

3.3 リードのフォーマット

リードの種類として，シングルエンドリード (single-end read)・ペアエンドリード (paired-end read) がある．シングルエンドリードは DNA 断片の片側をシーケンシングする手法である．これに対して、ペアエンドリードは断片の両側からシーケンシングする方法である．具体的には，ペアエンドリードは一つ目の配列情報，二つ目の配列情報，これらの配列に挟まれた領域 (インサート) のサイズの三つの情報から成り立っている．ペアエンドリードのインサートのサイズは両端の配列の距離であり，塩基配列の情報だけでは知りえなかった，より大きな構造を把握するための手掛かりとなる．*de novo* アセンブリでは，コンティグ間の関係を調べるために利用することが多い．インサートの情報だけでギャップを直接埋めることはできないが，生成したコンティグ間の距離や位置関係を決定したり，繰り返し配列の解決に利用することができる．

また，リードにはシーケンシングのクオリティスコアと呼ばれる情報を別途利用する場合がある．クオリティスコアは各塩基ごと設定されており，一般的には対応する塩基のシーケンシングエラーの確率を表している．塩基配列のシーケンスデータ (ACGT からなる文字列) のみを保存する場合，FASTA フォーマットと呼ばれるテキストファイルの記述方式が用いられている．FASTA フォーマットでは，1 つのシーケンスのデータは “>” で始まる 1 行のヘッダと、2 行目以降の実際のシーケンス文字列で構成される。ヘッダでは、“>” の次にシーケンスデータを識別するための文字列を記述し、続けてそのシーケンスデータを説明する文字列を記述する．“>” で始まる別の行が出現すると、そこでシーケンスデータが区切られ、別のシーケンスデータが始まる。一方，塩基配列のシーケンスデータとクオリティスコアを併せて保存する場合は，FASTQ フォーマット [15] と呼ばれる記述形式が用いられている．FASTQ フォーマットでは，塩基配列とクオリティスコアは各 1 文字の ASCII 文字で表さている．FASTA フォーマットと FASTQ フォーマットは，次世代シーケンサー等から出力された塩基配列のデータを保存

する際のフォーマットとして標準的な記述形式となっており，公開されているリードのほとんどがこれらの形式で表されている．

第4章　de Bruijnグラフを用いた *de novo* アセンブリアルゴリズム

一般的な *de novo* アセンブラでは de Bruijn グラフを用いたものが多く，提案手法においても de Bruijn グラフを用いた代表的なアセンブラである Velvet をベースに，消費メモリ量を削減するようにアルゴリズムを改良している．本章では，Velvet や SOAPdenovo 等の *de novo* アセンブラの多くに用いられている，de Bruijn グラフを用いた *de novo* アセンブリアルゴリズムの原理と課題について述べ，提案手法の実現方針について論じる．

4.1　de Bruijnグラフを用いたアセンブリの概要

de Bruijn グラフを用いたアセンブリでは，図 6 で示したようにリードからコンティグを求めるものであるが，アセンブリの問題を de Bruijn グラフの構築とグラフの余分な枝を取り除く問題に帰着させ，それを解くアルゴリズムを実装している．de Bruijn グラフを用いたアセンブリではまず，全てのリードから k-mer(k は整数) と呼ばれる $k$bp の塩基配列を取り出し，それらをノードとする de Bruijn グラフを構築する．このように構築した de Bruijn グラフにはシーケンスエラーや反復配列等の原因によって生じた分岐や閉路が多く含まれるため，不要なエッジを削除することでそれを解決し，グラフ内のノードを辿ることでコンティグを生成する．前章で述べたように，リードから未知の元の塩基配列をすべて決定することは困難であり，とくに大規模なものに対して正しい全塩基配列を再現するのは極めて困難である．そのため Velvet や SOAPdenovo といった de Bruijn グラフを用いた *de novo* アセンブラではコンティグを複数本出力する．

4.2 k-mer 整数

de Bruijn グラフを用いたアセンブリでは，まず入力したすべてのリードに対し k-mer のパターンを抽出する．多くの *de novo* アセンブラでは，そのまま文字列として扱うのではなく k-mer 整数と呼ばれる形式に変換することで，k-mer を整数として扱う．k-mer とは k 文字の塩基配列のパターンのことを指し，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 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．例えば，”ACGTA”の 5-mer 塩基配列に対応する k-mer 整数は 108 となる．表 3，表 4 では例として 3-mer と 5-mer の整数との対応を示す．

k-mer 整数を用いてリードを扱うことはメモリ使用量の面で有利であると言える．例えば，k-mer をそのまま文字列として表した場合，一般に $k$byte のメモリを消費するのに対し，k-mer 整数では同じ $k$byte で $4k$-mer 分の塩基列を表すことができ，メモリ使用量は 1/4 に節約できる．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，配列同士の比較やある配列の探索を行う場合，k 文字の文字列よりも k-mer 整数で扱う方がより高速に処理できる．

表 3. k-mer と k-mer 整数の対応表 (3-mer の場合)

k-mer	Quaternary	Binary	k-mer integer (decimal)
AAA	0	0	0
AAC	1	1	1
AAG	2	10	2
AAT	3	11	3
ACA	10	100	4
ACC	11	101	5
ACG	12	110	6
ACT	13	111	7
AGA	20	1000	8
:	:	:	:

表 4. k-mer と k-mer 整数の対応表 (5-mer の場合)

k-mer	Quaternary	Binary	k-mer integer (decimal)
AAAAA	0	0	0
AAAAC	1	1	1
AAAAG	2	10	2
AAAAT	3	11	3
AAACA	10	100	4
AAACC	11	101	5
AAACG	12	110	6
AAACT	13	111	7
AAAGA	21	1000	8
:	:	:	:
ACGGT	1223	01101011	107
ACGTA	1230	01101100	108
ACGTC	1231	01101101	109
:	:	:	:

4.3 ハッシング

de Bruijn グラフを用いたアセンブラの多くは，k-mer のパターンそのものだけでなく，リードの中に含まれる各 k-mer が出現する回数・位置などをハッシュテーブルと呼ばれるデータ構造を用いてメモリ上に格納し，素早く呼び出せるようにしている．このようなデータ構造を用いることによって，k-mer の位置情報の参照を比較的高速で行うことができ，消費メモリ量も抑制される．k-mer の出現回数や位置情報は，後の余分なエッジの削除で重要な役割を持つ (エッジの削除については次節以降で述べる)．ハッシュテーブルの単純な実装例を図 7 に示す．図 7 の例では索引配列と位置格納配列と呼ばれるデータ構造を用いている．索引配列とは k-mer 整数の配列であり，k-mer のパターンの数だけ必要となる．図 7 の例では文字列として表現されているが実際は整数であり，実装方法によっては配列の添え字として表現される．また，位置格納配列は対応する k-mer のリード上の出現位置や出現回数を表しており，各要素が索引配列の各要素と一対一に対応している．例えば，図 7 の例では，AAA...AA という k-mer は，3 番目のリードの 11 塩基目から出現，14 番目のリードの 1 塩基目から出現... ということを意味している．このようなハッシュテーブルを用いることで，k-mer そのものと各 k-mer の出現位置や出現回数を表現することができる．しかし，索引配列の大きさ i と位置格納配列 p の分だけの消費メモリ量，

$$i = 4^k \tag{1}$$

$$p = (l - k)n \tag{2}$$

を必要とする．ここで，l はリードの長さ，n はリードの総数であり，k は索引配列の要素の長さ，すなわち k-mer における k である．この関係式から，要求されるメモリ資源はリードの長さ l，リードの総数 n に依存し，k にも依存することがわかる．例えば，k-mer の k が 1 増加すると索引配列は 4 倍必要となり，それだけ消費メモリ量が増加する．そのため多くのアセンブラでは，膨大な k-mer の全てのパターンを保存するために，上記のよう

なハッシュテーブルに様々な改良や工夫を施すことで，消費メモリ量の削減に取り組んでいる．

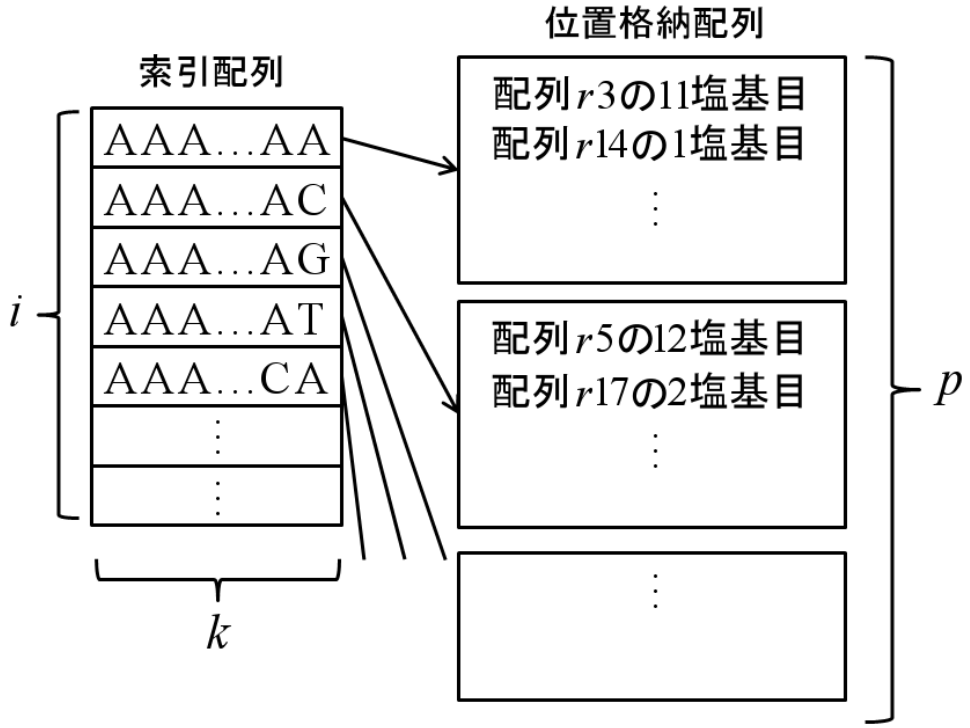


図 7. ハッシュテーブルの例

4.4 de Bruijn グラフの構築

ハッシュテーブルに格納された各 k-mer の情報を元に，図 8 に示すような de Bruijn グラフと呼ばれるグラフを構築する．de Bruijn グラフとは，任意のノードが k 文字の文字列に対応し，エッジで連結された両ノードの各文字列は $k-1$ 文字だけ重複するという特徴を持つ有向グラフである．まず，4.2 で全てのリードから得た k-mer をノードに対応する文字列とし，次に $k-1$ 文字の重複がある二つのノードを有向エッジで結ぶことで de Bruijn グラフを構築する．例えば，”AACCGCTT” というノード v_1 に対し，”ACCGCTTG” というノード v_2 が存在するならば，v_1 から v_2 へのエッジを生成する．あるノードから $k-1$ 文字の重複 (有向エッジ) を持つノードは，図 9 に示すように最大で 4 種類存在する．

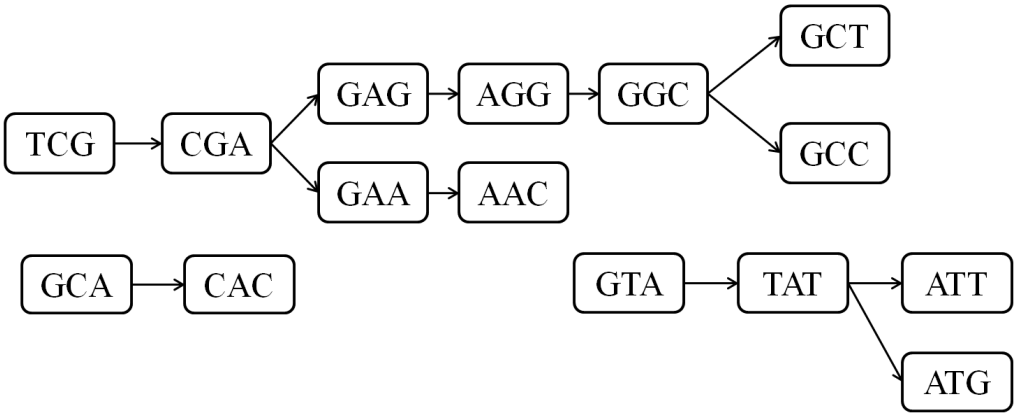


図 8. de Bruijn グラフの例 (3-mer の場合)

de Bruijn グラフを利用した *de novo* アセンブリにおいて最も理想的なグラフは，元の全塩基配列の両端の配列に対応する二つの端点 (次数が 1 のノード) と，次数が 2(入次数と出次数がそれぞれ 1) のノードのみで構成された連結グラフである．このようなグラフであれば，ある端点からもう一方の端点までノードを辿っていき，辿ったノードに対応する文字列 (k-mer) を連結していくことで，全塩基配列を生成することができる．このように，グラフ上で隣接しているノード同士を辿った際に，両端の 2 ノードの次数が 1，それ以外のすべてのノードの次数が 2 の経路 (系列) を単純パスと呼ぶ．実際には，反復配列やシーケンスエラーが含まれるため，単純パスになることはほとんどない．例えば，入次数 (或いは出次数) が 2 以上あるのノードが存在する場合，ノードを辿る際に分岐が生じてしまう．図 9 に示したように入次数 (或いは出次数) は高々4 であるが，実際に k-mer から構築された de Bruijn グラフにはこのような分岐が大量に発生する．また，パスの始点と終点が同じノードとなるパス (閉路) となる場合も非常に多い．元の塩基配列に同じ配列が何度も反復している領域 (反復配列) が存在する場合にこのような閉路になりやすく，リードの配列情報のみでは反復配列を正確に求めることは困難となる．実際に k-mer から生成された de Bruijn グラフは，これらの分岐や閉路が大量に，かつ複雑に連結された状態となっており，始点と終点までのパスが一意となる単純パスはほとんど存在しない．これらの問題となるグラフについては 5 章でより詳細に説明する．

多くの de Bruijn グラフを用いたアセンブラでは，単純パスを得るために様々な工夫を行いグラフを表現している．例えば，代表的なアセンブラである Velvet では，グラフ全体をそのまま用いるのではなく，分岐や閉路を含まない単純グラフを部分グラフとし，グラフの簡略化を行っている．検出された部分グラフは単純グラフであるため，部分グラフ内のノードを全て一度ずつ通る単純パスを一つ持つ．得られた部分グラフを一つのノードと見做すことで，グラフ全体の簡略化を行う．その後，反復配列やシーケンスエラーが原因と考えられる余分なエッジを除去し，単純パスを辿るこ

とでコンティグを生成する．Velvet における k=3 のときの de Bruijn グラフとそれを簡略化したグラフの例を図 10 に示す.

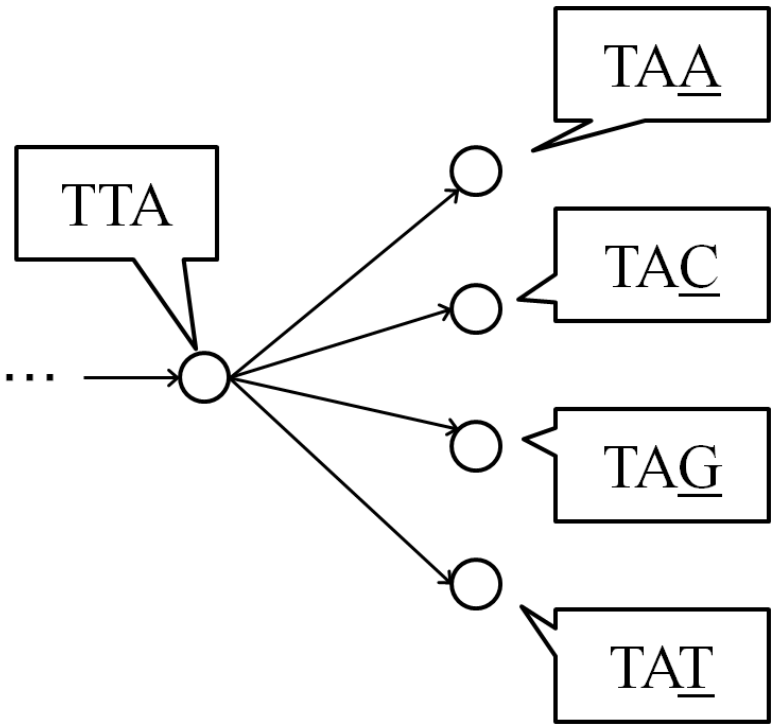


図 9. 有向エッジで接続されている 4 種のノードの例 (3-mer の場合)

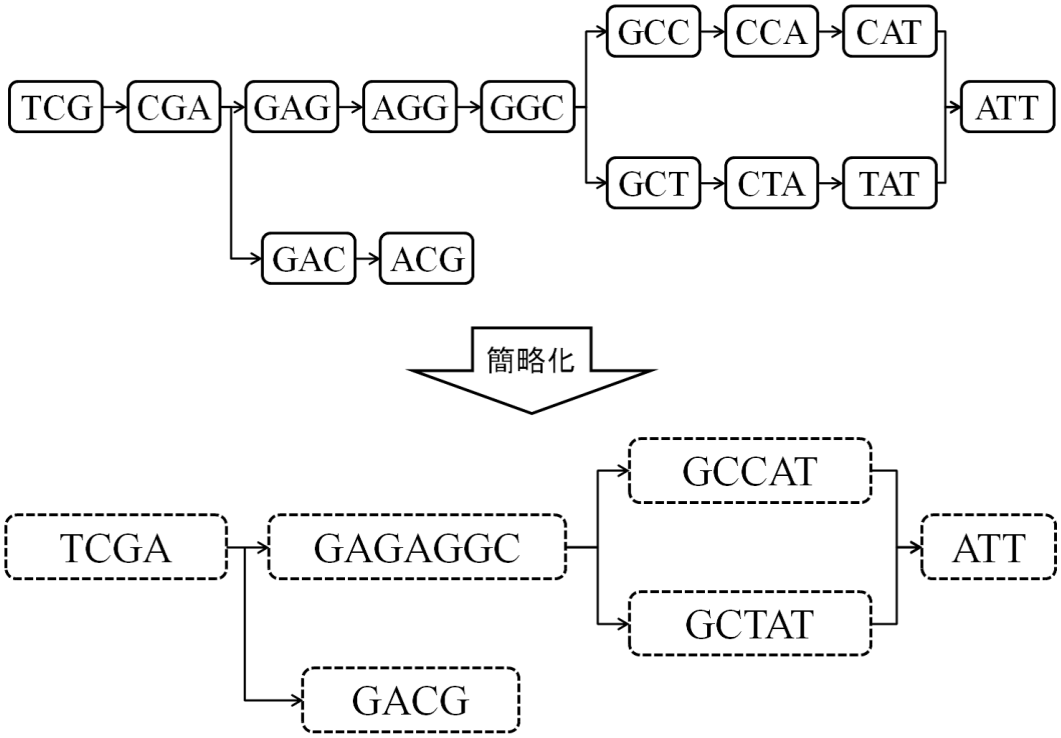


図 10. $k = 3$ の de Bruijn グラフとその簡略化の例

4.5 エッジの除去とコンティグの生成

既に述べたように，k-mer から構築した de Bruijn グラフは分岐や閉路を多く含んでおり，正しいコンティグを得ることは困難となる．そのため多くの de Bruijn グラフを用いた *de novo* アセンブラでは，誤っていると思われるエッジをグラフから除去する．例えば，Velvet では分岐が発生しているノードにおいて，分岐先の各ノードに対応する k-mer の数やその出現位置，そしてどちらのノードを辿ればより長いコンティグを得られるか等を調べる．そして，より正しいと思われるノードへのエッジを選択し，そうでない方のエッジは除去する．この処理を繰り返し行うことで，より正確でより長いコンティグの生成を行っている．ただし，例外として分岐しているパスが同じ閉路に含まれている場合はエッジ除去の対象としない．例えば，反復配列または同一の配列に挟まれた領域などが存在する場合，図 11 のようなグラフとなる．図 11 では，4.4 節で述べた簡略化を行った結果，閉路が形成されてしまった例である．このような場合，”ACCTCCGAGCT” という配列は少なくとも二回以上繰り返されている反復配列である可能性が高く，必ずしも間違ったパスではないと考えられるため，エッジの除去は行わない．多くの de Bruijn グラフを用いた *de novo* アセンブラでは，効率よく de Bruijn グラフから単純パスを求めるため，このような余分なエッジの除去 (選択) やグラフの簡略化などの様々な工夫を行っている．de Bruijn グラフを用いた *de novo* アセンブリでは，上記のような方法で de Bruijn グラフ

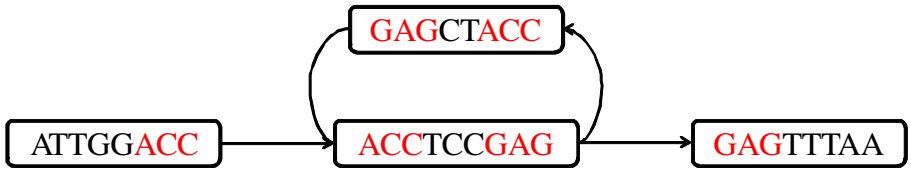


図 11. 閉路を含むグラフの例

を構築した後，図 12 に示すようにグラフの単純パスの始点から終点までのノードを辿ることで，それに対応する k-mer の塩基配列を連結しコンティグを出力する．図 12 では，破線で囲まれたノードによる単純パスから，各ノードを辿ることでコンティグを求めている．図 12 では破線で囲まれていないノードがいくつか残っている．これらのノード群から短いコンティグを生成することも可能であるが，このようなコンティグを生成するかはアセンブラの仕様や設定によって異なる． de Bruijn グラフを用いた *de novo* アセンブラでは，全てのノードを含んだ一つのグラフを構築し，元の全塩基配列を復元するようなコンティグに対応する単純パスを得ることが理想的である．しかし，分岐や閉路，或いはギャップやシーケンスエラーによって，複数のグラフ・単純パスとなる場合がほとんどである．そのため，実際の *de novo* アセンブリでは，よほど小規模ゲノムでない限り，出力は複数のコンティグとなる．

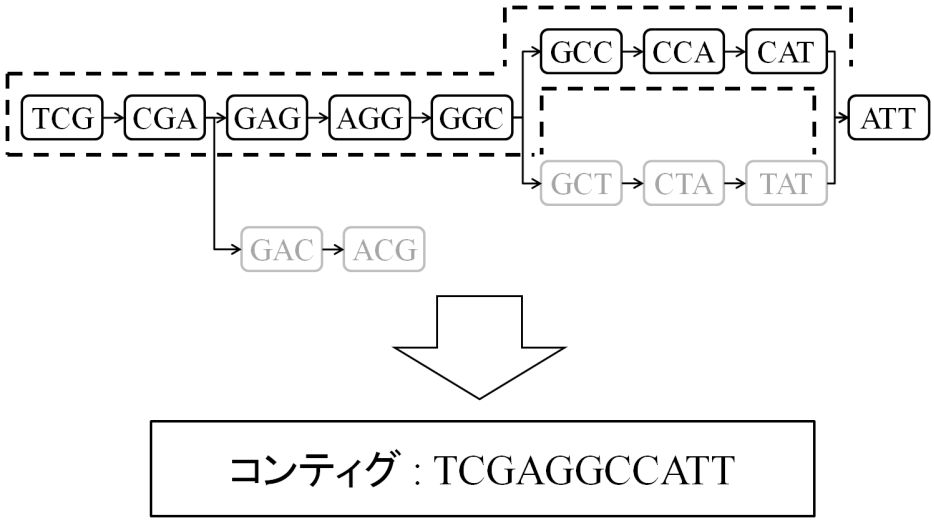


図 12. de Bruijn グラフからコンティグを生成する例

4.6 de Bruijn グラフを用いた *de novo* アセンブラ

de Bruijn グラフを用いた *de novo* アセンブラとして，Velvet や SOAPdenovo が現在では広く普及している．実際にこれらのアセンブラは数々のゲノムプロジェクトに利用されており，様々な生物種のゲノム解読に貢献している [17] [16]．しかし，数 Gbp を越える大規模な全塩基配列を決定する場合，これらのアルゴリズムを用いても，実行時に要求される消費メモリ量が非常に膨大でメモリ不足になりやすい．これに対し，de Bruijn グラフをコンパクトなデータ構造で表現することを目指した研究がある [18] [19] [20] [21]．文献 [18] [19] では，簡潔データ構造 (succinct data structure) と呼ばれるデータ構造を用いて de Bruijn グラフをコンパクトなデータ構造で表現している．しかし，これらの手法では，いかにしてグラフをコンパクトに表現するかという点に焦点が当てられており，リードあるいは k-mer を読みこむ処理や，グラフを構築する前処理などを含めた，アセンブリ全体の効率については詳細に検討されていない．簡潔データ構造を利用した de Bruijn グラフは非常にコンパクトであり，一度構築してしまえば展開等の操作を行うことなく高速にデータの参照を行うことができるが，その構築には時間的・空間的なコストが必要となる．また，それによって表現された de Bruijn グラフの変更に複雑な処理が必要であったり，変更の内容によっては新たにグラフを再構築する必要がある．

一方，文献 [20] [21] においても de Bruijn グラフをコンパクトなデータ構造を実現しており，消費メモリ量を減少させることに成功している．しかし，実際には k-mer を読みこむ処理を行う際に比較的大容量の外部記憶装置 (HDD など) を利用することで実現している．本章で述べてきたように，k-mer の読み込み処理は *de novo* アセンブリの各ステップの中でも重要な処理あるが，そのコストも *de novo* アセンブリの中で最も大きな処理の一つでもある．実際に，文献 [20] で行われた実験においても，この操作はアセンブリのステップの中で最も時間がかかっている．主記憶 (メモリ) と HDD 等の外部

記憶のアクセス速度は非常に大きな差があり，実装方法の工夫によっては全てメインメモリで処理することも可能ではあるが，文献 [20] [21] では詳細には言及されておらず，*de novo* アセンブリの結果であるコンティグの精度等に関しても評価がなされていない．このことから，これらの研究においてもいかにしてグラフをコンパクトに表現するかという点に焦点が当てられており，*de novo* アセンブリ全体の効率については詳細に検討されていない．

一方，本研究の目的は消費メモリ量の削減であるが，その方針は上記の研究の方針とは異なる．本研究では de Bruijn グラフのデータ構造のサイズのみに注目するのではなく，k-mer の読み込み処理やグラフ構築に要するコスト等も含めた，アセンブリ全体における最大の消費メモリ量を抑え，効率よく *de novo* アセンブリを行うことができる手法の提案を目的とした．また，*de novo* アセンブリの結果であるコンティグについても検討している．

第5章　提案手法

本章では，消費メモリ量を大幅に削減した *de novo* アセンブリアルゴリズムである提案手法について説明する．提案手法では 4 章で述べた，*de novo* アセンブリの問題を de Bruijn グラフを用いてコンティグを求めるアルゴリズムとして実装する．提案手法の全体の流れを図 13 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、構築したグラフを複数の部分グラフに分割する。このとき、各部分グラフは分岐による曖昧さや閉路を持たず、単純な経路 (単純パス) を必ず一つ持つように分割される。分割された各部分グラフを、より長い単純パスを持つように連結する。最後に、連結された各部分グラフ内のノードを辿り、コンティグを生成する．ただし，本手法ではグラフを構成する要素の表現や，メモリ上に保持する情報の厳選などの工夫によって，消費メモリ量の削減を行っている．本章の各節ではその詳細について述べる．

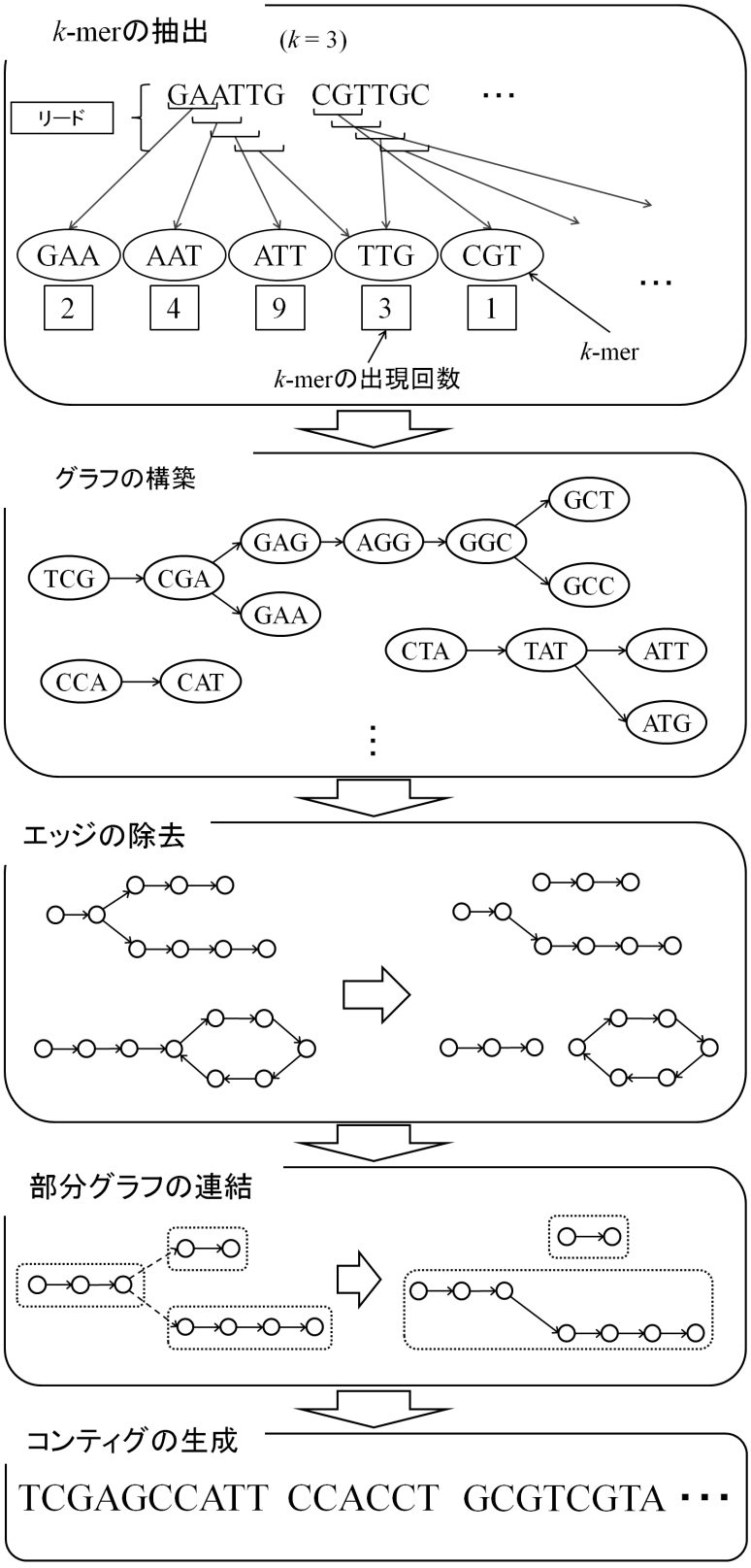


図 13. 全体の処理の流れ

5.1 k-mer 整数の登録とグラフの構築

一般的な de Bruijn グラフを用いたアセンブラと同様，本手法でも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 整数としてハッシュテーブルに登録する．図 14 に本手法の k-mer のパターンの抽出の流れを示す．本手法では，まずリードを一本ずつ読み込み k-mer を抽出する．得られた k-mer を k-mer 整数に変換し，単純なハッシュテーブルに登録していく．この処理を全てのリードに対して行う．

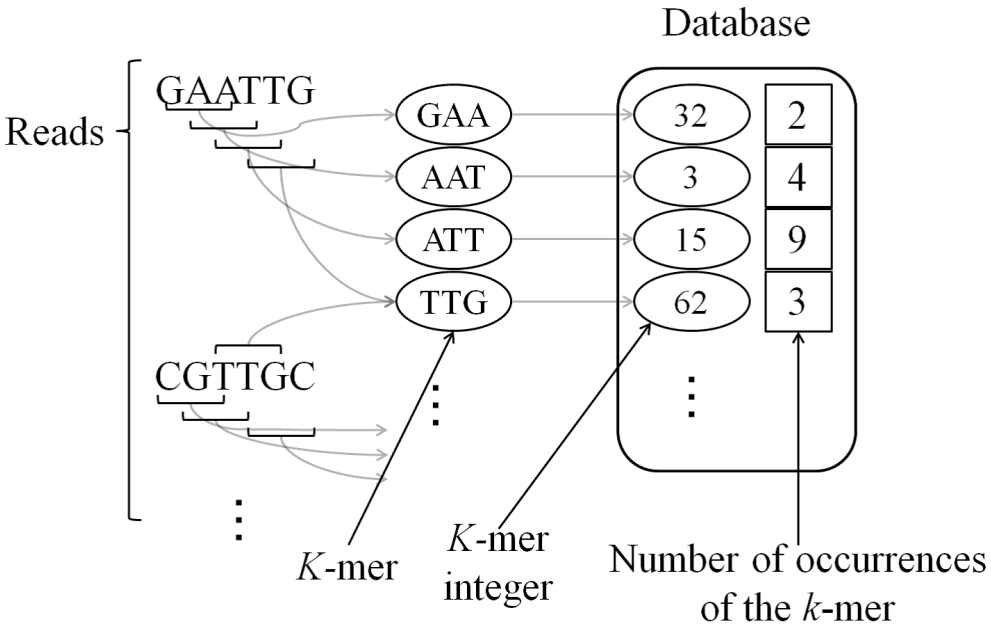


図 14. 本手法における k-mer の抽出処理

この時，第 4 章で述べたように，Velvet などのようなアセンブラの多くでは，その k-mer の出現回数やリード内における位置などの情報も併せて登録している．これらの情報を利用することで，コンティグの長さや精度を高めることができる．しかし，大規模なゲノムのアセンブリでは，必要なリードの数は大幅に増加してしまい，それに伴い k-mer のパターン数も膨

大なものとなる．それに伴い k-mer に付随する情報も大幅に増加し，それを保持するのに費やす消費メモリ量も飛躍的に増加するという問題が生じる．それに対し，本手法では k-mer 整数を登録する際にその k-mer の出現回数のみを登録することで，消費メモリ量の削減を図っている．後述となるが，de Bruijn グラフ上に発生する分岐に対してはこの出現回数のみを利用することで解決する．また本手法では，図 15 に示すような非常に単純なハッシュテーブルを用いて k-mer のパターン及び出現回数をメモリ上に格納する．本手法でのハッシュテーブルは，k-mer のパターンを基に生成された数値 (ハッシュ値) を添え字とした配列となっている．ハッシュテーブルの各要素には，k-mer 整数と出現回数が格納されているレコードへの参照情報が格納されている．ある k-mer を参照する場合は，まずハッシュ値を計算し，ハッシュテーブル上のハッシュ値に対応する要素からレコードを参照することで，目的の k-mer を参照することができる．これにより，図 7(29 項) で示したような索引配列は必要なく，k が変化してもテーブルの大きさは一定であるため，消費メモリ量を削減することができる．尚，本手法では相補鎖を考慮し，互いに相補的な k-mer は同一の k-mer として扱う．

次に，登録した k-mer を用いて de Bruijn グラフを構築する．グラフ理論におけるグラフは本来ノードとエッジの集合で表される．そのため，Velvet などの de Bruijn グラフを用いた手法では，de Bruijn グラフを構成するノード，ノード間のエッジ情報を用いることでグラフを表現する．しかし，大規模なゲノムのアセンブリではリードの数が大幅に増加し，それに伴って k-mer の数も膨大なものとなる．そのような膨大な k-mer から de Bruijn グラフを構築しようとすれば、膨大な数のノード・エッジをメモリ上に保持する必要があるため，消費メモリ量が大幅に増加するという問題が生じる．

そこで本手法では，de Bruijn グラフのエッジで連結された両ノードの各文字列は $k-1$ 文字だけ重複するという特徴を持つ有向グラフという特徴に着目した．de Bruijn グラフにおいてあるノードがエッジを持っているかを考える時，そのノードに対応する k-mer と $k-1$ 文字重複する k-mer を持つ

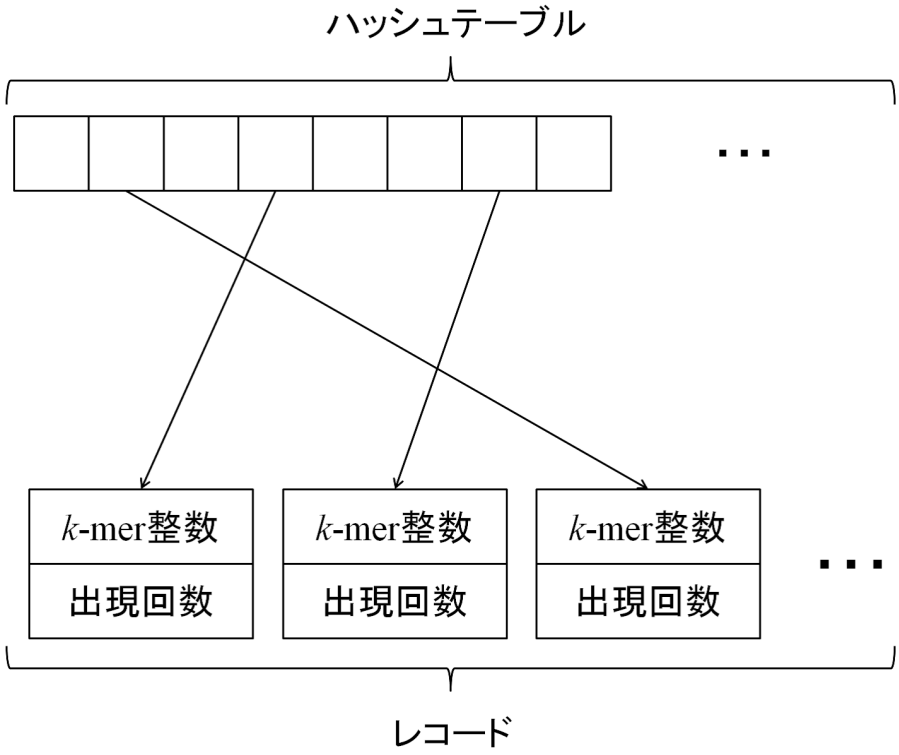


図 15. 本手法におけるハッシュテーブルの模式図

ノードが存在すれば，それらノード間にはエッジが存在するとみなすことができる．すなわち，全てのノードの存在を調べることができれば，全てのエッジの存在を調べることができる．本手法では，このような de Bruijn グラフの特徴を利用し，ノードのみで de Bruijn グラフを表現する．具体的には，メモリ上にはノードに関わる情報のみを保存し，エッジに関わる情報を一切保存しない．ノードに関わる情報とは，k-mer 整数，k-mer の出現回数，分岐・閉路の検出に必要なラベルである．ラベルの詳細については 5.2 節で述べる．あるノードがエッジを持つか否かは，パスを探索する際にノード 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 と $k-1$ 文字重複している 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．このように，de Bruijn グラフを仮想的に表現することで，大幅に消費メモリ量を削減している．ここで，k-mer”ACGTA”に対応するノードのエッジの有無を調べたいときの例を図 16 に示す．まず”ACGTA”の k-mer 整数である 108 の 2 進表記 “0001101100” を左に 2 ビットシフトすることで”CGTAA”の k-mer 整数である 660(2 進表記では “0110110000”) を得る．そしてこの値に 0∼ 3 を足した 660∼ 663 をハッシュテーブルから検索し，存在すればエッジがあると見做す．ただし，出現回数の低い k-mer はシーケンスエラーである確率が非常に高いため，そのような k-mer は無効とし，ハッシュテーブルから存在を確認しても無いものとしてみなす．実験 (第 6 章) では，出現回数が 1 回以下の k-mer は無効とした．本手法ではエッジをこのように仮想的に表現しており，有向エッジをあらかじめ調べて情報として保存する必要が無いため，大幅に消費メモリ量の削減を行うことができる．本手法では有向エッジの登録は行われないため，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．

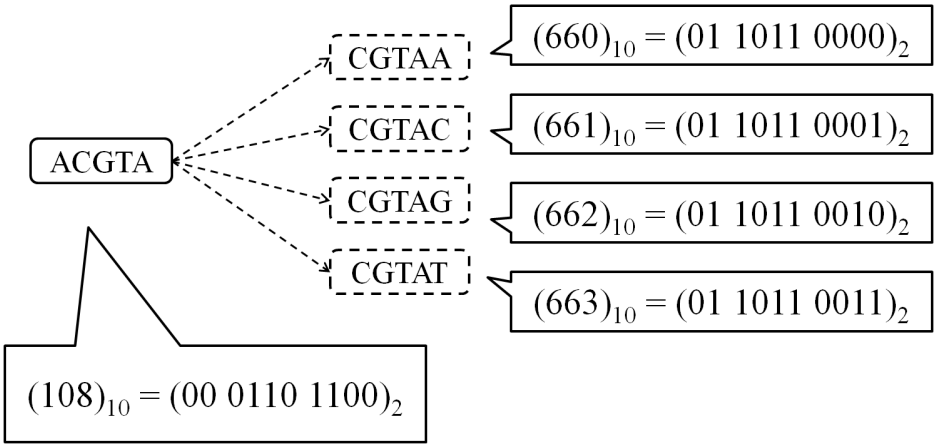
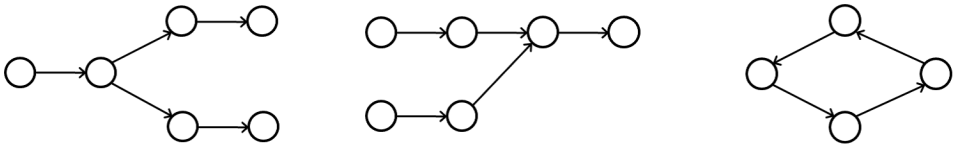


図 16. $k = 5$ の時のエッジの有無の計算の例

5.2 分岐と閉路の除去

4 章でも述べたように，k-mer から求めた de Bruijn グラフは大量の分岐や閉路が複雑に絡み合ったものとなるため，単純パスを求める上で重要な問題となる．問題となる具体的なグラフの例を図 17 に示す．図 17(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．この場合，複数あるエッジのどちらを選択するかという問題が生じる．図 17(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．この場合についても，複数あるエッジのどちらを選択するかという問題となる．図 17(c) は，パスに閉路を含む場合である．このようなグラフからそのままパスを得ようとすると，同じノードを無限に辿ってしまうこととなり問題となる．実際のグラフでは以上のような分岐や閉路のパターンが複雑に組み合わさっており，分岐や閉路の原因となるエッジを削除する必要がある．

そこで本手法では，コンティグを生成する前に以下に示すノードの走査を実施し，エッジを除去 (選択) する．最終的には，グラフ全体を閉路或いは単純パスを一つだけ持つ複数の部分グラフに分割する．



(a) (b) (c)

図 17. グラフに現れる分岐の例

1. 出現回数が最も多い k-mer に対応するノードを一つ選び開始ノードとする．

2. 開始ノードを注目ノードとする

3. 注目ノードから有向エッジで連結されているノードの存在を調べ，そのノードの数によって以下の処理を実施する．連結されているノードは最大で 4 つあり，4 種類の k-mer に対応している．具体的には始めの $k-1$ 文字は同一の塩基配列，末尾の塩基のみが異なる (ACGT の 4 種類)k-mer である．

   (a) 連結されているノードが一つしかなければ，そのノードを注目ノードとし，3 へ

   (b) ノードが複数連結されているなら，その中からノードを一つ選び注目ノードとし，3 へ (具体的な選択方法は後述する)

   (c) 連結されているノードが無ければ注目ノードノードを終止ノードとし，4 へ

4. 選択されていないノードを調べる

   (a) 注目ノードとして選択されていないノードが存在するなら，その中から出現回数が最も多いノードを開始ノードとして選択し，2 へ

(b) 注目ノードとして選択されていないノードが存在しないなら終了する。

上記のエッジ選択 (除去) 処理において，ある開始ノードからある終止ノードまで辿る間に，注目ノードとして選択されたノードに同一のラベルを与える．具体的には，i 番目に開始ノードとして選択されたノードから終止ノードまでの間に辿った全てのノードに，i というラベルが与えられる．このラベルを用いて，後述する分岐や閉路の検出や解決を行う．このラベルが同一である複数のノードは一つの部分グラフであることを意味しており，結果として 5.1 節で構築したグラフは複数の部分グラフに分割される．この時，各部分グラフにはラベルとパスの長さが併せて保存される．

ノードの走査で，分岐や閉路を検出した場合の各解決方法を以下に示す．図 17 のような分岐や閉路を検出した場合は，分岐や閉路のパターンによって次のように解決する．まず，図 17(a) のように注目ノードの出次数が 2 以上である場合は，最も出現回数の多い k-mer のノードへのエッジを選択する．本手法では各ノードに対応する k-mer と出現回数が保存されているため，その出現回数を参照・比較することで実現している．ただし，出現回数が同一であった場合はノードの選択を行わず，注目ノードノードを終止ノードとし，ステップ 4 の処理を行う．　また，図 17(b) のように注目ノードの入次数が 2 以上である場合も同様に，最も出現回数が多いノードからのエッジを選択する．図 17(b) の状態は，実際には図 18 に示すような場合に検出される，図 18 は，ノードの走査のステップ 3 において，v_a(注目ノード) から有向エッジで連結されているノード v_c が，既に異なるラベルが割り当てられている場合である．これに対し，図 18 の v_a と v_b の k-mer 出現回数を比較し，v_b よりも v_a の k-mer 出現回数の方が多ければ v_c を次の注目ノードとして選択し，v_b の出現回数の方が多ければ v_a を終止ノードとしステップ 4 の処理を行う．v_a と v_b の k-mer 出現回数が同一であった場合は，v_a と v_b を共に終止ノードとし，v_c から連結されているノードに別のラベルを割り当て直し，ステップ 4 の処理を行う．次に，図 17(c) のように，注目ノー

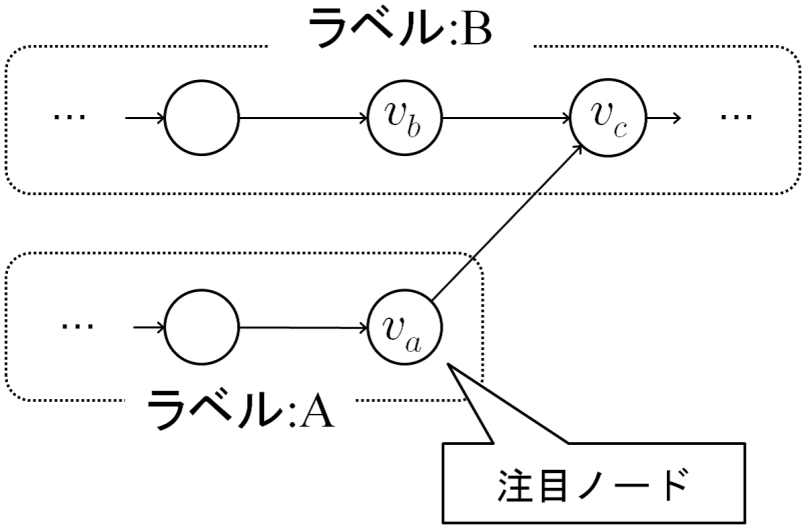


図 18. グラフに現れる分岐 (b) のラベルの状態

ドに連結されているノードが注目ノードと同一のラベルを与えられた開始ノードである場合，つまり閉路を検出した場合は，その時点の注目ノードを終止ノードとする．しかし，ほとんどの場合は，図 19 のように開始ノードではないノードに同一のラベルがあることが多い．その場合は，図 19 のように閉路となっている部分のノードに，新たな別のラベルを与え，閉路のみからなる部分グラフと単純パスを持つ部分グラフに分割する．

更に，図 20 のように注目ノードに連結されているノードが，異なるラベルの与えられた開始ノードである場合がある．これは分岐でも閉路でもない構造であるが，開始ノードとして選ばれるノードの順序によって生じる．この場合は，図 20 に示すように現在の注目ノードのラベルを与え直すことで一つの部分グラフとする．

5.3 部分グラフの連結とコンティグの生成

ノードの走査では，あるノードから複数のノードへのエッジがあり，複数のノードの k-mer の出現回数が同一であった場合は，それらのノードをそ

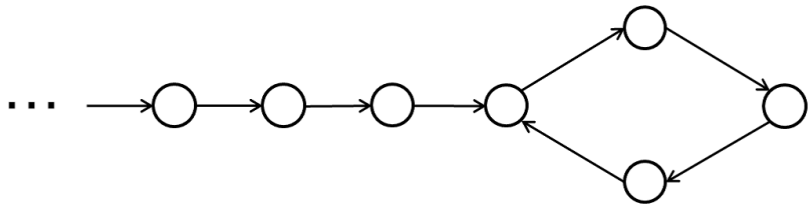


図 19. パスの一部に閉路がある例

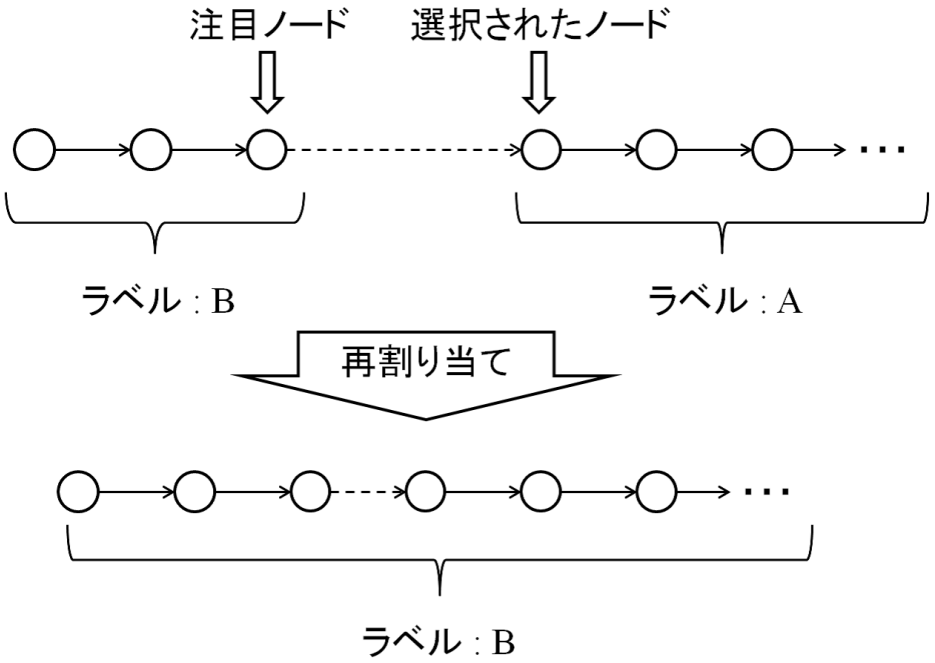


図 20. ラベルの再割り当ての例

れぞれ別々の部分グラフとなるようエッジを除去 (選択) した．また，閉路を検出した場合は閉路部分のみからなる部分グラフとした．ここでは，これらの部分グラフをより長い単純パスを構築できるように部分グラフ同士を連結する．具体的には，5.2 節で得られた複数の部分グラフを次の操作を行うことでグラフのマージを行う．

1. 部分グラフを一つ選択し開始グラフとする．この部分グラフは，まだ注目グラフとして選択されていない部分グラフの中で最も長い単純パスを持ち，なおかつその単純パスが閉路でないものが選択される．

2. 開始グラフを注目グラフとする．

3. 注目グラフの開始ノード (終止ノード) から連結されているノードで，注目グラフに含まれていないノードが存在するか調べる．あるノードから連結されているノードの存在は 5.1 で述べた方法を用いて調べる．

4. 注目グラフの開始ノードまたは終止ノードから連結されているノードが存在し，注目グラフ以外の部分グラフの終止ノード (開始ノード) であれば，そのグラフを注目グラフと連結可能なグラフとする．ただし，注目グラフの開始ノード (終止ノード) から連結されているノードが閉路を持つ部分グラフに含まれるノードであった場合は，適切なノードを開始ノードと終止ノードとし，閉路を持つ部分グラフを連結可能なグラフとする．(詳細は後述する)

5. 連結可能な部分グラフが存在するならば，連結可能な部分グラフに含まれるノード全てに注目グラフのノードと同じラベルを与え直すことで二つの部分グラフを接続する．そして連結可能とした部分グラフを注目グラフとする．

6. マージできる部分グラフが無くなるまで 3-5 の処理を繰り返す．

図 21 に部分グラフのマージの例を示す．図 21 では開始グラフから左方向へマージを繰り返している例である．同様の処理を右方向にも行い，両方向

で部分グラフのマージ・拡張を行う．ここで，もし連結可能な部分グラフを複数検出した場合，最も長いパスを持つ部分グラフを選択する．

閉路を持つ部分グラフの選択及び接続については以下のように行う．グラフのマージ処理のステップ 4 において，注目グラフの開始ノードまたは終止ノードから連結されているノードが閉路を持つ部分グラフに含まているノードであった場合，そのノードを仮想的な開始ノードとし，閉路を持つ部分グラフを単純パスを持つ部分グラフとみなすことで解決する．具体的な例を図 22 に示す．図 22 では，G_{path} は単純パスを持つ部分グラフであり，v_a をその開始ノードする．一方，v_b は閉路を持つ部分グラフ G_{cycle} に含まれるノードである．ここで，v_a と v_b に対応する k-mer が重複している，つまり v_a と v_b エッジで連結している場合，v_b と同様に G_{cycle} に含まれ，かつ v_b とエッジで連結されたノード v_c が存在する．v_b と v_c をそれぞれ G_{cycle} の開始ノードと終止ノードとみなすことで，G_{path} と G_{cycle} をマージする．その後，v_c と重複する別のノードが存在するか調べることで，部分グラフのマージを繰り返していく．同一の開始グラフから行ったグラフのマージでは同一の閉路を持つ部分グラフは一度しか選択できないが，異なる開始グラフにおけるマージであれば選択できるものとする．

全ての部分グラフのマージ処理が完了した後，図 23 に示すように，接続された全ての部分グラフの単純パスに含まれるノードの k-mer を順に参照することにより，各部分グラフに対応するコンティグを生成する．生成された各コンティグにおいて，長さが閾値以上となるコンティグをアセンブリの最終結果として出力する．実験 (第 6 章) では，この閾値はリードの長さに設定した．

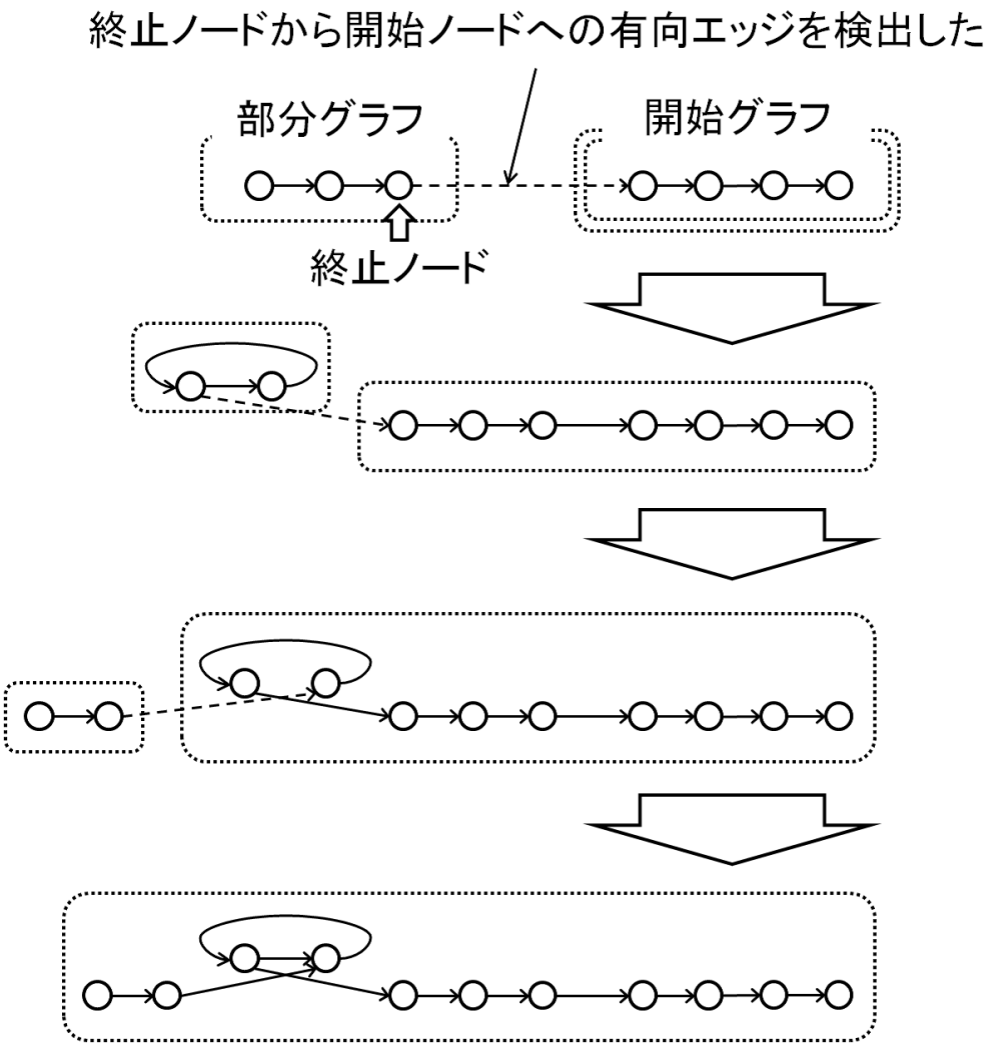


図 21. 部分グラフのマージの例

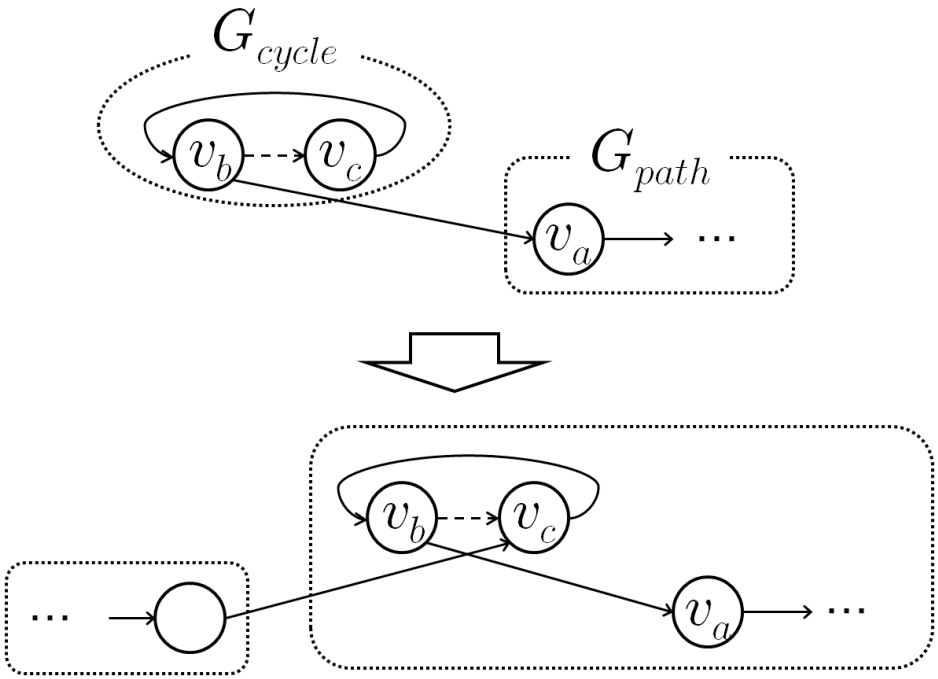


図 22. 閉路を持つ部分グラフのマージの例

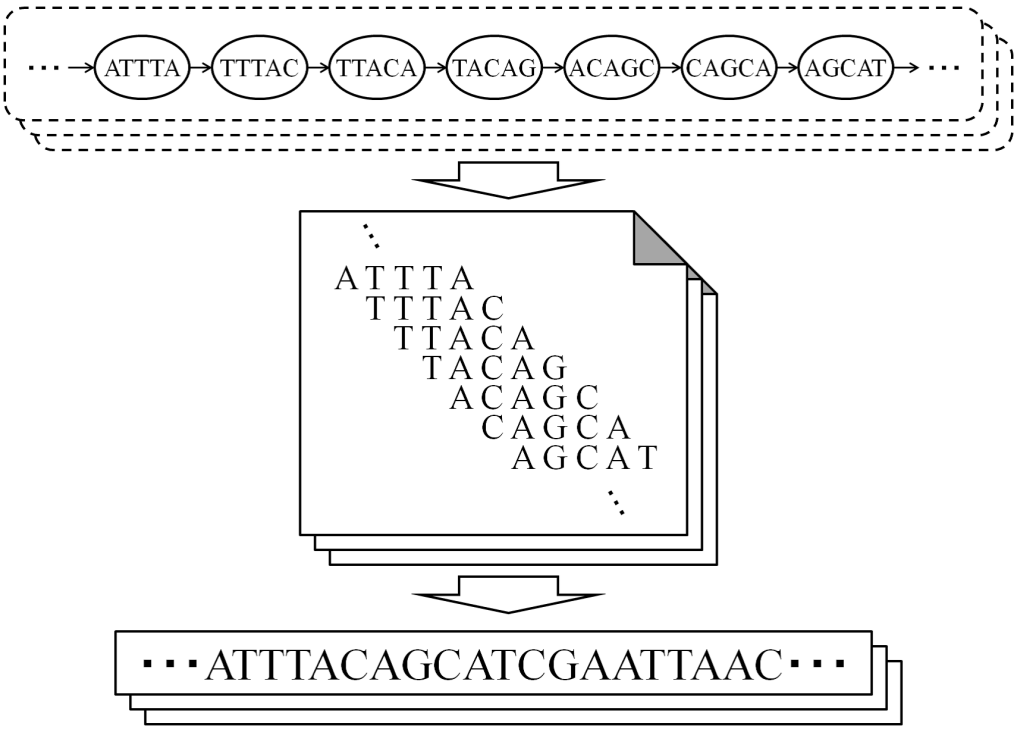


図 23. コンティグの生成

第6章　実験

提案手法の有効性を確かめるため，実際の次世代シーケンサより得られたリードを用いてアセンブリを行い，既存手法である Velvet と SOAPdenovo2 との比較を行った．Velvet は最も普及している *de novo* アセンブラの一つであり，de Bruijn グラフに基づいたアルゴリズムを採用している．また，SOAPdenovo も同様に最も普及している *de novo* アセンブラの一つである．SOAPdenovo はより大規模なゲノムのアセンブリのために設計されており，公開されている多くのゲノムのアセンブリに使用され，コンティグの生成に成功している．SOAPdenovo2 は SOAPdenovo の後継アセンブラであり，SOAPdenovo よりも消費メモリ量，コンティグの長さや精度等が向上している．実験では，これらの手法で塩基配列が既知であるゲノム，*E.coli* K-12 strain MG1655 及びヒトの 14 番染色体に対して得られたリードからそれぞれ *de novo* アセンブリを行い，最大消費メモリ量，実行時間を比較した．また，*de novo* アセンブリの結果となる各コンティグの長さや総塩基数を比較し，精度に関しては元の塩基配列を用いて比較・検討をした．なお全ての実験は 189GByte のメモリを搭載した Intel Xeon E5-2660(2.2GHz) 上で行った．

6.1 *E.coli* の *de novo* アセンブリ

最初の実験では，DNA の全塩基配列が既知である *E.coli* K-12 strain MG1655 に対して行った．全塩基配列の長さは約 4.64Mbp である．実験では，次世代シーケンサから得られた 29,879,853 本のリード (35bp) を使用した．使用したリードにはシーケンスエラーが含まれているが，ギャップと呼ばれる未確定の配列”N”は含まれていない．また，k-mer をパラメータ k=19, 21, 23, 25, 27, 29, 31 と変化させ，それに対する消費メモリ量と実行時間の変化を調べ，その結果を Velvet 及び SOAPdenovo2 と比較した．

図 24 と図 25 に，*E.coli* のアセンブリを行った際の各手法の最大消費メモリ量と実行時間を示す．図 24 より，すべての k の場合で最大消費メモリ量を削減できており，平均で SOAPdenovo2 の約 13%，Velvet の約 19%の消費メモリ量で実行できたことがわかる．このことから，*E.coli* のアセンブリでは提案手法の目的を達成できたと言える．一方，図 25 より，実行時間に関しては，Velvet が最もアセンブリに時間を要し，SOAPdenovo2 が提案手法よりもわずかに高速であった．また，全ての手法において k-mer のサイズが増加するとともに実行時間も低下していることがわかる．提案手法においては，k-mer のサイズが増加するとも消費メモリ量は低下した．一方，Velvet と SOAPdenovo2 における消費メモリ量と k-mer のサイズに因果関係は見られなかった．

次に，表 5 に *E.coli* の各手法のアセンブリの結果を示す．表 5 はアセンブリの結果得られたコンティグの長さや量，精度を示している．表中の最適 k-mer サイズとは N50 値が最大の時の k のサイズを表している．また N50 値とは，その値よりも長いコンティグが全てのコンティグの 50%を占めることを意味する数値であり，N50 値が長いほど優れた結果である．コンティグ数は得られたコンティグの本数，総塩基数は得られたコンティグの長さの総和である．カバー率は，得られたコンティグが元の全塩基配列をカバーできた割合を示している．エラー率は得られたコンティグの内，元の塩基配列に存在しなかい配列 (エラー) の割合を示している．表 5 より，提案手法

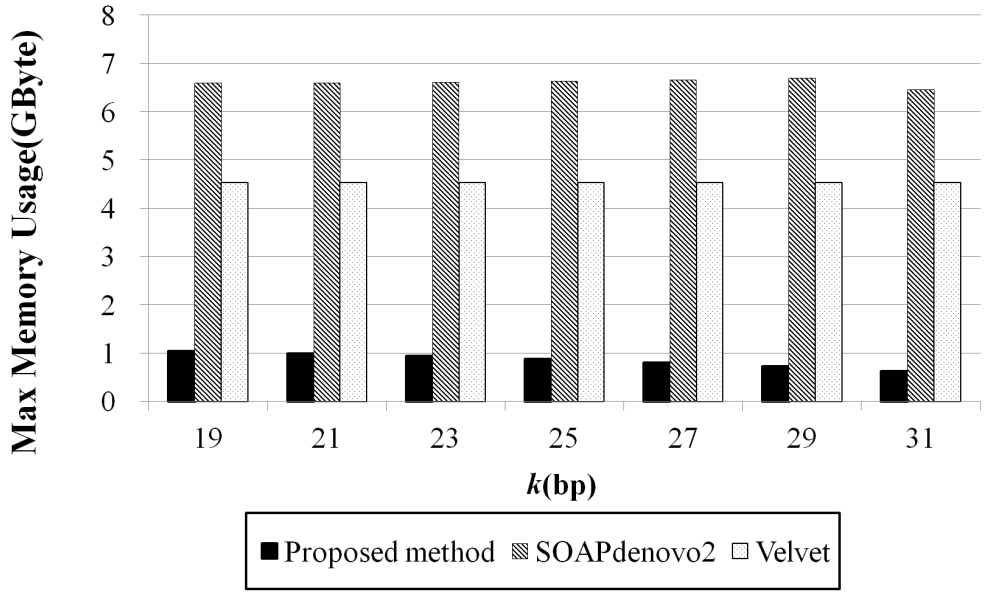


図 24. 最大消費メモリ量の比較 (*E.coli*)

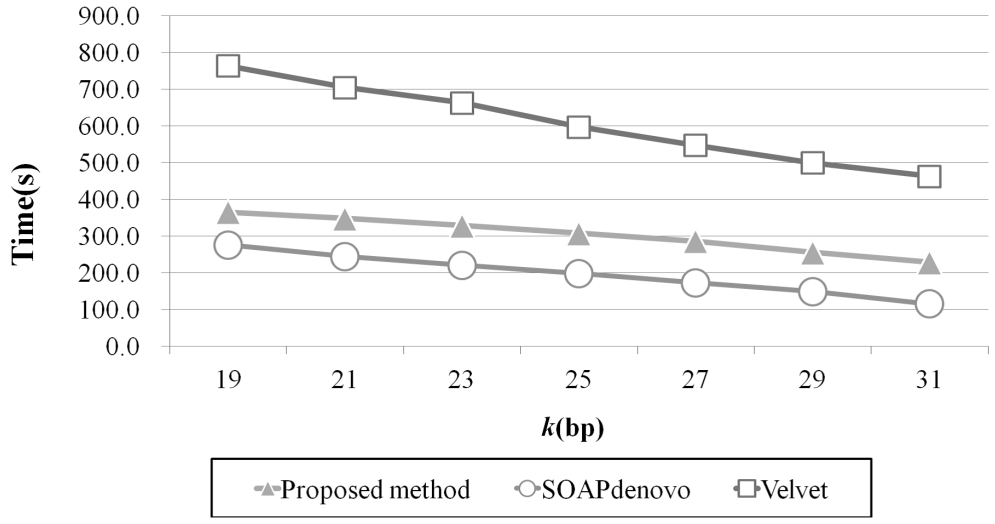


図 25. 実行時間の比較 (*E.coli*)

表 5. アセンブリ結果の比較 (*E.coli*)

Assembler	最適 k-mer (bp)	N50 (kbp)	コンティグ数	総塩基数 (kbp)	カバー率 (%)	エラー率 (%)
提案手法	25	16.4	602	4,514	96.77	0.00272
SOAPdenovo2	29	19.0	2008	4,542	98.33	0.06988
Velvet	29	22.9	754	4,544	98.03	0.00000

では N50 値は他の手法に比べわずかに短い結果となった．一方，コンティグの本数・総塩基数やカバー率・エラー率には大きな差が表れなかったため，どの手法が優れているとは言い難い結果となった．以上より，*E.coli* のアセンブリにおいて提案手法の有効性を確認できた．

6.2 ヒトの 14 番染色体の *de novo* アセンブリ

6.1 の実験では，比較的小規模なゲノムに対する *de novo* アセンブリであった．そこで次の実験では，より大規模なゲノムに対して *de novo* アセンブリを行う．DNA の全塩基配列が既知であるヒトの 14 番染色体に対して行った．全塩基配列の長さは約 100Mbp である．実験では，GAGE で用いられたデータセットのリード (101bp) を使用した．GAGE(Genome Assembly Gold-standard Evaluations) [22] とは，*de Novo* アセンブリの性能評価を行うための代表的な報告の一つである．GAGE ではアセンブリ結果 (コンティグ) の質に焦点が当てられており，消費メモリ量に関する詳細な検討が行われていない．GAGE の性能評価に使用されたデータセットは公開されているため，ここではその内のヒトの 14 番染色体のリードを使用した．本データセットに含まれるリードは本来はペアエンドリードの FASTQ 形式であるが，本実験では全てシングルエンドリードの FASTA 形式に変換しアセンブリを行った．リードの総本数は 61,579,272 本とし，長さは 101bp とした．また，使用したリードにはシーケンスエラーが含まれており，未確定の配列”N”が含まれている．本実験では k-mer をパラメータ k=51, 55, 59, 63, 67, 71, 75 と変化させ，それに対する消費メモリ量と実行時間の変化を調べ，その結果を Velvet 及び SOAPdenovo2 と比較した．

表 6. アセンブリ結果の比較 1(ヒトの 14 番染色体)

Assembler	最適 k-mer (bp)	N50 (bp)	コンティグ数	総塩基数 (kbp)
提案手法	51	1,119	217,033	101,255
SOAPdenovo2	63	3,682	186,558	91,358
Velvet	63	5,108	73,054	83,706

図 26 と図 27 に，ヒトの 14 番染色体のアセンブリを行った際の各手法の最大消費メモリ量と実行時間を示す．図 26 より，提案手法は全ての k の場合で従来手法よりも最大消費メモリ量を削減できており，平均で SOAPdenovo2 の約 54%，Velvet の約 63%の消費メモリ量で実行できたことがわかる．このことから，ヒトの 14 番染色体のアセンブリにおいても提案手法の目的を達成できたと言える．一方，図 27 より，実行時間は Velvet よりも提案手法は遅く，SOAPdenovo2 よりも若干遅い結果となった．また，Velvet や SOAPdenovo2 では，k-mer のサイズが増加するとともに最大消費メモリ量も低下していることがわかる．しかし，提案手法においては，k-mer のサイズの変化と最大消費メモリ量との因果関係は見られなかった．　次に，表 6，表 7 にヒトの 14 番染色体の各手法のアセンブリの結果を示す．表 6 はアセンブリの結果得られたコンティグの長さ・量について示し，表 7 はコンティグのその質・精度について示す．ここでギャップを除いたエラー率とは，元の DNA の塩基配列に存在する未確定の配列の領域 (ギャップ) を除いたカバー率である．本実験で使用したヒトの 14 番染色体の元の塩基配列は完全に既知ではなく，未確定の配列の領域が存在する．そのため，本実験ではギャップを除いたカバー率を評価項目として追加した．表 6 より，提案手法では N50 値は他の手法に比べて短い結果となった．コンティグ数は Velvet が最も少なく，提案手法と SOAPdenovo2 では比較的多い結果となったが，総塩基数は大きな差は表れなかった．一方，表 7 より，カバー率とエラー率・ギャップを除いたカバー率に関しては大きな差は表れなかった．以上より，ヒトの 14 番染色体対する *de novo* アセンブリにおいても，提案手法の有効性を確認できた．

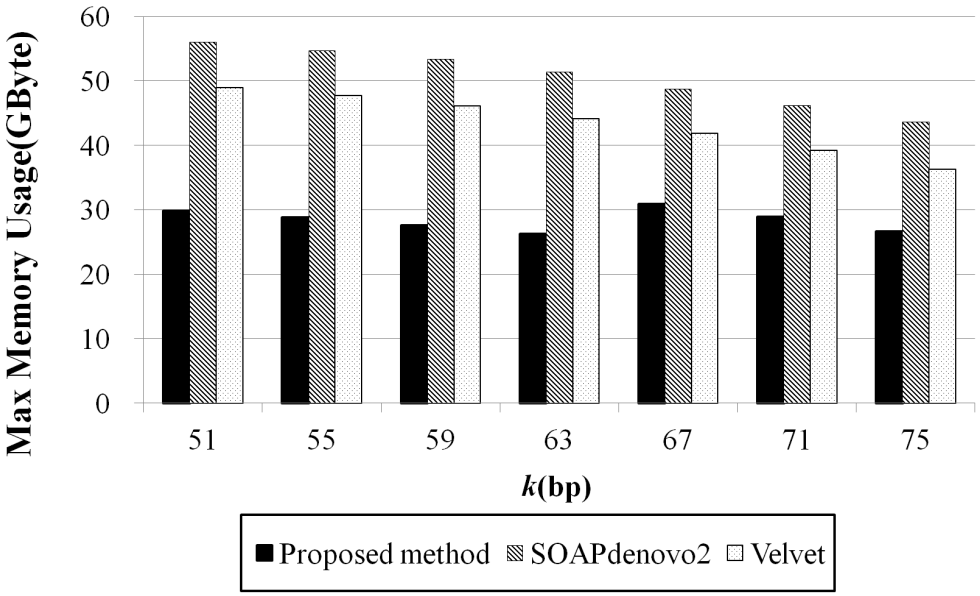


図 26. 最大消費メモリ量の比較 (ヒトの 14 番染色体)

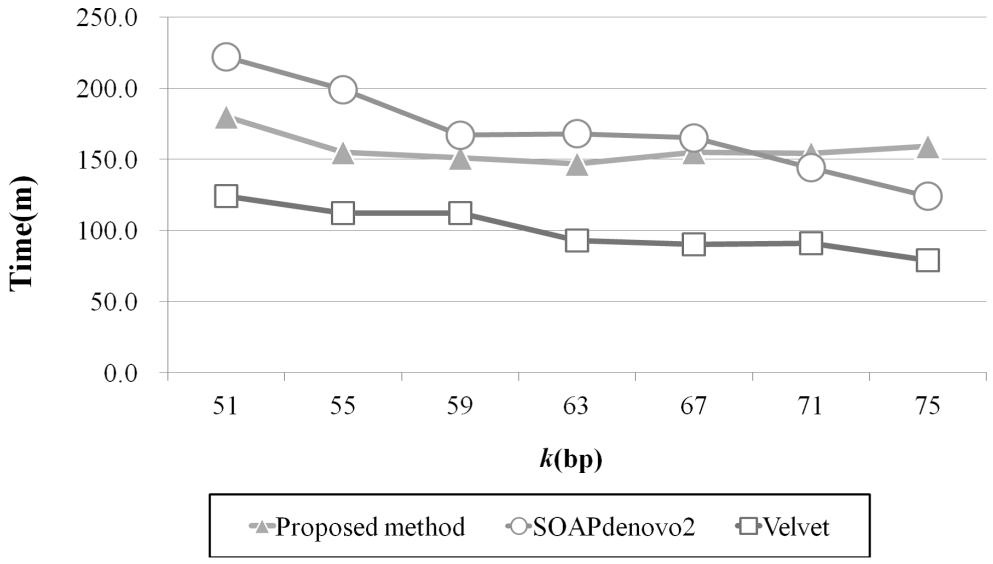


図 27. 実行時間の比較 (ヒトの 14 番染色体)

表 7. アセンブリ結果の比較 2(ヒトの 14 番染色体)

Assembler	カバー率 (%)	ギャップを除いたカバー率 (%)	エラー率 (%)
提案手法	79.39556	96.53553	0.71973
SOAPdenovo2	81.15031	98.66909	0.03682
Velvet	78.97097	96.01927	0.00026

6.3 考察

まず *E.coli* に対する実験結果について考察する．図 24 より，すべてのパラメータ k の場合で最大消費メモリ量を削減できており，平均で SOAPdenovo2 の約 13%，Velvet の約 19%の消費メモリ量で実行できた．このことから，*E.coli* のアセンブリでは 5.1 で述べた提案手法のグラフ構築方法は有効であると言える．一方，実行時間に関しては，図 25 より，SOAPdenovo2 よりわずかに提案手法が高速であり，Velvet が他手法よりも高速であった．この原因としては，提案手法では k-mer 間の k-1 文字の重複関係を意味するエッジを保存せず，ノードを辿る際に重複している (エッジで接続されている) ノードを計算している．このノード間のエッジの存在を調べる処理が全体の実行時間に影響を与えたと考えられる．しかし，5.2 節で述べたように，提案手法の分岐や閉路におけるエッジの選択 (除去) は非常に単純であるため，最終的な全体の実行時間に各手法との大きな差が表れなかったと考えられる．又，アセンブリの結果であるコンティグについては，表 5 から，提案手法では N50 値は他の手法に比べわずかに短い結果であった．これは，提案手法の分岐や閉路におけるエッジの選択 (除去) が非常に単純な処理であることが主な原因として挙げられる．一方，カバー率とエラー率には大きな差が表れなかった．

次に，ヒトの 14 番染色体に対する実験結果について考察する．消費メモリ量に関しては，図 26 より，すべての k の場合で最大消費メモリ量を削減できており，平均で SOAPdenovo2 の約 54%，Velvet の約 63%の消費メモリ量で実行できた．このことから，ヒトの 14 番染色体のアセンブリにおいて

も提案手法のグラフ構築方法は有効であり，提案手法の目的を達成できたと言える．一方，実行時間に関しては，図 27 より，提案手法は Velvet よりも若干遅い結果となった．この原因に関しても，*E.coli* のアセンブリの時と同様に，エッジ情報を保持せずに，エッジの有無をノードを辿る度に算したことが影響を与えたと考えられる．又，アセンブリの結果であるコンティグについては，表 6 より，提案手法では N50 値は他の手法に比べて短い結果となった．これは *E.coli* に対する実験結果の原因と同様に，提案手法の分岐や閉路におけるエッジの選択 (除去) は非常に単純であったことが主な原因として挙げられる．その一方で，表 7 から，カバー率とエラー等の評価に関しては大きな差は表れなかった．これは，*E.coli* における実験結果でも同様の傾向があった．このことから，提案手法の k-mer の情報の一部 (出現回数) のみを用いたアセンブリは，コンティグの精度をある程度維持しつつも消費メモリ量を大幅に削減できるということがわかった．

第7章　むすび

7.1　本研究のまとめ

本研究の目的は，de Bruijn グラフを用いた *de novo* アセンブリアルゴリズムの特徴と課題を明らかにし，次世代シーケンサから得られた大量のリードを用いて，大規模なゲノムに対しても *de novo* アセンブリが可能となるように，消費メモリ量の少ない *de novo* アセンブリアルゴリズムを提案することであった．本研究の成果は以下の通りである．

1. 既存の de Bruijn グラフを用いた *de novo* アセンブリの基本原理を明らかにし，消費メモリ量の増加に関与する課題を検討した．

2. *de novo* アセンブリにおける消費メモリ量増加の原因となる膨大な k-mer の情報を厳選し，de Bruijn グラフの特徴を利用したノードのみによるグラフの表現方法によって，消費メモリ量の少ない *de novo* アセンブリアルゴリズムを提案した．

3. 提案手法の有用性を確かめるために，*E. coli* K-12 strain MG1655 及びヒトの 14 番染色体から得られた，大量のリードに対し *de novo* アセンブリを行った．

4. 実験の結果から，本手法は *E.coli* K-12 strain MG1655 においては既存の手法の約 20%，ヒトの 14 番染色体においては既存の手法の約 60%の消費メモリ量でアセンブリが可能であることを確認し，本研究の提案手法の有効性を確認した．また，k-mer の情報の一部 (出現回数) のみを利用することで，消費メモリ量しつつある程度のコンティグの精度

を維持できるということを確認した．一方，実行時間がやや増加してしまい，コンティグの長さについてもやや短くなるという課題が浮き彫りとなった．

7.2 今後の課題・展望

本節では，本研究で得られた実験結果を踏まえ，今後の課題・展望について述べる．今後の課題としては，分岐による曖昧さや閉路に対する解決方法を見直し，実用レベルまでコンティグの長さ・精度を高めることが挙げられる．具体的には，分岐を検出した際に出現回数の比較のみではなく，どちらのエッジを選択すればより長いパスとなるかを調べた上でエッジを選択することでコンティグの長さを伸ばすことができると考えられる．また，直近の 2 つのノードの k-mer の出現回数のみを比較するのではなく，ある程度先のノードまで辿り，それらの k-mer の出現回数をも考慮するといった方法も，今後検討する必要がある．

また，更なるメモリ消費量の削減の検討も課題として挙げられる．具体的には，無効な k-mer をあらかじめ調べ，そのような k-mer をメモリ上に保存しないことで，より消費メモリを削減する方法である．本手法における無効な k-mer とは，その出現回数が一定数以下のものである．第 5 章でも述べたように，無効な k-mer はメモリ上 (ハッシュテーブル) に存在しても，それに対応するノードも存在しないものとして扱われる．とくに実際に次世代シーケンサから得たリードの k-mer には無効な k-mer が非常に多い．第 6 章の実験でも無効な k-mer が非常に多く現れており，コンティグに使用されたのは約一割の有効な k-mer であった．このような無効な k-mer をあらかじめ知ることができれば，メモリ上には保存する必要がなくなる大幅な消費メモリ量の削減が可能であると考えられる．しかし，k-mer の出現回数を知るためには全てのリードを調べる必要があり，一旦は有効無効の区別なく全ての k-mer をメモリ上に保存する必要があるという問題が生じる．そこで，リードの読み込みを複数回に分け，少しずつ k-mer をメモ

リ上に保存していくことでその問題の解決を図る．例えば，リードの読み込みを 2 回に分けることを考える．まずリードの前半部分を従来と同様に k-mer を抽出しメモリに保存する．前半部分の読み込みが終了したら，その時点で保存された k-mer が，リードの後半部分で何度出現しているかを調べる．その結果得られた出現回数が一定回数以下の k-mer は無効な k-mer であるため，メモリ上から除去する．そして，改めてリードの後半部分を読み込み，k-mer の抽出しメモリに保存していく．最後に，リードの後半部分でのみ出現した k-mer の出現回数を調べ，出現回数が一定回数以下の無効な k-mer をメモリ上から除去する．リードを 2 回に分けて読み込むことで，リードの読み込みに要する時間は 1.5 倍に増加してしまう．しかし，無効な k-mer がリード上に一様に出現し，なおかつ k-mer 全体の 9 割が無効な k-mer である仮定した場合，従来の約 55%に消費メモリ量を削減できる．更に分割回数を増加させれば，リードの読み込みに要する時間が増加するものの，消費メモリ量を大幅に削減できる可能性がある．今後は，上記のようにリードの読み込みを複数回に分けたときの消費メモリ削減量，実行時間の増加量について検討，そしてリードの読み込み要する時間が増加を防ぐための高速なデータ構造の検討を行う必要がある．

謝 辞

本研究を進めるに当たり，副指導教官の宇都宮大学工学研究科 外山史 准教授には学部，博士前期，博士後期課程時代を通じ，終始懇切丁寧なる御指導・御鞭撻を賜りました．ここに深く感謝致します．また，懇切丁寧なるご指導御指導，御鞭撻を賜りました指導教員の宇都宮大学工学研究科 東海林健二 教授に厚く御礼申し上げます．

本研究をまとめるに当たり，懇切丁寧なる御指導と御助言を賜りました副指導教官の宇都宮大学工学研究科 上村佳嗣 教授に深く感謝致します．

宇都宮大学工学研究科 阿山みよし 教授，渡辺裕 教授には副専門研修で御指導いただき，また，本研究をまとめるに当たって有益なる御助言を賜りました．ここに深く感謝致します．

また，本研究を進めるに当たり，大変有益なる御助言，御助力を賜りました筑波大学生命環境系 千葉親文 准教授に深く感謝致します．

本研究を進めるに当たり，有益なる御助言を賜りました宇都宮大学工学研究科 宮道壽一 名誉教授ならびに森博志 助教に深く感謝致します．

そして，本論文の完成に当たり，有益な御指導，御助言を賜りました，宇都宮大学工学研究科の諸先生方に心より感謝申し上げます．

本研究を進めるに当たり，数多くの御助言，御助力をいただいた外山研究室，東海林研究室，森研究室の大学院生，学部四年生，卒業生及び修了生に感謝致します．

常日頃からお世話になりました情報科学科技術官の川上典男氏，北本拓磨氏，月川淳氏，細島美智子志氏，杉山博美氏に心から感謝の意を表します．

最後に，研究を進めるに当たり，金銭・物資・精神など様々な面で支援を

していただいた家族に，深く感謝致します.

本論文は，ここでは記しきれないほどの本当に多くの方々のご協力によって成し得たものです．皆様には心より感謝申し上げます.

参考文献

[1] Maxam, A. M., and Gilbert, W., “A new method for sequencing DNA,” USA, 74, pp.560-564, 1977.

[2] Sanger, F., Nicklen, S. and Coulson, A.R., “DNA sequencing with chain-terminating inhibitors,” Proc. Natl. Acad. Sci. USA, 74, pp.5463-5467, 1977.

[3] Hernandez, D., Franois, P., Farinelli, L., sters, M. and Schrenzel, J., “De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer,” Genome Res., Vol. 18, No. 5, pp.802-809, 2008.

[4] Warren, R. L., Sutton, G. G., Jones, S. J. M., and Holt, R. A., “Assembling millions of short DNA sequences using SSAKE,” Bioinformatics, Vol. 23, No. 4, pp.500-501, 2007.

[5] Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T. , Magrini, V., Mardis, E. R., Dangl, J. L., and Jones, C. D., “Extending assembly of short DNA sequences to handle error,” Bioinformatics, Vol. 23, No. 21, pp.2942-2944, 2007.

[6] Zerbino, D. and Birney, E., “Velvet: Algorithms for de novo short read assembly using de Bruijn graphs,” Genome Res., Vol. 18, No. 5, pp.821-829, 2008.

[7] Simpson, J. T., Wong, K., Jackman, S., Schein, J., Jones, S. and Birol, I., "ABySS: A parallel assembler for short read sequence data," Genome Res., Vol. 19. No. 6, pp.1117-1123, 2009.

[8] Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J. and Wang, J., "De novo assembly of human genomes with massively parallel short read sequencing," Genome Res., Vol. 20, No. 2, pp.265-272, 2010.

[9] De Bruijn, N. G., "A combinatorial problem," Koninklijke Nederlandse Akademie v. Wetenschappen, Vol. 49, pp.758-764, 1946.

[10] Schatz, Michael, "Assembly of large genomes using cloud computing," http://schatzlab.cshl.edu/presentations/2010-07-23.Illumina.pdf, 2010.

[11] Miller, J. R., Koren, S., Sutton, G., "Assembly algorithms for next-generation sequencing data," Genomics, Vol. 95, No. 6, pp.315-327, 2010.

[12] Mullis, K. B., Faloona, FA., "Specific synthesis of DNA in vitro via a polymerase-catalyzed chain reaction," Methods Enzymol, Vol. 155, pp.335-350, 1985.

[13] International Human Genome Sequencing Consortium, "Initial sequencing and analysis of the human genome," Nature, Vol. 409, No. 6822, pp.860-921, 2001.

[14] Vender J. C. et al., "The sequence of the human genome," Science, Vol. 291, No. 5507, pp.1304-1351, 2001.

[15] Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L. and Rice, P. M., "The Sanger FASTQ file format for sequences with quality scores, and

the Solexa/Illumina FASTQ variants," Nucleic Acids Res., Vol. 38, No. 6, pp.1767-1771, 2010.

[16] Fujimoto, A., Nakagawa, H., Hosono, N., Nakano, K., Abe, T., Boroevich, KA., Nagasaki, M., Yamaguchi, R., Shibuya, T., Kubo, M., Miyano, S., Nakamura, Y. and Tsunoda, T., "Whole-genome sequencing and comprehensive variant analysis of a Japanese individual using massively parallel sequencing," Nature Genetics, Vol. 42, pp.931-936, 2010.

[17] Li, R. et al., "The sequence and de novo assembly of the giant panda genome," Nature, Vol. 463, No. 7279, pp.311-317, 2010.

[18] Conway, T. C. and Bromage, A. J., "Succinct data structures for as- sembling large genomes," Bioinformatics, Vol. 27, No. 4, pp.479-486, 2011.

[19] Bowe, A., Onodera, T., Sadakane, K. and Shibuya, T., "Succinct de Bruijn graphs," WABI, Lecture Notes in Computer Science, Vol. 7534, Springer, pp.225-235, 2012.

[20] Chikhi, R. and Rizk, G., "Space-efficient and exact de Bruijn graph rep- resentation based on a Bloom filter," WABI, Lecture Notes in Computer Science, Vol. 7534, Springer, pp.236-248, 2012.

[21] Chikhi, R., Limasset, A., Jackman, S., Simpson, J. and Medvedev, P., " On the Representation of de Bruijn Graphs," RECOMB, Lecture Notes in Computer Science, Vol. 8394, Springer, pp.35-55, 2014.

[22] Salzberg, S. L., Phillippy, A. M., Zimin, A., Puiu, D., Magoc, T., Ko- ren, S., Treangen, T. J., Schatz, M. C., Delcher, A. L., Roberts, M., Marais, G., Pop, M. and Yorke, J. A., "GAGE: A critical evaluation

of genome assemblies and assembly algorithms," Genome Res., Vol. 22, No. 3, pp.557-567, 2012.

[23] 遠藤友基，外山史，東海林健二，宮道壽一，"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，" FIT2011，第 2 分冊，pp.569-570，2011.

[24] Endo, Y., Toyama, F., Chiba, C., Mori, H. and Shoji, K., "De Novo Short Read Assembly Algorithm with Low Memory Usage," BIOINFORMATICS 2014 - Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms, pp.215-220, 2014.

[25] Endo, Y., Toyama, F., Chiba, C., Mori, H. and Shoji, K., "A Memory Efficient Short Read De Novo Assembly Algorithm," IPSJ transaction on Bioinformatics, Vol.8, pp.2-8, 2015.