Vol.2013-BIO-34 No.1
2013/6/27

Sparse k-mer graphアルゴリズムの評価とVelvetへの実装

吉川 舜亮[1]　石田 貴士[1]　関嶋 政和[1,2]　秋山 泰[1]

概要：近年，次世代シーケンサの開発により大量のゲノム情報を高速かつ低価格で読み取ることが可能となった一方で，一度に読み取ることができるリードの長さが第一世代のシーケンサが数百塩基であるのに比べて第二世代以降のシーケンサでは数十から百数十塩基と短くなっており，それに伴って様々なショートリード向けのアセンブラが考案されている．本研究では，まず高速，省メモリのアセンブラとして近年提案されたSparse Assemblerの性能評価を行い，主に最大メモリ使用量，実行時間の面で優れる一方で，出力されるコンティグの質は他のアセンブラに劣ることを確認した．そこで本研究では多くの研究で使用実績のあるVelvetにこのアルゴリズムを適用することで，Velvetのアルゴリズムを基本としながらも従来のものより短時間かつ少ないメモリ使用量のアセンブラを実装することを目指した．

キーワード：次世代シーケンサ，*de novo* アセンブリ，Velvet，Sparse Assembler

Evaluation of Sparse k-mer graph algorithm and its implementation on Velvet

Yoshikawa Shunsuke[1]　Ishida Takashi[1]　Sekijima Masakazu[1,2]　Akiyama Yutaka[1]

Abstract: Development of Next Generation Sequencing technologies has provided an unprecedented increase in DNA sequencing throuput while this technology produces shorter reads than previous technology. These two factors make a number of new short read *de novo* assemblers. In this report, by evaluation of Sparse Assembler, we clarified that this asssembler is useful at the point of memory usage and time, while producing less quality of contigs than other assemblers already developed. This study was made to assembly with less computer memory and in shorter time than previous study by implement Sparse k-mer graph algorithm on Velvet, which is the most widely used assembler.

Keywords: Next Generartion Sequencing, *de novo* assembly, Velvet, Sparse Assembler

1. はじめに

1.1 研究背景

近年，次世代シーケンサの開発により大量のゲノム情報を高速かつ低価格で読み取ることが可能となった一方で，一度に読み取ることができるリードの長さが第一世代のシーケンサが数百塩基であるのに比べて第二世代のシーケンサでは数十から百数十塩基と短くなっており，新規にゲノム決定を行う場合，従来に比べてより困難な *de novo* アセンブリを行う必要が生じている．*de novo* アセンブリとはシーケンサから得られたリードの情報のみから配列の再構築を行うことである．多くの場合，*de novo* アセンブリを行ってもゲノム配列の完全な再構築はされず，数千から数万塩基程度の長さのコンティグと呼ばれる配列を複数作成する．*de novo* アセンブリは膨大なメモリ空間と長い実

[1] 東京工業大学 大学院情報理工学研究科計算工学専攻
Department of Computer Science, Graduate school of Information Science and Engineering, Tokyo Institute of Technology
[2] 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

行時間を必要とする処理であり，現在ではそれが大きな問題となっている．例えば代表的なアセンブラの一つである Velvet[1] でヒトゲノムを解析すると，約 2TB ものメモリと何週間もの実行時間が必要である [2][3]．そのため，近年ではこの膨大なメモリ使用量と実行時間を削減するために様々な手法が考案され，それらの比較が行われている [4][5]．

そのような手法の中に，Ye らによって提案された Sparse k-mer graph アルゴリズム [6] と呼ばれる手法がある．Ye らは，このアルゴリズムを用いることで，メモリ使用量と実行時間の両方を同時に削減できるということを主張している．しかし，この手法を用いたアセンブラである Sparse Assembler は 2012 年に発表されたばかりで実際に使用された例がなく，その信頼性については疑問が持たれる．そのため，このアルゴリズムの性能に対する客観的な評価が必要であり，また，すでに発表されてからある程度の期間使用され，信頼性が高いとされるアセンブラソフトウェアがこのアルゴリズムによって改良されることが望ましい．

1.2 研究目的

本研究ではまず，Sparse k-mer graph アルゴリズムに基づくアセンブラである Sparse Assembler の性能の評価を比較実験によって行う．次に，多くの研究で使用実績のある *de novo* アセンブラである Velvet にこのアルゴリズムを適用することで，Velvet のアルゴリズムを基本としながらも，従来の Velvet より高速かつ少ないメモリ使用量で実行できるアセンブラを提案することを目指す．

1.3 本論文の構成

本論文では，第 2 節で本論文で改良を行う Velvet および，Sparse k-mer graph アルゴリズムの特徴をそれぞれ紹介する．第 3 節では Sparse Assembler について比較実験を行うことでそのアルゴリズムを評価する．第 4 節では Sparse k-mer graph アルゴリズムを Velvet に適用し，その評価を行う．第 5 節では本論文の結論と今後の課題をまとめる．

2. ゲノムアセンブリ

この節では Sparse Assembler と Velvet が利用する de Bruijn graph を用いた *de novo* アセンブリのアルゴリズムについて説明を行い，その後本研究で利用する Sparse k-mer graph アルゴリズムの特徴を説明する．

2.1 de Bruijn graph を用いるアプローチ

アセンブリを効率的に行う方法として考案された de Bruijn graph を用いた手法を説明する．de Bruijn graph を作成するためにまず，与えられたリードセットから 1 文字ずつずらしながら長さ k ずつに分割した配列（k-mer）を

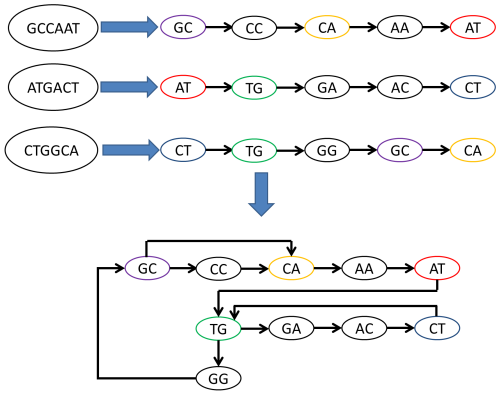

図 1 de Bruijn graph の作成の様子
Fig. 1 Set up of a de Bruijn graph

作り出す．次に，図 1 のように作成した k-mer を頂点，隣り合う k-mer 同士を辺としてグラフを作成し，同じ k-mer の頂点を一つにまとめることで複数のグラフを繋げて作成されたグラフが de Bruijn graph である．図 1 では長さ 6 のリードに対して k=2 としてグラフを作成している．

de Bruijn graph を用いるアプローチでは，グラフのすべての辺を通るパスを見つけることでアセンブリを完了させることができる．このすべての辺を通るパスを見つけるという問題はオイラー経路問題 (Eulerian Path Problem) と呼ばれ，効率的に解くことができるアルゴリズムが存在する [7]．ただし，このアルゴリズムを利用しても，*de novo* アセンブリではシーケンサのエラー訂正やリピート構造など解決すべき問題は残っている．de Bruijn graph を利用したアセンブラには Velvet，ABySS[8]，Euler_sr[9] などがある．

2.2 Velvet

本研究では de Bruijn graph を用いたアセンブラである Velvet に改良を加える．Velvet は velveth と velvetg という 2 種類の実行ファイルによって成り立っており，velveth では Roadmap というリード同士が繋がっていると考えられる場所の情報を保存するファイルを作成し，velvetg ではその Roadmap を入力ファイルとしてコンティグの作成を行う．

一般的に velvetg に比べて velveth は多くメモリを必要とする．そのため，今回の実装では velveth にのみ改良を加え，特にメモリ使用量の削減に焦点を置く．

2.3 Sparse k-mer graph アルゴリズム

Sparse k-mer graph アルゴリズムは，メモリ使用量と実行時間の削減を目指して 2012 年に Ye[6] らによって開発されたアルゴリズムである．

一般的な de Bruijn graph を用いたアルゴリズムでは 1 本の長さ l のリードに対して k-mer を $(l-k+1)$ 本保存して

表 1　性能評価における指標とその単位
Table 1　Evaluation criteria and their units

指標	単位
最大メモリ使用量	GB
実行時間	秒
N50 値	bp
最大コンティグ長	bp
コンティグの本数	本
エラーの個数	個

それらの重なりを探していく．しかし，Sparse Assembler ではすべての k-mer を保存するのではなく数本の k-mer をスキップしながら保存していく．このスキップの大きさを以下では g を用いて記述する．g=1 のときは，間をスキップせずに全ての隣接した k-mer を保存することを意味し，従来の de Bruijn graph と同じグラフを作成する．g=2,3... と大きくなる度に飛ばす k-mer の個数が 1 個ずつ大きくなる．このアルゴリズムを適用することで保存する k-mer の個数は約 1/g となり，それに従って最大メモリ使用量の削減および実行時間の短縮が実現できる．

3. Sparse Assembler の性能評価

ここでは Sparse k-mer graph アルゴリズムを用いたアセンブラである，Sparse Assembler の性能を評価する．

3.1 比較実験における性能評価指標

Sparse Assembler の性能を評価するにあたって，最大メモリ使用量，実行時間，N50 値 [10]，最大コンティグ長，コンティグ数，エラーの個数の 6 項目を評価の指標に用いた．用いた指標とそのそれぞれの単位を表 1 に示す．

N50 値とは，アセンブリ結果の質の評価において重要な指標であり，N50 値が大きいほどアセンブリが成功していることを意味している [10]．具体的には，出力されたコンティグを長いものから順に並べたときに，予想される全ゲノム長の 50%の位置に来るコンティグの長さのことを表している．

また，ここで用いているエラーの個数は，Indel エラーおよび misjoin エラーの合計の個数を示す．これらのエラーは典型的なアセンブリミスとして知られており，様々な場面でアセンブリ結果を評価する指標となっている [11]．

3.2 計算機環境と使用データ，測定方法

3.2.1 計算機環境

実験には表 2 に示す計算機を利用した．

3.2.2 使用データ

本実験では 2 種類のデータセットを使用して実験を行った．これらのデータセットは Salzberg らによるアセンブラの比較論文 [12] で利用されていたデータセットである．2 種類のデータセットの生物種は，*Staphylococcus aureus*（黄色ブドウ球菌），*Rhodobacter sphaeroides*（紅色細菌）であり，その詳細を表 3 に示す．なお，2 種のデータセットはそれぞれ *S. aureus*，*R. sphaeroides* として表 3 内では表記する．

表 2　計算機環境
Table 2　Computing environment used

CPU	Intel Xeon X5675 3.07 GHz (6 cores) × 2 ソケット
Memory	96GB
OS	CentOS 5.8
Compiler	gcc 4.1.2

表 3　使用したデータセット: Library1 と Library2 ではインサート長に違いがあるため，同時に利用することでより良いアセンブリ結果を得られる．
Table 3　Datasets: There are differences between insert length of Library1 and Library2.Using them simulataneously improves quality of assembly results.

	生物種	*S. aureus*	*R. sphaeroides*
	ゲノムサイズ (MB)	2.90	4.60
Library1	平均リード長 (bp)	101	101
	インサート長 (bp)	180	180
	リード数	1,294,104	2,050,868
	カバレッジ	45	45
Library2	平均リード長 (bp)	37	101
	インサート長 (bp)	3500	3500
	リード数	3,494,070	2,050,868
	カバレッジ	45	45
	シーケンサ	Illumina Genome Analyzer II	

なお，ここで用いたデータセットはいずれも paired end read のレイアウトでシーケンシングを行っている．paired end read は，シーケンサからリード情報を読み取るときの方式の一つで，データ配列を異なる両端から読む方式である．例えば仮に，500bp にデータ配列を切って揃えて両端から 35bp だけ配列を読むと，片方の端から長さ 35bp の配列情報，もう片方の端から長さ 35bp の配列情報，両者の配列間の距離が 430bp であること，という 3 つの情報を得られることになる．

3.2.3 測定方法

最大メモリ使用量は，アセンブリの実行中に ps コマンドを 1 秒ごとに実行してその内容をテキストファイルに書き込むことで測定を行った．ps コマンドは，実行した時点でのプロセスの状態を一覧で表示するので，上記の実行を行うことで 1 秒ごとに使用しているメモリ量を測ることができる．最大メモリ使用量は，そこで得た数値のなかで最大のものを抜き出して利用した．実行時間の測定には上記

の ps コマンドの実行回数から判断した．N50，最大コンティグ長，出力コンティグ数，エラーの個数の測定には上記の比較論文 [12] で用いられていた比較用ツールを利用した [13].

3.3 他のアセンブラとの比較

ここでは，Sparse Assembler の性能を評価するために他のアセンブラと同じデータセットを利用して実行を行い，上記の指標を用いての比較を行った．なお，比較においてすべてのアセンブラで k-mer のサイズを示すパラメータの設定を必要とするが，23 から 59 までのすべての奇数を k に入力して実行し，最も N50 値が大きくなるものを選んで比較を行った．なお，ABySS についてはパラメータによってエラー数が極端に大きくなるものがあるので，他のアセンブラのエラー数と同じ程度（20 個以下）のエラー数のパラメータのなかから最も N50 値が大きくなるものを選んだ．また，それに加えて Sparse Assembler ではそれぞれの k の値に対して g の値を 1 から 15 まですべてを入力して実行し，同様にエラー数が極端に大きくないもののなかで最も N50 値が大きくなるものを選んで比較を行った．いずれのデータセットを用いても実行結果は似た傾向を示したため，ここでは *Staphylococcus aureus* を用いたアセンブリ結果のみを表 4 に示す.

表 4 *Staphylococcus aureus* におけるアセンブリ結果の比較
Table 4 Comparison of assembly results for *Staphylococcus aureus*

アセンブラ	Velvet	ABySS	Sparse
パラメータ	k=27	k=29	k=25,g=5
最大メモリ使用量 (GB)	2.75	2.77	0.865
実行時間（秒）	164	375	104
N50 値 (bp)	24726	27375	19464
最大コンティグ長 (bp)	93142	86623	51888
コンティグの本数（本）	322	248	432
エラー（個）	15	15	5

表 4 の結果を比較すると，Velvet の最大メモリ使用量は他のアセンブラに比べて大きい一方で，高品質なアセンブリ結果を出力していることがわかる．ABySS はこのデータセットではややエラーが多く，実行時間の面で劣るものの N50 値，最大コンティグ長はこれらのアセンブラの中でも充分に大きな値を得ている．

そして Sparse Assembler は Velvet や ABySS に比べて少ないメモリ使用量で実行をすることができている．また，Sparse Assembler は実行時間も他のアセンブラに比べて短く，実行時間を延ばすことなくメモリ使用量の削減に成功していると言える．ここで得られたアセンブリ結果については，Velvet や ABySS には及ばないものの最低限の値は出力できており，充分に利用価値があるということが N50 値や最大コンティグ長から判断できる．

大幅なメモリ使用量の削減や実行時間の短縮は，保存する k-mer の個数を減らすことによって実現できたと考えられる．

以上により N50 値等のアセンブリ結果の質とのトレードオフではあるが，Sparse k-mer graph アルゴリズムはメモリ使用量，実行時間を削減する手段として効果的であるということがわかった．Sparse Assembler のコンティグの質が低いのは，Velvet などの既存のアセンブラに比べてエラー削除等のポストプロセスが不十分だったためと考えられる．

4. Sparse k-mer graph アルゴリズムの Velvet への実装と評価

3 節で記述した通り，Sparse Assembler は大幅な実行時間の短縮とメモリ使用量の削減に成功しており，Sparse k-mer graph アルゴリズムの有用性が示された．その一方で出力されたコンティグの長さは Velvet や ABySS などの従来のアセンブラに比べて短い結果となっている．

そこでこの節では，より高品質なコンティグを出力する Velvet に Sparse k-mer graph アルゴリズムを適用し，最大メモリ使用量や実行時間を従来の Velvet に比べて削減できるか，アセンブリ結果の質はどうなるかなどについて評価を行う．

4.1 Sparse k-mer graph アルゴリズムの Velvet への実装

ここでは Sparse k-mer graph アルゴリズムの Velvet への実装について説明する．Velvet の実行ファイルは Roadmap を作成するまでの velveth と，その Roadmap を読み込んでコンティグを出力するまでの velvetg の 2 つに分かれている．今回の実装では，velveth で Roadmap を作成する際に利用するハッシュテーブルへのアクセスにおいて，g=2 として velveth と同じ作業を行う．その際，従来の Velvet にフォーマットを合わせて Roadmap を作成する．こうすることで出力された Roadmap をそのまま velvetg の入力ファイルとして利用でき，Roadmap の違いが出力されるコンティグに与える影響を考察することができる．

4.2 実験

Velvet に付属されている人工的に作成されたテストデータと，3 節で利用したリアルデータのデータセットを利用して Sparse k-mer graph を実装した Velvet の実行を行った．

4.2.1 テストデータを用いた実験

Velvet に付属されたテストデータ（表 5）を用いて実験を行った．しかし，リアルデータを用いた実験と結果が似た傾向を示したのでここでは紙面の都合で割愛する．なお，このテストデータは人工的に作られたデータであり，

シーケンシングエラーを含まないデータである．

表 5 Velvet に付属されたテストデータ（人工データ）
Table 5 Test data attached to Velvet (artificial data)

ゲノムサイズ (MB)	101
平均リード長 (bp)	35
インサート長 (bp)	100
リード数	142857

4.2.2 リアルデータを用いた実験

3 節で利用した *Staphylococcus aureus* ゲノムから得られたリードを用いて提案手法と元の Velvet および Sparse Assembler を実行した結果を表 6 に示した．ここで使用したパラメータは k=27 で，Sparse Assembler では g=2 を用いた．なお，パラメータを変えても出力結果は似た傾向を示す．

表 6 *Staphylococcus aureus* におけるアセンブリ結果の比較 (k=27)
Table 6 Comparison of assembly results for *Staphylococcus aureus* (k=27)

アセンブラ	提案手法	Velvet	Sparse
最大メモリ使用量 (GB)	1.60	2.75	2.13
実行時間（秒）	143	164	175
N50 値 (bp)	436	124726	14313
最大コンティグ長 (bp)	2758	93142	50336
コンティグの本数（本）	6503	322	505
エラーの個数	9272	15	6

4.3 評価

最大メモリ使用量，実行時間，アセンブリ結果のそれぞれについて評価を行う．

4.3.1 最大メモリ使用量

最大メモリ使用量についてリアルデータの結果を比較すると，従来の Velvet が 2.75GB であるのに対して提案手法では 1.60GB となっており，約 4 割のメモリ使用量の削減に成功している．これは，保存する k-mer の個数を減らすことによって半分近くのメモリを節約できたと考えられる．

4.3.2 実行時間

実行時間については，全体では従来の Velvet が 164 秒であるのに対して提案手法では 143 秒となっており，約 1 割の実行時間が短縮されていることがわかる．ここで実行時間を velveth の部分と velvetg の部分に分けた結果を表 7 に示す．表 7 に示されている通り，velveth では従来の Velvet の 49 秒に対して提案手法では 29 秒と約 4 割の実行時間の短縮に成功しているが，velvetg では実行時間に大きな変化はない．これは，velvetg では利用する k-mer の本数が少なくなると処理の時間が短縮されるものの，コンティグの本数が多くなることによってグラフの単純化やエラー処理にかかる時間が長くなってしまい，結果的に提案手法の velvetg の実行時間は従来の Velvet の velvetg と変わらないものとなってしまったためと考えられる．Velvet の実行時間全体において velvetg が占める割合は velveth に比べて大きいため，全体の実行時間の短縮は約 1 割に留まっている．

表 7 velveth と velvetg の実行時間の比較（秒）
Table 7 Computing time break for velveth and velvetg. (in seconds)

	velveth	velvetg	合計
提案手法	29	114	143
Velvet	49	115	164

4.3.3 アセンブリ結果

アセンブリ結果については，従来の Velvet に比べて提案手法は N50 値，最大コンティグ長が共に低い値を示しており，大きくアセンブリの質を落としてしまっている．

今回の提案手法では，リードから得られる k-mer を 1 つ飛ばし (g=2) で取り出して利用した．Sparse Assembler では 1 本の k-mer を保存するときに，同時に g の大きさに対応して長く前後の配列情報を保存していたが，今回我々はこの処理を実装しなかった [6]．そのため，k の値に対して充分に長くリード同士が重なっていたとしても，お互いに保存される k-mer がずれてしまうことがあるため，リード同士が重なっているという情報が減ってしまい，繋ぐことのできるリードの数が従来手法に比べて減ってしまう．今回の実装では g=2 に限定し，カバレッジが充分に厚いデータを用いることで仮にアセンブリ結果が下がってしまったとしても，最低限の質を保ってアセンブリを行うことができると予測したが，残念ながら実際の実行結果は実用に耐えるものとは言えない結果となってしまった．

以上のことから，今回の実装で得られるリードの重なりの情報は，有用なコンティグを作成するためには不十分な量となってしまうということがわかった．

また，提案手法に含まれるエラーの大半は Indel エラーによるものであった．これは，インサート長の情報により間の配列がわからなくてもリード間の距離によって一見長く見えるコンティグを作り出しているためであるといえる．多くの paired end read のレイアウトのデータセットを利用するアセンブラでは，このように配列がわからない部分をブランクで埋めてコンティグを作り出す機能を持つ．今回の実行ではリード同士が繋がらないままこの機能が従来の Velvet と同様に使用され，結果的に長さ 5 以上のブランクを持ったコンティグが多く作られてしまい，大量の Indel エラーが出力されてしまったと考えられる．

5. おわりに

最後に，本研究のまとめ，および今後の課題を記述する．

5.1 本論文の結論

de novo アセンブリにおける使用メモリと実行時間の削減は重要な課題となっている．本論文では Sparse k-mer graph アルゴリズムに着目し，比較実験によってこのアルゴリズムの評価を行い，Sparse Assembler では k-mer の保存数を減らすことでメモリ使用量と実行時間を同時に削減していることがわかった．

これらの評価から，Velvet にこのアルゴリズムを適用することで，Velvet のアルゴリズムを基本としながらも，メモリ使用量，実行時間がともに削減されたアセンブラを開発できると考え，実装を行った．具体的な実装方法としては，Velvet の前半部分である velveth にこのアルゴリズムを適用することで，Roadmap の作成におけるメモリ使用量および実行時間の効率化を目指した．

実際にこの提案手法のアセンブラを実行したところ，従来の Velvet に比べて最大メモリ使用量は約 4 割の削減に成功した．これは，Roadmap 作成時に保存する k-mer の本数がほぼ半減されることによって実現したと考えられる．

また，実行時間に関しては従来の Velvet に対して約 1 割程度の短縮に留まった．これは，利用する k-mer の本数を減らすことによって velveth では実行時間が短縮されたが，Velvet の実行時間のうち多くを占める velvetg では実行時間が短縮されなかったためである．

実行時間と最大メモリ使用量は従来の Velvet から改善されたものの，提案手法によって作成されたコンティグはきわめて短いものとなってしまった．これは，Sparse Assembler とは異なり，各 k-mer に対して前後情報を長めに保存するということを今回の提案手法では行わず，Roadmap に記述されたリードの重なりの情報が不十分になってしまったためであると考えられる．

アセンブリ結果は悪化してしまったものの，本研究で最も重要な目的としていた最大メモリ使用量の削減が成功したことは大きな成果である．5.2 であげる課題を克服することで，同分野において重要な課題であるメモリ不足が解消できる可能性もあると考えられる．

5.2 今後の課題

本研究の実装では実行時間，最大メモリ使用量をともに削減できたものの，アセンブリ結果は実用に耐えないものとなってしまった．

アセンブリの質を落とすことなくメモリ使用量や実行時間を改善する方法として，以下の 2 点が考えられる．

- 今回は実装しなかった各 k-mer に対して前後情報を長めに保存するということを実装することで，アセンブリの質を落とさずにメモリ使用量を削減できると考えられる．新たな実装により従来の Velvet とまったく同じ Roadmap が作成された場合，velvetg におけるメモリ使用量と実行時間およびアセンブリ結果は従来手法と同じものとなるはずである．また，新たな実装により velveth の実行時間が削減できたとしても，大幅な実行時間の短縮のためには velvetg の改良も必要になると考えられる．
- 今回は Velvet に Sparse Assembler のアルゴリズムを実装したが，逆に Velvet のアルゴリズムを Sparse Assembler に実装するという方法もある．上記のように Velvet に Sparse Assembler のアルゴリズムを実装させるためには Velvet 全体のアルゴリズムを大幅に変更する必要がある．そのため，むしろエラー削除等のポストプロセスを Velvet のアルゴリズムから Sparse Assembler へと適用させる方が効率的にアセンブリの質の向上およびメモリ使用量と実行時間の削減が実現できる可能性もあると考えられる．

参考文献

[1] Zerbino D, Birney E. “Velvet: algorithms for de novo short read assembly using deBruijn graphs.”, *Genome research*, Vol. 18, No. 5, pp. 821-9, 2008.
[2] Schatz M. “Assembly of Large Genomes using Cloud Computing.”, 2010. url = http://schatzlab.cshl.edu/presentations/2010-07-23.Illumina.pdf.
[3] Venter C, Adams D, Myers W, et al. “The sequence of the human genome.”, *Science*, Vol. 291, No. 5507, pp. 1304-51, 2001.
[4] Bao S, Jiang R, Kwan W, Wang B, Ma X, Song YQ, et al. “Evaluation of next-generation sequencing software in mapping and assembly.”, *J Hum Genet*, Vol. 56, No. 21, pp. 406-14, 2011.
[5] 杉浦典和, “ハッシュテーブルの分割による de novo アセンブリの改良”, 情報処理学会研究報告, BIO-29, 2012
[6] Ye C, Ma Z, Cannon C, et al. “Exploiting sparseness in de novo genome assembly.”, *BMC Bioinformatics*, 13, suppl. 6:S1, 2012.
[7] Fleischner H. *Eulerian Graphs and Related Topics*. North-HOLLAND, 1990.
[8] Simpson J, Wong K, Jackman S, et al. “ABySS : A parallel assembler for short read sequence data.”, *Genome Research*, vol. 19, NO 6, pp. 1117-23, 2009.
[9] Chaisson M, Pevzner P. “Short read fragment assembly of bacterial genome.s”, *Genome Research*, vol. 18, NO 2, pp. 324-30, 2008.
[10] Lander E, Linton L, Birren B, et al. “Initial sequencing and analysis of the human genome.”, *Nature*, vol. 409, NO. 6822, pp. 860-921, 2001.
[11] Phillippy A, Schatz M, Pop M. “Genome assembly forensics: finding the elusive mis-assembly.”, *Genome Biol*, vol. 9, NO. 3, R55, 2008.
[12] Salzberg S, Phillippy A, Zimin A, et al. ”GAGE: A critical evaluation of genome assemblies and assembly algorithms.”, *GENOME RESEARCH*, vol. 22, NO. 3, pp.

557-67, 2012.
[13] http://gage.cbcb.umd.edu/data/index.html