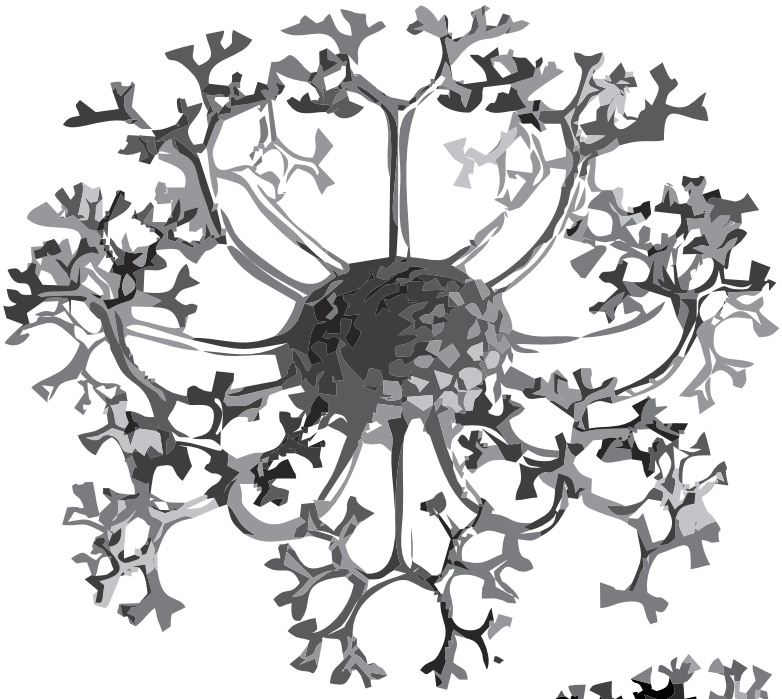
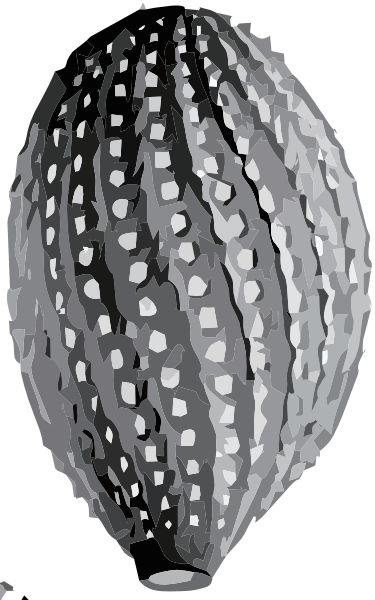
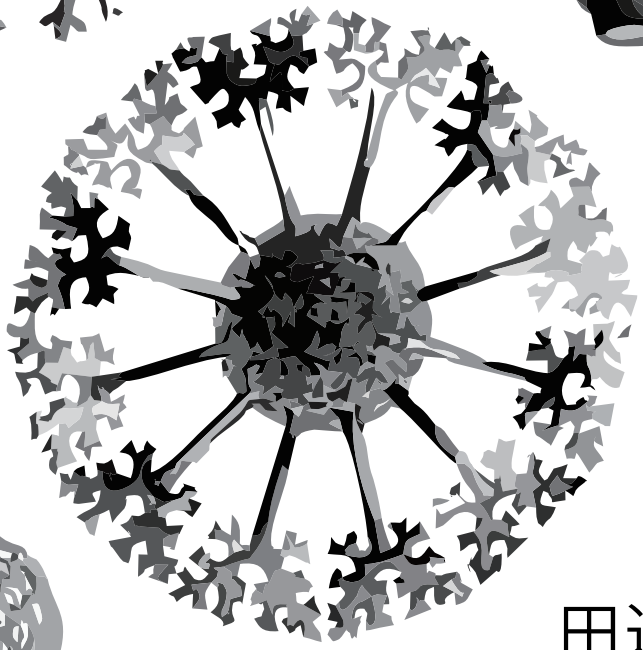
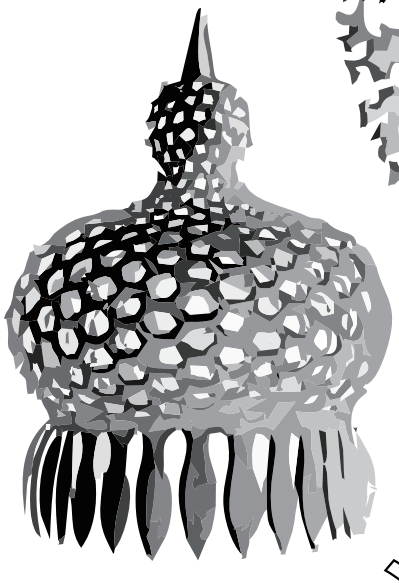

田辺晶史

生態学のための メタバーコーディング とDNAバーコーディング

生態学のためのメタバーコーディングと DNA バーコーディング

田辺晶史

2016 年 7 月 1 日

目次

はじめに

本書は、筆者が開発した塩基配列アセンブラー「Assams」と、配列のクラスタリングとホスト生物同定のためのパッケージ「Claident」の利用を普及促進するために執筆を開始しました。これらのプログラムの利用方法のみならず、データの取得方法から DNA バーコーディングによる生物の同定方法まで解説しています。

細菌などの研究においては既に広く利用されているメタバーコーディングですが、細菌以外の分類群でも幅広い応用が考えられます。例えば、土壌や動植物体内の真菌、水中の動植物プランクトンなどにも適用可能だと思います。水中の溶け出している大型動植物の環境 DNA でも同様の解析が可能でしょう。本書では、環境 DNA やメタゲノムから、DNA バーコーディングに利用される領域を増幅し、比較的長い解読長の第 2 世代以降のシーケンサーで配列を読み取って、DNA バーコーディングによって配列のホスト生物を特定することで生物の在・不在を調査する方法を解説します。また、メタバーコーディングに限らず、様々な分野で応用が進みつつある DNA バーコーディングを用いて生物を同定する方法についても解説しています。

本書が皆さんの役に立つことができましたら幸いです。この機会を与えて下さった京都大学大学院人間・環境学研究科の東樹宏和博士、水産研究・教育機構中央水産研究所の長井敏博士と、本書をお読みの皆さんに感謝します。

凡例

本書ではコンピュータに入力するコマンドやその結果を表記する際に以下のように記述しています。

```
# コメント
> command argument1 \
argument2 \
argument3 ↓
output of command
> command argument1 argument2 argument3 ↓
output of command
```

上記の例では command argument1 argument2 argument3 という全く同じコマンドを 2 回実行しており、コマンド実行後に output of command がコマンドにより表示されています。ここで、#から改行まではコメントを表しており、入力の必要はありません。行頭の>とそれに続くスペースはコマンドの入力の開始を表しており、↓までがコマンドとオプションの入力内容になります。>とそれに続くスペースはあくまで入力の開始を示すためのものですので、入力しないで下さい。↓は入力の終端を表し、ここで Enter キーを押すことを指示する記号です。↓を入力しないようにして下さい。なお、コマンドとオプションを見やすくするためにコマンドやオプションの途中に改行を意図的に入れることがありますが、そのような改行の直前には \ を記してあります。したがって、\ が直前にある改行はコマンドの終端や改行入力の指示を意味しません。また、表示環境によってはワードラップ機能により筆者の意図しない改行が入ってしまうことがありますが、これもコマンドの終端や改行入力の指示を意味しませんので注意して下さい。

また、本書では様々なファイルを使用しますが、その内容は以下のように記述しています。

```
| 1 行目の内容
| 2 行目の内容
```

この例では、行頭の|とそれに続くスペースはファイル内の行頭を表しており、ファイル作成の際は入力しないように注意して下さい。これは、ワードラップ機能による筆者の意図しない改行とファイルに入力すべき改行を区別できるようにするためのものです。

第 0 章

ソフトウェアのインストールと環境設定

本書では、Debian GNU/Linux jessie (以下 Debian)、Ubuntu Linux 14.04 LTS (以下 Ubuntu) を利用環境として想定しています。Windows 環境の方は、Linux のどちらかをインストールして環境を整えて下さい。インストールには、インストール用 CD・DVD・USB メモリが使えます。HDD や SSD が 1 台しかない場合、Windows 用の EaseUS Partition Master などのパーティションリサイズが可能なソフトを使ったり、インストール用 CD・DVD・USB メモリ内に用意されている機能を使って Windows 用の領域を縮小して容量を空ける必要があります。HDD または SSD を追加してスペースを確保していただいても良いでしょう。USB メモリや USB 接続 HDD にインストールすることも可能です。Ubuntu は見た目・操作性の異なる何種類かがありますが、標準の Ubuntu よりも Xubuntu というのがおすすめです。

Mac でも、これらの OS がインストールできます。空きディスクがない場合はディスクユーティリティを用いて Linux インストール用のスペースを空ける必要があります。ディスクユーティリティで HDD・SSD をクリックして、MacOS X が使用している領域を縮小して下さい。そして rEFIt や rEFInd というソフトをインストールして起動時に起動デバイス・起動 OS 選択メニューが出るようにします (インストール後に 2 回の再起動が必要です)。そうすると、Linux インストール用 CD・DVD・USB メモリから起動できるようになりますので、そこから起動してインストールして下さい。くれぐれも既存の OS 用領域を誤って削除しないようにご注意下さい。空いているディスク領域がある場合はディスクユーティリティでの操作は必要ありません。rEFIt や rEFInd を導入してインストール CD・DVD・USB メモリから起動し、空いている領域に Linux をインストールしていただければ結構です。また Mac でも、インストール先を USB メモリや USB 接続 HDD にすることができます。

なお、本書は 64bit 対応の Intel・AMD 製 CPU 搭載機しか想定していません。それ以外の環境でも動作するでしょうが、自力で解決していただく必要があります。導入する Linux は 64bit 版にして下さい。32bit 版では大容量メモリを活かすことができません。

0.1 Linux への Claident および同定用データベースと必要なプログラムのインストール

以下のコマンドをターミナルかコンソールで実行して下さい。必要なものが全てインストールされます。sudo を利用可能なユーザーである必要があります。途中、sudo の実行時に何回かパスワードを質問されますので、入力して下

さい。

```
> mkdir -p ~/workingdirectory ↓
> cd ~/workingdirectory ↓
> wget https://www.claident.org/installClaident_Debian.sh ↓
> sh installClaident_Debian.sh ↓
> wget https://www.claident.org/installOptions_Debian.sh ↓
> sh installOptions_Debian.sh ↓
> wget https://www.claident.org/installDB_Debian.sh ↓
> sh installDB_Debian.sh ↓
> wget https://www.claident.org/installUCHIMEDB_Debian.sh ↓
> sh installUCHIMEDB_Debian.sh ↓
> cd .. ↓
> rm -r workingdirectory ↓
```

標準のインストール先は/usr/local 以下になります。また、Permission denied と言われた直後にパスワードを尋ねられたりしますが、パスワードに答えることで進行する場合は問題ありません。これは、最初は sudo なしにユーザー権限でインストールを試み、うまく行かなかった場合 (ここでエラーになる) に初めて sudo を使って管理者権限でインストールしようとする (ここでパスワードを尋ねられる) ためです。

もしも外部ネットワークへのアクセスにプロキシを設定する必要がある場合は、上記のコマンド実行の前に以下のコマンドを実行して環境変数を設定しておいて下さい。これにより、外部へはプロキシを経由してアクセスが行われるようになります。

```
> export http_proxy=http://server.address:portnumber ↓
> export https_proxy=$http_proxy ↓
> export ftp_proxy=$http_proxy ↓
> export all_proxy=$http_proxy ↓
```

なお、ユーザー名とパスワードを用いた認証が必要なプロキシでは、以下のようにして下さい。

```
> export http_proxy=http://username:password@server.address:portnumber ↓
> export https_proxy=$http_proxy ↓
> export ftp_proxy=$http_proxy ↓
> export all_proxy=$http_proxy ↓
```

0.1.1　バージョンアップの場合には

全てのプログラムとデータベースを更新する場合は、インストールと同様の手順でコマンドを実行して下さい。この手順では、Assams、Claident、PEAR、VSEARCH、Metaxa、ITSx が/usr/local 以下へインストールされ、/usr/local/share/claident 以下へ NCBI BLAST+ と同定用 BLAST データベース・Taxonomy データベース、その他の依存プログラム等がインストールされます。NCBI BLAST+ や BLAST データベースが別途システムにインストールされている場合でも、Claident が利用する BLAST+ やデータベースとは共存可能です。

なお、以下のコマンドを利用することで、一部の更新を無効化して他のものだけ更新することができます。

```
> mkdir -p ~/workingdirectory ↓
> cd ~/workingdirectory ↓
# Assams の更新を無効化
> touch .assams ↓
# Claident の更新を無効化
> touch .claident ↓
# PEAR の更新を無効化
> touch .pear ↓
# VSEARCH の更新を無効化
> touch .vsearch ↓
# NCBI BLAST+ の更新を無効化
> touch .blast ↓
# プログラムを更新> wget https://www.claident.org/installClaident_Debian.sh ↓
> sh installClaident_Debian.sh ↓
# HMMer の更新を無効化
> touch .hmmer ↓
# MAFFT の更新を無効化
> touch .mafft ↓
# Metaxa の更新を無効化
> touch .metaxa ↓
# ITSx の更新を無効化
> touch .itsx ↓
# オプションプログラムを更新> wget https://www.claident.org/installOptions_Debian.sh ↓
> sh installOptions_Debian.sh ↓
# 同定用データベース「overall」の更新を無効化
> touch .overall ↓
# 同定用データベース「prokaryota」の更新を無効化
> touch .prokaryota ↓
# 同定用データベースを更新> wget https://www.claident.org/installDB_Debian.sh ↓
> sh installDB_Debian.sh ↓
# キメラ検出用データベース「rdp」の更新を無効化
> touch .rdp ↓
# キメラ検出用データベース「unite」の更新を無効化
> touch .unite ↓
# キメラ検出用データベースを更新> wget https://www.claident.org/installUCHIMEDB_Debian.sh ↓
> sh installUCHIMEDB_Debian.sh ↓
> cd .. ↓
> rm -r workingdirectory ↓
```

0.1.2 標準以外の場所にインストールする

前述の方法でインストールすると、標準では/usr/local 以下へインストールされます。コマンドは/usr/local/bin にインストールされます。他のプログラム (特に旧バージョン) との共存を図りたい場合は、インストール場所を変更して、使用する場合に環境変数 PATH を変更するのがいいでしょう。以下のコマンドで標準とは異なる場所へインストールすることができます。

```
> mkdir -p ~/workingdirectory ↓
> cd ~/workingdirectory ↓
> export PREFIX=インストール先にしたい場所↓
> wget https://www.claident.org/installClaident_Debian.sh ↓
> sh installClaident_Debian.sh ↓
> wget https://www.claident.org/installOptions_Debian.sh ↓
```

```
> sh installOptions_Debian.sh ↓
> wget https://www.claident.org/installDB_Debian.sh ↓
> sh installDB_Debian.sh ↓
> wget https://www.claident.org/installUCHIMEDB_Debian.sh ↓
> sh installUCHIMEDB_Debian.sh ↓
> cd .. ↓
> rm -r workingdirectory ↓
```

なお、使用する際には、以下のコマンドを実行する必要があります。

```
> export PATH=インストールした場所/bin:$PATH ↓
```

毎回上記コマンドを実行するのが面倒な場合は、~/.bash_profile や~/.bashrc の末尾に上記コマンドを記述しておけばいいでしょう。

0.1.3　複数のバージョンを共存させる

Claident は標準のインストール先にインストールすると以前のバージョンがあっても上書きされますが、前述の通り標準以外の場所にインストールすれば複数のバージョンを共存させることができます。ただし、ユーザーのホームディレクトリ (/home/ユーザー名) の直下もしくは/etc/claident に作成される設定ファイル.claident は共存できませんので、本設定ファイルは使用するバージョンに合わせて切り替える必要があります。ユーザーのホームディレクトリの設定ファイルが優先的に読み込まれますので、複数バージョンを共存させる場合はそれぞれ別のユーザーを用意してそのユーザーのホームディレクトリ以下にインストールしていただければ、ユーザーを切り替えることでバージョンを切り替えることができます。

第 1 章

次世代シーケンサーによる大量サンプルの塩基配列決定

ここでは、Roche GS シリーズシーケンサーや、Ion PGM、Illumina MiSeq によるタグ付き塩基配列決定法について簡単に説明します。これらのシーケンサーは、いずれも 400bp 以上の長さの塩基配列解読能力があり、メタバーコーディング、DNA バーコーディングに適しています。ただし、MiSeq は両側から解読したもの (ペアエンドリード) を連結する必要があります。したがって、両側の配列間で重複ができるよう、できれば 400bp 程度、長くても 500bp 程度のアンプリコンができるように PCR しなくてはなりません。連結せずに片側ずつ解析することもできますが、reverse 側配列は質がかなり低いため、あまり有用ではないと思います。

次世代シーケンサーは非常に多くの塩基配列を解読できますが、かつてはランニングコストが高く、1 度に多数のサンプル由来配列を由来サンプルがわかるように解読することができませんでした。しかし、ランニングコストはかなり下がってきた上、塩基配列に由来サンプル識別用の数塩基の「MID (multiplex identifier) タグ」を予め付加して解読することで、多数のサンプルの塩基配列を混合 (multiplex) して 1 度に解読できるようになりました。これにより、1 サンプル当たりのコストも劇的に低下しています。MID タグをバーコードと呼ぶことも多いですが、DNA バーコーディングでは利用する塩基配列をバーコードと呼び、紛らわしいのでここでは単にタグに統一します。なお、インデックス配列と呼ぶ場合もあります。他の文書を読まれる場合は注意して下さい。

なお、後の解析では、PCR の際に生成されるキメラ配列や読み間違いの多い配列が多様性の把握の際に問題となりますが、PCR のレプリケートを確保して塩基配列決定を行うことで、全レプリケートで共通して見られる配列のみをキメラでなく読み間違いも少ない配列と考えることができます。これは、キメラのできる配列の組み合わせとキメラ配列の「継ぎ目」は多数あるが、キメラでない配列は非常に限られていてしかも PCR 前からあって観測される数がキメラより多くなりやすいため、真の配列は全てのレプリケートで検出されやすいが、キメラはそうではないこと、同様に読み間違いのパターンも無数にあるが真の配列はただ一通りで真の配列が全レプリケートで検出されやすいことを利用するものです。ソフトウェアによる処理だけではキメラや読み間違いを十分に除去できませんが、この方法を併用することで検出効率を改善できると期待できます。キメラと読み間違いの検出後、レプリケートは足し合わせて解析することができます。

1.1　タグ・アダプター配列付きプライマーを用いた PCR

タグを配列に付加するには、タグの付いたプライマーを用いて PCR を行うのが最も簡単で確実です。PCR 方式では初期投資としてタグ付きプライマーを購入する必要があり、タグの種類を多くしようとするとこれに最大数十万円を要します。次世代シーケンサー用のサンプル調製キットでは、予めサンプル内の DNA 末端に特定のアダプター配列の DNA が付加されていることを前提としています。そこで、以下のような配列のプライマーを用いて PCR を行います。なお、鋳型 DNA はメタゲノムでも単一のゲノムでもどちらでも構いません。

▌ 5’ー［アダプター配列］ー［タグ配列］ー［本来のプライマー配列］ー 3’

両側にタグを付けた場合は、増幅産物は以下のような形になります。

▌ 5’ー［アダプター配列 F］ー［タグ配列 F］ー［本来のプライマー配列 F］ー［解読したい配列］ー［本来のプライマー配列 R 逆相補］ー［タグ配列 R 逆相補］ー［アダプター配列 R 逆相補］ー 3’

片側から解読する場合、読み始めがタグ配列 F で、本来のプライマー配列 F、解読したい配列の順で読むことになります。

どのようなタグを使うかは、Hamady *et al*. (2008) の Supplement など、既存の文献を見て決めて下さい。ただし、タグが必要なのは解読開始側だけですので、片側からしか読まない場合には反対側はタグのないプライマーを使います。両側から読む場合でも、サンプル数が少ない場合は片側だけでいいのですが、解読ミスのためフォワード配列とリバース配列の対応がおかしくなることがあり、片側しかタグがないとそれが検出できませんので両側に付けることを推奨します。

このようなプライマーを用いて PCR を行うと、プライマーは鋳型 DNA に Y 字型にアニールし、アダプター配列とタグ配列が端に付加された増幅産物ができます。この増幅産物をできるだけ等濃度で混合してシーケンスサンプルを調製し、シーケンサーで塩基配列を解読します。シーケンスサンプル調整ステップ以降はメーカーから提供されている純正キットとそのプロトコルに従って下さい。メーカー提供のプロトコルにはいくつか改善すべき点があり、改善によってデータの品質が向上することもあるようです。詳しくはその筋の方にお問い合わせ下さい。濃度の測定は、Nanodrop などの分光光度計では十分な精度で行うことができません。そのため、最低でも ThermoFisher (旧 Invitrogen) の Qubit などで測定して下さい。シーケンスサンプルのアダプター配列をターゲットとして定量 PCR によって測定するタイプのキットが最も高精度ですが、これはランニングコストが非常に高いためサンプルが多い場合は適用できないと思います。

なお、プライマー配列もサンプルを区別する用途に使うことができますので、例えば植物の *rbcL* と *matK* などの複数領域配列を同時に各サンプルから得ることもできます。もちろんこれらは塩基配列データを見ることでも区別できます。PCR では、伸長時間を長めに取り、サイクル数を少なくするようにして下さい。シーケンスサンプル調製に必要な DNA 量は多くありませんので、サイクル数をそれほど多くする必要はないのです。サイクル数が多く伸長時間が短いほど、「途中まで合成された産物」が生成され、それが次のサイクルで別の鋳型に基いて伸長する、ということが多くなります。このようにして生成された DNA はキメラ DNA と呼ばれ、「存在しない新奇生物の発見」や「多様性の過大評価」に繋がります。Finnzymes 社の Phusion や東洋紡の KOD などの高正確性 DNA ポリメラーゼを用いるのも対策として有効です。キメラ DNA の生成を抑制するためには、熱変性後にアニーリング温度へと下げるときの速度を下げることがよいという報告があります (Stevens *et al.,* 2013) ので、もし温度の変化速度を設定可能なサーマルサイ

クラーであれば温度変化速度を下げるようにして下さい。後述するプログラムによる処理によって除去することもできますが、完全ではありませんので生成を抑えるのに越したことはありません。

増幅の難しい鋳型の場合、夾雑物があっても増えやすくする島津製作所の Ampdirect などの試薬を用いて PCR を行ったり、そもそも DNA を抽出する段階でホモジナイザーやビーズ破砕機を用いて細胞を破砕した上で DNA を抽出することで収量を増やすなどの対策が必要になります。破砕の前にディープフリーザーで凍結して、凍結した状態で破砕するのも効果的です。ポリフェノール類や多糖類を除去するキットを用いる必要がある場合もあるかもしれません。この辺りの課題は従来の方法と違いはありません。タグ・アダプターの付いていないプライマーでは増幅できるがタグ・アダプター付きのプライマーだとうまくいかないことがありますが、その場合はタグ・アダプターなしのプライマーで PCR した増幅産物を用意し、プライマー除去・精製して (ExoSAP-IT などでいいでしょう) から、タグ・アダプターの付いたプライマーで数サイクルの PCR を行い、増幅産物にタグ・アダプターを付加するというやり方でうまくいくことがあります。

1.1.1 中間アダプターを用いたコスト抑制方法

タグ・アダプター配列付きプライマーは非常に長いため、高価になりがちです。しかも、増幅するプライマーセットごとに用意しなくてはならず本数も多くなってしまいます。そこで、PCR を 2 回行うことで、1 組のタグ・アダプター付きプライマーを使い回す方法を紹介します。まず、1 回目は以下のプライマーセットを用いて PCR を行います。

- 5’ー［中間アダプター配列］ー［本来のプライマー配列］ー 3’

これで末端に中間アダプター配列の付いた増幅産物ができます。中間アダプター配列はシーケンスに必要なアダプター配列と異なるもので、非特異的増幅を起こさないものであれば何でも構いません。そういう配列は各種知られていますので、既に使用されているものを探してきて下さい。そして、この増幅産物を鋳型として以下のプライマーセットでさらに PCR を行います。

- 5’ー［本来のアダプター配列］ー［タグ配列］ー［中間アダプター配列］ー 3’

このような 2 段階 PCR によって、1 組のタグ・アダプター付きプライマーを使い回すことができるようになりますので、コスト削減になります。ただし、PCR を繰り返すため、PCR による人工的な置換や PCR バイアスが増加しますし、解読できる長さも減少します。その点には注意が必要です。なお、増幅産物は以下のような形になります。

> 5’ー［本来のアダプター配列 F］ー［タグ配列 F］ー［中間アダプター配列 F］ー［本来のプライマー配列 F］ー［解読したい配列］ー［本来のプライマー配列 R 逆相補］ー［中間アダプター配列 R 逆相補］ー［タグ配列 R 逆相補］ー［本来のアダプター配列 R 逆相補］ー 3’

Illumina の Nextera XT Index Kit を用いた multiplex 法 (Illumina corporation, 2013) はこれをキットで実現しているものです。この方法で 2 つのタグ配列と両側からの解読を行う場合、最初に中間アダプター配列 F の後ろから、つまり本来のプライマー配列 F から読み始め、解読したい配列方向へ解読します。次に、中間アダプター配列 R 逆相補の後ろからタグ配列 R 逆相補を解読します。3 番目にアダプター配列 F の後ろからタグ配列 F を解読し、方向を反転させて、最後に中間アダプター配列 R の後ろから本来のプライマー配列 R、および解読したい配列 (逆ストランド) を解読します。それぞれが R1、R2、R3、R4 を含むファイル名で保存されます。R1、R2、R3 は同じ strand で、R4 だけ逆 strand になります。R2 はタグ配列 R 逆相補で、R3 がタグ配列 F です。ややこしいので注意が必要です。なお、上記のキットでは R1 と R4 のシーケンスプライマーは中間アダプター配列をターゲットとするものになっているため、本来のプライマー配列が R1 と R4 のデータ配列に入ってしまい、解読したい領域が 500bp 程度ある場合には長さが不

足する可能性がありますが、本来のプライマーをシーケンスプライマーとして利用することでデータ配列にプライマー部分が入らないようすることも可能です。ただし、この方法では後述する N の挿入によるシーケンス品質の改善ができません。

1.1.2　N の挿入によるシーケンス品質の改善

Illumina では、フローセル上の蛍光を光学センサーで読み取っていますが、メタゲノムを鋳型としたユニバーサルプライマーでの PCR 増幅産物は配列が似ているため、フローセル上で DNA が近接していると区別が困難になります。また、読み始めに「ほとんどの配列が A であるため発光点が少なく真っ暗になる」と、異常と判断して解読を停止してしまうことがあるとのことです。ゲノムショットガンや RNA-seq では近接している DNA が非常によく似ている可能性がずっと低いために区別できる上、真っ暗になることもないのですが、同じ遺伝子座の増幅産物 (しかも読み始めがほとんど変異のないプライマー領域) だとまずいわけです。そこで、本来のプライマー配列と中間アダプター配列の間にわざと `NNNNNN` を入れてしまうことにします。そうすると、この部分の解読時に近接した DNA も区別することができ、真っ暗になることもないため、シーケンス品質が改善するとのことです (Nelson *et al.*, 2014)。MiSeq や HiSeq をお使いの方はお試し下さい。さらに、この長さをサンプルごとに変えてしまうと、解読するゲノム上の位置がずれるため、全く同じ配列が隣接していても区別できるようになるためにシーケンス品質が改善するそうです (Fadrosh *et al.*, 2014)。これらの対処法により、PhiX 標準配列を混ぜる量を減らすことができるので、一度に得られるデータ量も増加するはずです。

第 2 章

塩基配列データのクラスタリング

2.1　塩基配列データの前処理

GS シリーズシーケンサーや Ion PGM では、最終出力として`*.sff` というファイルが得られます。Illumina MiSeq の場合は`*.fastq` となります。ここでは、これらのファイルに含まれる配列をサンプルごとに分別 (demultiplex) して別々のファイルに分けた上で、低品質の配列の除去を行う方法を説明します。どの機種でもメーカーのプログラムの機能による demultiplex は品質の値を見ていなかったりするため、Claident のコマンド `clsplitseq` を利用して demultiplex を行います。以下ではコマンドは全てターミナルかコンソールで実行して下さい。基本的なターミナルの使い方の知識は持っているという前提で話を進めます。ターミナルの操作に慣れていない方は、最低でも付録 B の内容を理解できるようになっておいて下さい。

2.1.1　SRA/DRA/ERA の登録データや demultiplex 済 FASTQ の変換

Claident では、配列の名前として`配列 ID__ラン ID__タグ ID__プライマー ID` を、また拡張子を除いた入力ファイル名として`ラン ID__タグ ID__プライマー ID` を仮定しています。そのため、SRA/DRA/ERA の登録データや demultiplex 済 FASTQ をそのまま用いることができません。そこで、`climportfastq` というコマンドを用いることで、上記の仮定を満たす入力ファイルに変換することができます。ペアエンドの場合は変換する前に第 2.1.3 節の内容に従ってフィルタリングと連結を先に行っておきます。変換するには、以下のような入力ファイルを用意します。

```
| 配列ファイル名 1 ラン ID__タグ ID__プライマー ID
| 配列ファイル名 2 ラン ID__タグ ID__プライマー ID
| 配列ファイル名 3 ラン ID__タグ ID__プライマー ID
```

ラン ID は適当にでっち上げていただいて構いません。プライマー ID は同じプライマーを用いたファイル間で同じになるようにすれば何でも構いません。タグ ID はファイルごとに異なるものにします。配列ファイル名と同じでも構いません。

上記の入力ファイルが用意できたら、以下のように `climportfastq` を実行することで、出力フォルダに変換後のファイルが作成されます。

```
> climportfastq \
--numthreads=CPU 数 \
入力ファイル \
出力フォルダ↓
```

変換後、シングルエンドデータであれば第 2.1.2 節に従って低品質配列の除去を行って下さい。

2.1.2　GS シリーズシーケンサーや Ion PGM の場合

SFF から FASTQ の生成

まずは SFF ファイルから FASTQ 形式のファイルを作成します。

```
> sff_extract -c SFF ファイル名↓
```

-c オプションを付けることで、読み始めの TCAG が除去されてファイルが作成されます。もし後述するタグ配列の冒頭に TCAG を付けている場合には、このオプションは使用しないで下さい。SFF ファイルが HOGEHOGE.sff だとすると、HOGEHOGE.fastq という FASTQ 形式のファイルとして塩基配列は保存されます。HOGEHOGE.xml というファイルもできますが、ここでは使いませんので削除して構いません。出力される塩基配列はタグ配列で始まり、解読開始側プライマー配列、解読対象の塩基配列データと続き、反対側プライマー配列 (の逆相補配列) で終わります。ただし、全域が読めるとは限りませんので、途中で読み終わっている配列も含んでいます。なお、ここでは sff_extract コマンドを用いていますが、読み始めの TCAG を除いた塩基配列の FASTQ 形式ファイルが得られれば、シーケンサーメーカーから提供される純正のコマンドを使っていただいても構いません。逆に後述する clsplitseq に与えるタグ配列の冒頭に TCAG を追加していただいても構いませんが、TCAG 部分の品質値まで厳格にチェックされます。

塩基配列をサンプルごとに分ける

次に、タグ配列とプライマー配列に基づいてサンプルごとに配列を別々のファイルに分離 (demultiplex) します。事前に以下のようなタグ配列を記した FASTA ファイル、解読開始側プライマー配列を記した FASTA ファイルがそれぞれ必要です。

```
| >タグ ID
| タグ配列
| >examplesample1
| ACGTACGT
```

```
| >プライマー ID
| プライマー配列
| >exampleprimer1
| ACGTACGTACGTACGTACGT
```

タグ配列には縮重コード表記は使えませんが、プライマー配列には縮重コード表記を利用可能です。タグ配列、プライ

マー配列ともに複数の配列を記述しておくことも可能です。なお、中間アダプターを用いたりした場合は、プライマー配列ファイルは以下のような内容で作成して下さい。

```
| >プライマー ID
| 中間アダプター配列プライマー配列
```

つまり、プライマー配列ファイルには、タグ配列と解読対象の配列の間にある配列を記述して下さい。

これらのファイルが用意できたら、以下のコマンドで塩基配列をサンプルごとに別ファイルへ分割します。

```
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル \
--primerfile=プライマー配列ファイル \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
入力ファイル \
出力フォルダ↓
```

ラン ID はランごとに異なるように適当に決めて指定して下さい。通常はシーケンサーのメーカーが適当に付けているはずなので、それを記述します。ファイル名やファイル中の配列名などに含まれていますが、メーカーによってルールが異なりプログラムから自動的に取得することが面倒なため、ユーザー側で指定する必要があります。`--minqualtag` はタグ配列部分の品質値の下限を指定するオプションです。この値より品質値が低い塩基がタグ配列に含まれる場合は、その配列は出力されません。品質値 27 という値は、Roche GS シリーズシーケンサーの読み取りミスをクラスタリング・コンセンサス配列作成によって除去するのに適した値です (Kunin *et al.*, 2010)。他機種では別の値の方が良い可能性があります。品質値 30 がかなり広く使用されていますので、30 にしてもいいでしょう。

タグを利用したマルチプレックスシーケンスを行っていない場合、`--tagfile` オプションが不要となりますが、単に省略しただけだとこの後の Claident のコマンドで処理可能な配列ファイルが作成されません。そのため、ダミーでも構いませんので`--indexname=タグ ID` というオプションを付加して実行して下さい。

出力される配列からは、タグとプライマーの配列部分は除去されています。タグ配列部分の認識は厳密一致です。1 塩基の読み間違いを許すなどのオプションはありません。プライマー配列の認識は、Needleman-Wunsch アルゴリズムを用いたアライメントを行い、14% まで不一致を許容します (閾値は変更可能)。出力フォルダには、`ラン ID__タグ ID__プライマー ID.fastq.gz` という名前の塩基配列ファイルが保存されます。`clsplitseq` では並列処理によって多数の CPU を活かした高速化が可能です。CPU が 4 個の場合、`--numthreads` オプションに 4 を指定すれば最も高速に処理ができるはずです。ただし、OS やディスクへの書き込み速度がボトルネックになって思うように速度が出ないこともあります。なお、出力ファイルは GZIP で圧縮されていますので、GZIP 圧縮 FASTQ に対応していないプログラムで読み込むには `gzip -d` などで圧縮を解除しておく必要があります。これ以降の Claident のコマンドは対応しているためそのまま使えます。

DDBJ Sequence Read Archive (DRA) など、次世代シーケンサーの生データを登録するサイトには、ここで生成された GZIP 圧縮 FASTQ を登録します。

■数ラン分に分けて大量サンプルをシーケンスした場合　`clsplitseq` による処理を複数回行います。ただし、

clsplitseq は出力先フォルダが既に存在しているとエラーになり処理できません。そこで、2 回目以降は--append オプションを付けて実行します。例えば以下のようにして下さい。

```
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル \
--primerfile=プライマー配列ファイル \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
出力フォルダ↓
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル \
--primerfile=プライマー配列ファイル \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
--append \
入力ファイル 2 \
出力フォルダ↓
```

■異なる長さのタグが混在している場合について　clsplitseq では、タグの長さは全て同一であることを仮定しています。その仮定を使うことで処理を高速化しています。そのため、タグの長さが不統一の場合は、長さの異なるタグごとに別々の FASTA ファイルを準備し、以下のようにして複数回 clcplitseq による処理を行って下さい。

```
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル 1 \
--primerfile=プライマー配列ファイル \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
入力ファイル \
出力フォルダ↓
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル 2 \
--primerfile=プライマー配列ファイル \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
--append \
入力ファイル \
出力フォルダ↓
```

■リバースプライマー配列の認識と除去　これまでの手順では、リバースプライマー配列がアニールする部分の配列は除去されていません。これもできれば除去した方が良いでしょう。それには、まず以下のような内容のリバースプライマー配列のファイルを用意します。

```
| >プライマー ID
| プライマー配列
| >exampleprimer1
| TCAGTCAGTCAGTCAGTCAG
```

配列は複数記述できますが、フォワードプライマーと 1 対 1 対応していることを仮定していますので注意して下さい。フォワードプライマーと配列数が異なるとエラーになります。配列順序も対応していることを仮定していますので、リバースプライマーは同一でもフォワードプライマーが異なるサンプルがある場合、それぞれ異なるプライマー ID を割り当てて両方記述する必要があります。

ファイルが用意できたら、clsplitseq を以下のように実行して下さい。

```
> clsplitseq \
--runname=ラン ID \
--tagfile=タグ配列ファイル \
--primerfile=プライマー配列ファイル \
--reverseprimerfile=リバースプライマー配列ファイル \
--reversecomplement \
--minqualtag=27 \
--numthreads=使用する CPU 数 \
入力ファイル \
出力フォルダ↓
```

このコマンドでは前述のコマンドの処理に加えて、さらにリバースプライマー逆相補配列を Needleman-Wunsch アルゴリズムを用いたアライメントによって 15% まで不一致を許して探します。該当する配列が見つかったら、該当箇所の先頭以降を元の配列から除去します。該当する配列が見つからなかった場合は、他の条件を満たしていればそのまま出力ファイルに保存されます。ただし、読み間違いなどによってリバースプライマーの付く部位以降が除去しきれないこともあります。なお、--needreverseprimer というオプションを追加することで、リバースプライマー逆相補配列に一致する部位が見つからなかった配列は出力されなくなります。最後まで解読しきれた配列だけにしたい場合はこのオプションを付けて実行してみて下さい。

低品質配列の除去

FASTQ 形式の塩基配列には、解読時のデータから推定された塩基の品質値がありますので、その情報に基づいて質の低い配列をこの時点である程度除去してしまうことができます。以下のように clfilterseq を実行して下さい。

```
> clfilterseq \
--minqual=27 \
--minquallen=3 \
--minlen=350 \
--maxlen=400 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
入力ファイル \
出力ファイル↓
```

ここで、--minqual は品質値の下限値で、終末端側からこの値以上の品質値が--minquallen の値以上に連続して出現するまで配列を削っていきます。その上で、--minlen の長さ未満の配列は除去し、--maxlen より長い場合はこの値になるまで終末端を削ります。残った配列において、--minqual 未満の配列が--maxplowqual の割合より多い場合はその配列は除去します。通常は出力はファイルになりますが、--output=folder というオプションを与えた場合はフォルダを作成して入力ファイルと同じ名前のファイルに出力します。既存のフォルダに出力する場合は--append を

追加して下さい。

clsplitseq の出力フォルダ内の全配列ファイルに対して同じ処理を行う場合は、以下のようなコマンドを実行して下さい。

```
> for f in clsplitseq の出力フォルダ/*.fastq.gz ↓
do clfilterseq \
--output=folder \
--append \
--minqual=27 \
--minquallen=3 \
--minlen=350 \
--maxlen=400 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
$f \
出力フォルダ↓
done ↓
```

2.1.3　Illumina 社シーケンサーの場合

BCL から FASTQ の生成

Illumina 社のシーケンサーでは、標準で demultiplex する機能がありますが、タグ配列部分の品質値を一切見ていないため、タグ配列の品質が低い配列がファイルに含まれています。このため、生データである BCL ファイルから、Illumina が配布している bcl2fastq (1.x 系と 2.x 系はどちらでも構いません) を用いて、demultiplex していない FASTQ 配列を生成し、clsplitseq で demultiplex を行います。bcl2fastq のインストールは付録に書きましたのでそちらを参照願います。また、Illumina 社のシーケンサーでは 300bp のシーケンスでは 301bp (150bp のシーケンスでは 151bp) のデータが出力されるため、末尾の 1 塩基は除去する必要があります (何故最初から 1 塩基除いたデータを出力してくれないのかは知りません)。下記の例では bcl2fastq で FASTQ 配列生成の際に除去するようになっています。

まず、シーケンサー付属マシンから BaseCalls フォルダを含むランデータフォルダをまるごと bcl2fastq をインストールした PC へコピーします。なお、BaseCalls フォルダ内またはランデータフォルダ内に SampleSheet.csv が存在する場合は、これをリネームしておく必要があります。

次に、bcl2fastq の 1.x 系では以下のようにコマンドを実行します。

```
> cd BaseCalls フォルダの直上フォルダ↓
> configureBclToFastq.pl \
--fastq-cluster-count 0 \
--use-bases-mask Y300n,Y8,Y8,Y300n \
--input-dir BaseCalls フォルダ \
--output-dir 出力フォルダ↓
> make -j4 ↓
```

--fastq-cluster-count 0 でファイルの自動分割機能を無効にし、--use-bases-mask Y300n,Y8,Y8,Y300n で

フォワード側配列を 300bp (末端 1bp は切り捨て)、インデックス 1 (タグ配列 R 逆相補) を 8bp、インデックス 2 (タグ配列 F) を 8bp、リバース側配列を 300bp (末端 1bp は切り捨て) とし、それぞれ*_R1_001.fastq.gz、*_R2_001.fastq.gz、*_R3_001.fastq.gz、*_R4_001.fastq.gz として出力するよう設定しています。これは Dual Index (インデックスは各 8bp) で両側から各 300bp ずつ解読する (PE300) 設定でシーケンスした場合ですので、異なる設定でシーケンスした場合は値を適宜変更して下さい。make -j4 を実行すると、4 つの CPU を使用して BCL から FASTQ の生成が行われます。生成された FASTQ は GZIP というプログラムで圧縮されたものになります。ファイル名の末尾の拡張子.gz はそれを示すものです。Claident は GZIP で圧縮された FASTQ をそのまま扱えるため、圧縮したままで構いません。Single Index の場合や、片側からしか解読していない場合は--use-bases-mask オプションを適宜変更する必要があります。Single Index (長さ 6bp) で片側 250bp の解読を行った場合は--use-bases-mask Y250n,Y6 となり、Dual Index (長さ各 8bp) で片側 300bp の場合は--use-bases-mask Y300n,Y8,Y8 となります。

bcl2fastq の 2.x 系を使用している場合は以下のようにコマンドを実行します。

```
> bcl2fastq \
--processing-threads 4 \
--use-bases-mask Y300n,Y8,Y8,Y300n \
--runfolder-dir ランデータフォルダ \
--output-dir 出力フォルダ↓
```

--processing-threads は処理に使用する CPU 数を指定するオプションです。CPU が 16 個ある PC で専有できるなら 16 を指定して下さい。--use-bases-mask オプションの意味は 1.x 系と同じです。--runfolder-dir にはランデータフォルダを指定します。1.x では--input-dir に BaseCalls フォルダを指定していましたが、2.x では指定方法が異なりますので注意して下さい。

塩基配列をサンプルごとに分ける

FASTQ ファイルができたら、タグ配列とプライマー配列に基づいてサンプルごとに配列を別々のファイルに分離 (demultiplex) します。事前に以下のようなタグ配列を記した FASTA ファイル、解読開始側プライマー配列を記した FASTA ファイルがそれぞれ必要です。ペアエンドの場合は、当然リバース側の配列を記したファイルも必要です。

```
| >タグ ID
| タグ配列
| >examplesample1
| ACGTACGT
```

```
| >プライマー ID
| プライマー配列
| >exampleprimer1
| ACGTACGTACGTACGTACGT
```

タグ配列には縮重コード表記は使えませんが、プライマー配列には縮重コード表記を利用可能です。タグ配列、プライマー配列ともに複数の配列を記述しておくことも可能ですが、フォワードプライマーとリバースプライマーは配列数も配列順序も対応していることを仮定しています。リバースプライマーは同一でもフォワードプライマーが異なるサンプルがある場合、それぞれ異なるプライマー ID を割り当てて両方記述する必要があります。タグに関しても同様です。

なお、プライマーの前に NNNNNN を付加した場合は、それも含めてプライマーを記述して下さい。サンプルごとに N の長さを変化させている場合は最も長い場合だけを記述して下さい。

これらのファイルが用意できたら、以下のコマンドで塩基配列をサンプルごとに別ファイルへ分割します。

```
> clsplitseq \
--runname=ラン ID \
--index1file=インデックス 1 配列 (タグ配列 R 逆相補) ファイル \
--index2file=インデックス 2 配列 (タグ配列 F) ファイル \
--primerfile=プライマー配列ファイル \
--reverseprimerfile=リバースプライマー配列ファイル \
--minqualtag=30 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
入力ファイル 2 \
入力ファイル 3 \
入力ファイル 4 \
出力フォルダ↓
```

入力ファイルは*_R1_001.fastq.gz、*_R2_001.fastq.gz、*_R3_001.fastq.gz、*_R4_001.fastq.gz の順で記述して下さい。--index1file は Illumina が言うところのインデックス 1、つまりタグ配列 R 逆相補を指定し、--index2file には Illumina が言うところのインデックス 2、つまりタグ配列 F を指定して下さい。逆にしないように注意して下さい。フォワードプライマーは 14% まで、リバースプライマーは 15% まで不一致を許容して探します。なお、プライマーの前に NNNNNN を付加した場合は、--truncateN=enable を付加して下さい。このオプションにより、プライマー配列先頭の N とデータ配列の対応する部位は削った上で不一致度を算出するようになります。N の長さを変化させていても、その部位を除いて不一致度が算出されるようになるためプライマー配列ファイルには N が最長の場合だけを記述しておけば全てヒットするようになります。処理が終わったら、各ファイルの配列数を確認し、Illumina 純正のプログラムで demultiplex したファイルの配列数と比較して下さい。正しく処理できていれば、全て Illumina 純正プログラムで処理した場合より少なくなっているはずです。

本来のプライマー配列をシーケンスプライマーとして用いた場合、データ配列にはプライマー配列に当たる部位が含まれていません。この場合は--primerfile と--reverseprimerfile オプションが必要ないわけですが、配列の名前をこの後の Claident のコマンドで処理可能な形式とするためにオプションとして--primername=プライマー ID を付加して実行して下さい。ダミーでも構いません。

タグを利用したマルチプレックスシーケンスを行っていない場合、--index1file と--index2file オプションが不要となりますが、単に省略しただけだとこの後の Claident のコマンドで処理可能な配列ファイルが作成されません。そのため、ダミーでも構いませんので--indexname=タグ ID というオプションを付加して実行して下さい。

出力されるファイルはラン ID__タグ ID__プライマー ID.forward.fastq.gz とラン ID__タグ ID__プライマー ID.reverse.fastq.gz になります。DDBJ Sequence Read Archive (DRA) など、次世代シーケンサーの生データを登録するサイトには、ここで生成された GZIP 圧縮 FASTQ を登録します。なお、DRA への登録では配列の長さが揃っているかどうかを指定する箇所がありますが、長さが可変のプライマー部位の配列を除去していますから揃っていません (プライマーセットが 1 つであってもアライメント時にギャップが入る可能性があるため揃わない) ので誤って揃っていると指定しないようにして下さい。

フォワード配列とリバース配列の連結

ペアエンドでシーケンスを行った場合には、clconcatpair を用いてフォワード配列とリバース配列を連結します。フォワード配列とリバース配列にオーバーラップがある場合には、以下のように実行することで VSEARCH を呼び出して連結します。

```
> clconcatpair \
--mode=OVL \
--numthreads=使用する CPU 数 \
入力フォルダ \
出力フォルダ↓
```

以上のコマンドでは、入力フォルダから*.forward.fastq と*.reverse.fastq を探しだして自動的に連結します。末尾に.gz や.bz2 が付く圧縮ファイルでも構いません。出力ファイルは指定したフォルダ内に*.fastq.gz として作成されます。

入力ファイル名が*.forward.fastq と*.reverse.fastq になっていない場合は、以下のように実行することで 1 組のファイルの配列を連結可能です。

```
> clconcatpair \
--mode=OVL \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
入力ファイル 2 \
出力ファイル↓
```

入力ファイル 1 にフォワード配列の FASTQ ファイル、入力ファイル 2 にリバース配列の FASTQ ファイルを指定して下さい。出力ファイルは末尾に.gz や.bz2 を付けないと自動的に圧縮はされません。

また、フォワード配列とリバース配列間にオーバーラップがない場合や、あるものとないものが入り混じっている場合には、先に以下のように clfilterseq を用いて低品質部位のトリミングや低品質配列の除去を行います。

```
> clfilterseq \
--minqual=30 \
--minquallen=3 \
--minlen=100 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
入力ファイル 2 \
出力フォルダ↓
```

ここで、--minqual は品質値の下限値で、終末端側からこの値以上の品質値が--minquallen の値以上に連続して出現するまで配列を削っていきます。その上で、--minlen の長さ未満の配列は除去します。残った配列において、--minqual 未満の配列が--maxplowqual の割合より多い場合はその配列は除去します。この処理では、ペアエンドの

どちらか一方でも除去される場合、ペアとなるもう一方も除去します。出力はフォルダを作成して入力ファイルと同じ名前のファイルになります。既存のフォルダに出力する場合は--append を追加して下さい。フォルダ内の全ての*.forward.fastq と*.reverse.fastq に適用する場合は、以下のように実行して下さい。

```
> for f in ‘ls *.forward.fastq.gz | grep -P -o ’^[^\.]+’‘↓
do clfilterseq \
--minqual=30 \
--minquallen=3 \
--minlen=100 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
$f.forward.fastq.gz \
$f.reverse.fastq.gz \
出力フォルダ↓
done ↓
```

このように低品質配列を除去した後、以下のように clconcatpair を用いて連結を行います。

```
> clconcatpair \
--mode=NON \
--numthreads=使用する CPU 数 \
入力フォルダ \
出力フォルダ↓
```

この方法では、以下のようなフォワード配列とリバース配列を入力として仮定します。

```
5’ ー フォワード配列 ー 3’
5’ ー リバース配列 ー 3’
```

これらの配列を、clconcatpair --mode=NON では以下のように連結します。

```
3’ ー リバース配列 (逆相補) ー 5’ ー ACGTACGTTGCATGCA ー 5’ ー フォワード配列 ー 3’
```

ノイジー配列とキメラ配列の除去はそのまま通常の連結したペアエンドデータと全く同様に行い、配列のクラスタリングに用いる clclassseqv と代表配列への生配列のマッピングを行う clrecoverseqv の実行時に--paddinglen=16 を加えて実行します。上記の連結では、連結点を後から識別できるように人工配列である ACGTACGTTGCATGCA を加えているため、配列の類似度が過大になってしまいます。そこで、完全一致するはずの ACGTACGTTGCATGCA の長さ分を配列の類似度計算から除外することで正しい配列類似度に基づくクラスタリングを行います。

ホスト生物の同定をする際には、ACGTACGTTGCATGCA を探して配列を分割し、フォワード配列とリバース配列で別々に同定し、第 3.3 節の内容に従って 2 つの同定結果を統合します。一般にフォワード配列の方が品質が高いはずですので、何も事前情報がない場合はフォワード配列に基づく同定結果を優先する設定で統合するのが良いでしょう。配列の分割は cldivseq を以下のように実行することで行えます。

```
>cldivseq \
--query=ACGTACGTTGCATGCA \
入力ファイル \
出力ファイル 1 \
```

```
出力ファイル 2
```

出力ファイル 1 がリバース配列の逆相補配列、出力ファイル 2 がフォワード配列になります。

フォワード配列とリバース配列にオーバーラップがある場合には、PEAR (Zhang *et al.,* 2014) または VSEARCH を用いて連結することもできます。PEAR の場合は demultiplex した配列ファイルのあるフォルダで、以下のコマンドを実行して下さい。

```
> pear \
-p 0.0001 \
-u 0 \
-j 使用する CPU 数 \
-f ラン ID__タグ ID__プライマー ID.forward.fastq.gz \
-r ラン ID__タグ ID__プライマー ID.reverse.fastq.gz \
-o ラン ID__タグ ID__プライマー ID ↓
```

正常に処理が終わると、以下のファイルができます。

ラン ID__タグ ID__プライマー ID.assembled.fastq　連結された配列
ラン ID__タグ ID__プライマー ID.unassembled.forward.fastq　連結されなかったフォワード配列
ラン ID__タグ ID__プライマー ID.unassembled.reverse.fastq　連結されなかったリバース配列
ラン ID__タグ ID__プライマー ID.discarded.fastq　検定で一致していると判定されなかった配列

この後で使用するのはラン ID__タグ ID__プライマー ID.assembled.fastq だけです。また、出力は圧縮されていないため、ディスク容量の節約のためにも圧縮しておいた方がいいかもしれません。

なお、フォルダ内の全てのファイルに対して PEAR による連結を連続して行うには、以下のようなコマンドをお使い下さい。

```
> for f in ‘ls *.forward.fastq.gz | grep -P -o ’^[^\.]+’‘↓
do pear \
-p 0.0001 \
-u 0 \
-j 使用する CPU 数 \
-f $f.forward.fastq.gz \
-r $f.reverse.fastq.gz \
-o $f ↓
done ↓
```

直接 VSEARCH を用いて連結する場合には以下のようにコマンドを実行して下さい。

```
> vsearch \
--threads 使用する CPU 数 \
--fastq_mergepairs ラン ID__タグ ID__プライマー ID.forward.fastq.gz \
--reverse ラン ID__タグ ID__プライマー ID.reverse.fastq.gz \
--fastq_allowmergestagger \
--fastqout ラン ID__タグ ID__プライマー ID.assembled.fastq ↓
```

増幅産物長が解読長よりも短い場合、フォワード配列の読み終わりがリバース配列の読み始めを超えてしまったりリバース配列の読み終わりがフォワード配列の読み始めを超えてしまいますが、デフォルトではそのような配列が連結されません。--fastq_allowmergestagger はこれを許して連結するオプションです。反対側の読み始めを超えてしまったオーバーハング部分は人工配列なので切り捨てられます。上記のようなケースの可能性がないライブラリを解読した場合は指定する必要はありません。PEAR では--fastq_allowmergestagger を指定した場合と同じ処理がなされます。PEAR と同様に連結できなかった配列を得たい場合は--fastqout_notmerged_fwd オプションと--fastqout_notmerged_rev オプションに保存先のファイル名を指定して下さい。そのほか、最小オーバーラップ長を--fastq_minovlen で、連結後の最小・最大配列長を--fastq_minmergelen・--fastq_maxmergelen で、オーバーラップ中の最大許容不一致数を--fastq_maxdiffs で、連結後の最大許容期待エラー数を--fastq_maxee で指定できます。

フォルダ内の全てのファイルに対して VSEARCH による連結を連続して行うには、以下のようなコマンドをお使い下さい。

```
> for f in ‘ls *.forward.fastq.gz | grep -P -o ’^[^\.]+’‘↓
do vsearch \
--threads 使用する CPU 数 \
--fastq_mergepairs $f.forward.fastq.gz \
--reverse $f.reverse.fastq.gz \
--fastq_allowmergestagger \
--fastqout $f.assembled.fastq ↓
done ↓
```

低品質配列の除去

FASTQ 形式の塩基配列には、解読時のデータから推定された塩基の品質値がありますので、その情報に基づいて質の低い配列をこの時点である程度除去してしまうことができます。以下のように clfilterseq を実行して下さい。

```
> clfilterseq \
--minqual=30 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
入力ファイル \
出力ファイル↓
```

このコマンドでは、入力配列において--minqual 未満の配列が--maxplowqual の割合より多い場合はその配列を除去します。通常は出力はファイルになりますが、--output=folder というオプションを与えた場合はフォルダを作成して入力ファイルと同じ名前のファイルに出力します。既存のフォルダに出力する場合は--append を追加して下さい。Illumina でペアエンドを連結した場合は、両端は品質値が高く、重複部分も一致していれば品質値が高くなりますので、末端から品質値の低い配列を削るのはあまり意味がありません。連結したペアエンドではこのように品質値が低い配列が一定量を超えた配列を除去するのがよいでしょう。もちろん、既存の FASTQ の品質確認を行うプログラムを利用されてもいいと思います。そのようなプログラムとしては、FastQC (Andrews, 2010) や PRINSEQ (Schmieder and Edwards, 2011) があります。

PEAR で連結した全配列ファイルに対して同じ処理を行う場合は、以下のようなコマンドを実行して下さい。

```
> for f in clsplitseq の出力フォルダ/*.assembled.fastq ↓
do clfilterseq \
--output=folder \
--append \
--minqual=30 \
--maxplowqual=0.1 \
--numthreads=使用する CPU 数 \
$f \
出力フォルダ↓
done ↓
```

VSEARCH の機能を用いて、品質値に基づいた配列全体での期待エラー数に上限を設定したフィルタリングも可能です。これは以下のようなコマンドで適用できます。

```
> vsearch \
--threads 使用する CPU 数 \
--fastq_filter 入力ファイル \
--fastq_maxee 1.0 \
--fastqout 出力ファイル↓
```

片側だけのシーケンスデータや、連結していないデータの場合は、Roche GS シリーズシーケンサや Ion PGM の場合と同様の処理を行えばいいと思います。第 2.1.2 節をご参照下さい。ただし、品質値や長さの閾値は適宜変更する必要があると思います。

2.1.4 同じ鋳型を複数回 PCR または複数回シーケンスした場合について

キメラ配列は PCR によって生成されますので、同一 PCR 由来配列なのか、異なる PCR 由来配列なのかをプログラムに正しく認識させなくてはなりません。そのため、同じ鋳型を複数回 PCR または複数回シーケンスした場合には、プログラムがデータを正しく扱えるようにいくつかの操作が必要になります。なお、ここでは、Nested PCR や、増幅のための PCR とタグ配列付加のための PCR はまとめて 1 回と数えることに注意して下さい。つまり、ここで「複数回 PCR」とは、「増幅産物を鋳型にしてさらに PCR で増幅する」ことではなく、「同じ鋳型を使用した PCR」を複数回行うことです。

同じ鋳型を複数回 PCR して同じタグ配列を付加して同じランでシーケンスした場合

実験デザインが間違っており、複数回 PCR を活かしたキメラ検出・除去は行えません。通常の 1 回の PCR と同様にしか扱えません。

同じ鋳型を複数回 PCR して別のタグ配列を付加して同じランでシーケンスした場合

Claident ではサンプル ID=ラン ID__タグ ID__プライマー ID ですから、同じ鋳型由来のサンプルが複数存在するデータとなります。わざと別サンプルのままにしておいて、1 つ以上のサンプルで出現しない OTU はキメラや読み間違い (PCR の合成ミスも含む) のある配列と判断するというのも良い考えだと思います。ただ、キメラの生成や読み間違い

がランダムであればこれはベストな方法でしょうが、できやすいキメラや起きやすい読み間違いというものがもしあれば (あるはずです)、同一のキメラ配列・読み間違いのある配列が全サンプルに出現する可能性があります。この方法がどの程度正確かという情報はまだ十分ではありませんので、プログラムによるキメラ検出も併用した方が良いかもしれません。この方法の効果を調べた研究 (Lange *et al.*, 2015) も参照して下さい。この方法は Claident でサポートしているため実施方法は後述します。サンプルごとの各 OTU の配列数の集計表をこの後で作成しますが、集計表の加工コマンド `clfiltersum` により複数のサンプルを統合して 1 サンプルにすることが可能です (各 OTU の配列数は和になります)。

同じ鋳型を 1 回 PCR して同じタグ配列を付加して複数のランでシーケンスした場合

1 回のランでは、たまたま得られる配列が少ないことがあります。そのため同一サンプルを複数のランでシーケンスし、全ランのデータを使いたいことがあります。Claident ではサンプル ID=ラン ID__タグ ID__プライマー ID ですから、同じ鋳型由来のサンプルが複数存在するデータとなります。そのような場合はそのまま別サンプルとして解析を進めることもできますし、1 サンプルにまとめてから解析を進めることもできます。別サンプルのまま処理を進める方を推奨します。そして、サンプルごとの各 OTU の配列数の集計表を作成する際に複数のサンプルを統合して 1 サンプルにまとめて下さい (各 OTU の配列数は和になります)。後者の方法では旧パイプラインによる解析でサンプルごとにキメラ検出を行うことが可能ですが、それ以外特にメリットはありません。

同じ鋳型由来の複数サンプルを別サンプルのまま解析を進めた場合、サンプルごとの各 OTU の配列数の集計表を作成後に集計表の加工コマンド `clfiltersum` により複数のサンプルを統合して 1 サンプルにまとめて下さい (各 OTU の配列数は和になります)。

同じ鋳型由来の複数サンプルを 1 つのサンプルに統合してからこの後の解析に進む場合は、以下のように処理して下さい。

```
> clsplitseq \
オプション省略 \
--runname=HOGEHOGE \
入力ファイル 1 \
出力フォルダ↓
> clsplitseq \
オプション省略 \
--runname=HOGEHOGE \
--append \
入力ファイル 2 \
出力フォルダ↓
```

このように`--runname` オプションでラン ID を置換し、2 つの入力ファイルのラン名を同一にしてやることで、2 つのファイルからの同一サンプルからの配列が `HOGEHOGE__タグ ID__プライマー ID.fastq.gz` という 1 つのファイルに書き出されます。

Claident ではラン ID とタグ ID とプライマー ID が同じなら、同一 PCR 産物=同一サンプルの配列として扱います。PCR が 1 回でタグ配列ファイルとプライマー配列ファイルが同じなら、当然タグ ID とプライマー ID は同じになっているはずですので、ラン ID も同じにしてしまうことで、同一 PCR 産物の配列として認識させることができるようになります。

同じ鋳型を複数回 PCR して同じタグ配列を付加して別々のランでシーケンスした場合

複数ラン分の配列ファイルがあるはずですので、clsplitseq によるサンプルごとの配列の分別と clfilterseq による低品質配列の除去を別々に実行し、別々のフォルダに出力します。この後のノイジー配列とキメラ配列の除去も別々に行い、クラスタリングの際に複数ラン分のデータをまとめて入力ファイルとして与えて下さい。そして、サンプルごとの各 OTU の配列数の集計表を作成する際に複数のサンプルを統合して 1 サンプルにまとめて下さい (各 OTU の配列数は和になります)。

同じ鋳型を複数回 PCR して異なるタグ配列を付加して別々のランでシーケンスした場合

1 回のランでは、たまたま得られる配列が少ないことがあります。そのため同一サンプルを複数のランでシーケンスし、全ランのデータを使いたいことがあります。Claident ではサンプル ID=ラン ID__タグ ID__プライマー ID ですから、同じ鋳型由来のサンプルが複数存在するデータとなります。そのような場合はそのまま別サンプルとして解析を進めることもできますし、1 サンプルにまとめてから解析を進めることもできます。別サンプルのまま処理を進め、ノイジー配列とキメラ配列の除去の際に「同じ鋳型由来の複数の PCR 産物に共通に出現しない塩基配列はキメラもしくはノイジー配列とみなす」方法を適用することを推奨します。そして、サンプルごとの各 OTU の配列数の集計表を作成する際に複数のサンプルを統合して 1 サンプルにまとめて下さい (各 OTU の配列数は和になります)。

clsplitseq は、複数の配列ファイルがあるはずですので以下のように実行して下さい。ここではファイルが 2 つの場合を示します。

```
> clsplitseq \
オプション省略 \
--tagfile=タグ配列ファイル 1 \
--primerfile=プライマー配列ファイル \
入力ファイル 1 \
出力フォルダ↓
> clsplitseq \
オプション省略 \
--tagfile=タグ配列ファイル 2 \
--primerfile=プライマー配列ファイル \
--append \
入力ファイル 2 \
出力フォルダ↓
```

プライマー配列ファイル、出力フォルダは同じものを与えて下さい。--runname オプションでは異なるラン ID を指定します。また、タグ配列ファイルは個別に用意して与えて下さい。ただし、同じ鋳型由来の配列に付加したタグ ID は同じにして下さい。配列が同じタグであっても、異なる鋳型由来の配列に付加したタグの ID を同一にしてはいけません。例えば以下の 2 ファイルを用意したとします。

```
| >sample1
| ACGTACGT
| >sample2
| ATGCATGC
```

```
| >sample1
| ATGCATGC
| >sample2
| ACGTACGT
```

この場合、sample1 の鋳型には 1 ラン目には ACGTACGT を付加し、2 ラン目では ATGCATGC を付加したことになります。sample2 の鋳型では逆になります。もしも、ほぼ同じ領域を増幅してはいるがプライマー配列は異なる場合、タグと同様に同じ ID を付けた別々のファイルを用意して与えて下さい。この後の解析に関しては後述します。

2.2　重複配列カウントによるノイジー配列とキメラ配列の除去

Claident では、配列のクラスタリング結果から読み間違いの多いノイジーな配列を推定して除去することができます。方法は CD-HIT-OTU (Li *et al.*, 2012) とほとんど同じです。また、旧パイプラインでは VSEARCH に実装された UCHIME (Edgar *et al.*, 2011) アルゴリズムを呼び出して結果を受け取ることで、キメラ配列の検出・除去も可能です。新パイプラインでは、クラスタリング処理が終わってからキメラ検出・除去処理を適用します。

2.2.1　VSEARCH を用いた新パイプラインの場合

以下のコマンドでノイジー配列の検出・除去が行えます。なお、多数のファイルを一度に与えることができますが、できるだけ 1 ラン分のファイルを与えて下さい。これは、ランごとに塩基配列決定の質にばらつきがあるためです。1 ラン分の配列が非常に多く (1000 万超) 時間がかかってしまう場合は、1 サンプルごとにこの処理を行って下さい。

```
> clcleanseqv \
--derepmode=PREFIX \
--primarymaxnmismatch=0 \
--secondarymaxnmismatch=1 \
--pnoisycluster=0.5 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

オプションのうち、--derepmode は前処理において全長一致 (FULLLENGTH) で配列をまとめるか、前方一致 (PREFIX) で配列をまとめるかを指定します。オーバーラップのあるペアエンド配列を連結して得られた配列を処理する場合は FULLLENGTH を、シングルエンド配列を処理する場合は PREFIX を指定して下さい。--primarymaxnmismatch はプライマリクラスタリングで許容するミスマッチ数で通常は 0 にします。--secondarymaxnmismatch はセカンダリクラスタリングで許容するミスマッチ数で、通常は 1 にします。ただし、「先頭から末尾までに読み間違いが全くない配列」がほとんどないと思われるようなあまりにもノイズの多いデータでは、--primarymaxnmismatch を 1 に、--secondarymaxnmismatch を 3 にしてみて下さい。それでもダメなら--primarymaxnmismatch を 2 に、--secondarymaxnmismatch を 5 にして試します。--secondarymaxnmismatch は--primarymaxnmismatch の 2 倍より 1 大きい値に設定して下さい。--pnoisycluster はノイジー配列と判定する感度に関するオプションです。

0 以上 1 以下で指定して下さい。大きいほど感度が上がります。デフォルト値・推奨値は 0.5 です。クラスタリングを最終的に 97% 以下の閾値で行うのであればこれでいいと思いますが、99% 前後でクラスタリングする場合、--pnoisycluster は 0.9 くらいにするといいかもしれません。ただし、レア OTU を捨ててしまう可能性が高くなるので、1 サンプル当たりの配列数を多めに読んでおく必要があります。

出力フォルダには、以下のファイル群が保存されます。

parameter.txt　プライマリクラスタをノイジーと判定するのに使用した所属配列数の下限値
primarycluster.denoised.fasta.gz　ノイジーと判定されなかったプライマリクラスタの代表配列
primarycluster.fasta.gz　プライマリクラスタの代表配列
secondarycluster.fasta.gz　セカンダリクラスタの代表配列
ラン ID__タグ ID__プライマー ID.noisyreads.txt.gz　ノイジー配列と判定された配列のリスト
ラン ID__タグ ID__プライマー ID.singletons.txt.gz　プライマリクラスタリングでシングルトンになった配列のリスト

他にも多くのファイルが出力されますが、後の解析に必要なものもありますので削除しないようにして下さい。

PCR レプリケート法によるノイジー配列・キメラ配列の除去

PCR レプリケートを用意してノイジー配列・キメラ配列を検出・除去するには、まず同一鋳型 DNA 由来の PCR レプリケートはどのサンプルとどのサンプルなのかを記したファイルを用意する必要があります。以下のような内容で用意して下さい。

```
| sample1  sample2  sample3
| sample4  sample5
| sample6  sample7
```

同一の行内にタブ区切りで書かれたサンプルが、同一鋳型由来の PCR レプリケートであることを表しています。異なる鋳型由来のサンプルは別の行に記述して下さい。PCR レプリケートは 3 つ以上あっても構いません。また、レプリケート数が鋳型ごとに異なっても構いません。

上記ファイルが用意できたら、以下のように clcleanseqv を実行することでレプリケート間で共通しないプライマリクラスタをキメラもしくはノイジーであると判断して除去します。

```
> clcleanseqv \
--replicatelist=PCR レプリケート指示ファイル \
--derepmode=PREFIX \
--primarymaxnmismatch=0 \
--secondarymaxnmismatch=1 \
--pnoisycluster=0.5 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

レプリケートが 3 つ以上であっても、全てのレプリケートで検出されたプライマリクラスタのみがノイジーでもキメラでもないと判断されます。また、1 つのプライマリクラスタが複数の鋳型から検出されていた場合、どれか 1 つの鋳型

でノイジーまたはキメラと判定されていれば、他の鋳型でもノイジーまたはキメラとみなして除去します。これらの設定は以下のオプションで変更することができます。

--minnreplicate　整数値で指定。この値以上のレプリケートで観測されたプライマリクラスタはそのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 2。
--minpreplicate　小数値で指定。プライマリクラスタが観測されたレプリケート数/サンプルの総レプリケート数がこの値以上であれば、そのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 1。
--minnpositive　整数値で指定。この値以上の配列 (サンプルではない) がノイジーまたはキメラと判定されていれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 1。
--minppositive　小数値で指定。ノイジーまたはキメラと判定された配列数/プライマリクラスタの総配列数がこの値以上であれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 0。
--runname　ラン ID を指定。サンプル名に含まれるラン ID が全てこれに置換される。同一サンプル名になったサンプルは統合される。

--minnreplicate と--minpreplicate の値に基づく 1 サンプル内での判定と、--minnpositive と--minppositive の値に基づく全サンプルにまたがる判定が 2 段階で行われていることに注意して下さい。前者では 2 つの条件を両方満たすとノイジーでもキメラでもないと判定され、後者では 2 つの条件を両方満たすとノイジーまたはキメラであると判定されます。また、同一のプライマリクラスタが、あるサンプルではノイジーまたはキメラだが、別のサンプルではそうでない、といった判定は行えません。最終的な判定は全サンプル共通です。また、PCR レプリケート指示ファイルに記されていないサンプルでは判定が行われません。

上記のように clcleanseqv を実行すると、以下のようなファイルも出力フォルダに保存されます。

primarycluster.chimeraremoved.fasta.gz　キメラと判定されなかったプライマリクラスタの代表配列
primarycluster.cleaned.fasta.gz　ノイジーでもキメラでもないと判定されたプライマリクラスタの代表配列
ラン ID__タグ ID__プライマー ID.chimericreads.txt.gz　キメラ配列と判定された配列のリスト

2.2.2　Assams を用いた旧パイプラインの場合

以下のコマンドでノイジー配列とキメラ配列の検出・除去が行えます。なお、多数のファイルを一度に与えることができますが、できるだけ 1 ラン分のファイルを与えて下さい。これは、ランごとに塩基配列決定の質にばらつきがあるためです 1 ラン分の配列が非常に多く (100 万超) 時間がかかってしまう場合は、1 サンプルごとにこの処理を行って下さい。

```
> clcleanseq \
uchime --minh 0.1 --mindiv 0.8 end \
--detectmode=N+C \
--pnoisycluster=0.5 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

オプションのうち、uchime から end の間は UCHIME の実行オプションです。--minh と--mindiv はキメラ配列と判定する感度に関するオプションです。--minh は値が小さいほど感度が高くなります。UCHIME の作者は 0.1 以上 5 以下の範囲にするよう推奨しています。デフォルト値は 0.1 です。--mindiv の指定値が 0.8 の場合、0.8% 以下の違

いしかない配列間はキメラが形成されていても無視されます。最終的に 97% 一致規準で配列をクラスタリングするとしたら、2% 程度の違いしかない配列間のキメラはクラスタリング・コンセンサス配列作成によって潰されるため、この値を大きくすることで計算が速くなります。また、--pnoisyclusterはノイジー配列と判定する感度に関するオプションです。0 以上 1 以下で指定して下さい。大きいほど感度が上がります。デフォルト値・推奨値は 0.5 です。クラスタリングを最終的に 97% 以下の閾値で行うのであればこれでいいと思いますが、99% 前後でクラスタリングする場合、--pnoisyclusterは 0.9 くらいにするといいかもしれません。ただし、レア OTU を捨ててしまう可能性が高くなるので、1 サンプル当たりの配列数を多めに読んでおく必要があります。

各入力ファイルは同一の鋳型を同一のチューブ内で PCR した配列群を含んでいる必要があります。Claident がキメラ検出に利用している UCHIME では、元配列数が多いコンセンサス配列ほど PCR 当初から存在した＝キメラではないと仮定し、元配列数が少ないコンセンサス配列が、より元配列数の多いコンセンサス配列の部分配列の組み合わせで説明できるかどうかを検討します。したがって、UCHIME に「同一の鋳型を同一のチューブ内で PCR した配列群内でのコンセンサス配列の元配列数」を正確に伝える必要があるため、1 ファイルは同一の鋳型を同一のチューブ内で PCR した配列群でなくてはならないのです。これは 1 ラン分のファイルを一度に与えることよりもはるかに優先度が高いので、同一の鋳型を同一のチューブ内で PCR したサンプルを複数回のランで塩基配列決定した場合などは複数ラン分のデータを入力しても構いません。

UCHIME によるキメラ判定はそれぞれのサンプルごとに行われ、複数のサンプルで観測されている完全一致配列の場合、どれか 1 つのサンプルでキメラと判定されていれば、他のサンプルでもキメラとみなして除去します。ただし、下記のオプションでこの動作を変更することができます。両オプションの両方を満たせばキメラと判定されます。

--minnpositive　整数値で指定。この値以上の配列 (サンプルではない) がキメラと判定されていれば、全サンプルでキメラと見なす。デフォルト値は 1。
--minppositive　小数値で指定。キメラと判定された配列数/完全一致した全配列数がこの値以上であれば、全サンプルでキメラと見なす。デフォルト値は 0。

出力フォルダには、以下のファイル群が保存されます。

ラン ID__タグ ID__プライマー ID.chimeraremoved.dereplicated.fastq.gz　キメラ配列除去し完全一致配列をまとめた配列
ラン ID__タグ ID__プライマー ID.chimeraremoved.fastq.gz　キメラ配列除去した配列
ラン ID__タグ ID__プライマー ID.denoised.dereplicated.fastq.gz　ノイジー配列除去し完全一致配列をまとめた配列
ラン ID__タグ ID__プライマー ID.denoised.fastq.gz　ノイジー配列除去した配列
ラン ID__タグ ID__プライマー ID.cleaned.dereplicated.fastq.gz　キメラ配列とノイジー配列を除去し完全一致配列をまとめた配列
ラン ID__タグ ID__プライマー ID.cleaned.fastq.gz　キメラ配列とノイジー配列を除去した配列
ラン ID__タグ ID__プライマー ID.chimericreads.txt.gz　キメラ配列と判定された配列のリスト
ラン ID__タグ ID__プライマー ID.noisyreads.txt.gz　ノイジー配列と判定された配列のリスト
ラン ID__タグ ID__プライマー ID.singletons.txt.gz　完全一致配列をまとめたときにシングルトンになった配列のリスト
ラン ID__タグ ID__プライマー ID.uchime.txt.gz　UCHIME によるキメラ判定結果のログ
ラン ID__タグ ID__プライマー ID.parameter.txt　まとまった完全一致配列をノイジーと判定するのに使用した所属配列数の下限値

他にも多くのファイルが出力されますが、後の解析に必要なものもありますので削除しないようにして下さい。キメラ配列除去を無効化してノイジー配列除去のみを行う場合は--detectmode=noise、逆にノイジー配列除去を無効化してキメラ配列除去のみを行う場合は--detectmode=chimeraオプションを付けて実行して下さい。ただし、--detectmode=noiseで処理した結果を--detectmode=chimeraで処理しても、一度にノイジー配列・キメラ配列除去を行った場合 (--detectmode=N+Cを指定した場合またはデフォルト設定) と結果は一致しませんのでご注意下さい。処理する場合は同時に行う必要があります。これは、キメラ配列もノイジー配列と判定するための情報を持っており、同時処理の場合にはそれを利用するためです。

PCR レプリケート法によるノイジー配列・キメラ配列の除去

PCR レプリケートを用意してノイジー配列・キメラ配列を検出・除去するには、まず同一鋳型 DNA 由来の PCR レプリケートはどのサンプルとどのサンプルなのかを記したファイルを用意する必要があります。以下のような内容で用意して下さい。

```
| sample1  sample2  sample3
| sample4  sample5
| sample6  sample7
```

同一の行内にタブ区切りで書かれたサンプルが、同一鋳型由来の PCR レプリケートであることを表しています。異なる鋳型由来のサンプルは別の行に記述して下さい。PCR レプリケートは 3 つ以上あっても構いません。また、レプリケート数が鋳型ごとに異なっても構いません。

上記ファイルが用意できたら、以下のように clcleanseq を実行することでレプリケート間で共通しないプライマリクラスタをキメラもしくはノイジーであると判断して除去します。

```
> clcleanseq \
--replicatelist=PCR レプリケート指示ファイル \
uchime --minh 0.1 --mindiv 0.8 end \
--detectmode=N+C \
--pnoisycluster=0.5 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

レプリケートが 3 つ以上であっても、全てのレプリケートで検出されたプライマリクラスタのみがノイジーでもキメラでもないと判断されます。また、1 つのプライマリクラスタが複数の鋳型から検出されていた場合、どれか 1 つの鋳型でノイジーまたはキメラと判定されていれば、他の鋳型でもノイジーまたはキメラとみなして除去します。これらの設定は以下のオプションで変更することができます。

--minnreplicate　整数値で指定。この値以上のレプリケートで観測されたプライマリクラスタはそのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 2。
--minpreplicate　小数値で指定。プライマリクラスタが観測されたレプリケート数/サンプルの総レプリケート数がこの値以上であれば、そのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 1。
--minnpositive　整数値で指定。この値以上の配列 (サンプルではない) がノイジーまたはキメラと判定されていれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 1。
--minppositive　小数値で指定。ノイジーまたはキメラと判定された配列数/プライマリクラスタの総配列数がこの値以上であれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 0。
--runname　ラン ID を指定。サンプル名に含まれるラン ID が全てこれに置換される。同一サンプル名になったサンプルは統合される。

--minnreplicate と--minpreplicate の値に基づく 1 サンプル内での判定と、--minnpositive と--minppositive の値に基づく全サンプルにまたがる判定が 2 段階で行われていることに注意して下さい。前者では 2 つの条件を両方満たすとノイジーでもキメラでもないと判定され、後者では 2 つの条件を両方満たすとノイジーまたはキメラであると

判定されます。また、同一のプライマリクラスタが、あるサンプルではノイジーまたはキメラだが、別のサンプルではそうでない、といった判定は行えません。最終的な判定は全サンプル共通です。また、PCR レプリケート指示ファイルに記されていないサンプルでは判定が行われません。

2.3 サンプル内での配列クラスタリング

2.3.1 VSEARCH を用いた新パイプラインの場合

新パイプラインではこのステップはありません。

2.3.2 Assams を用いた旧パイプラインの場合

以下のコマンドで、同一の鋳型を同一のチューブ内で PCR した配列群のクラスタリングを行います。前節の結果得られる、完全一致配列をまとめた結果 (ラン ID__タグ ID__プライマー ID.cleaned.dereplicated.fastq.gz) を入力します。

```
> clclassseq \
--minident=0 \
--strand=plus \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

前節の出力ファイルに対してこの処理を適用する場合、以下のようにコマンドを打てばいいでしょう。

```
> clclassseq \
--minident=0 \
--strand=plus \
--numthreads=使用する CPU 数 \
"前節の出力フォルダ/*.cleaned.dereplicated.fastq.gz" \
出力フォルダ↓
```

オプションのうち、--minident=0 は不一致が 1 塩基以下の配列をまとめる設定です (実際には、類似度が (最短配列の長さ-11/最短配列の長さ-10) 以上の配列をまとめます。最長配列の長さが 400bp で最短配列の長さが 350bp なら、0.997 となり、400bp の配列でも 1 塩基しか不一致は許容されませんので意図通りになりますが、最短配列と最長配列の差が大きすぎると意図した通りになりません。そのため、最短配列は最長配列の半分より 11 塩基以上長くなくてはなりません)。最終的に 97% で配列をまとめる場合もこの時点では 0 にしておくのがおすすめです。最終的に 99% でまとめる場合にもここでは 0 にしておきます。--strand=plus は同一ストランドの比較だけでクラスタリングを行うオプションです。片側からしか読んでいないので、両ストランドの比較をする必要はなく、比較するとかえって異常なクラスタリングをしかねません。両側から読んでいる場合も、連結した後はストランドは揃っているのでこれでいいはずです。出力フォルダ内の*.assembled.fastq.gz がクラスタリング後の配列です。

2.4　サンプル間での配列クラスタリング

2.4.1　VSEARCH を用いた新パイプラインの場合

塩基配列のクラスタリングを行って OTU を作成するには、以下のコマンドを実行して下さい。

```
> clclassseqv \
--minident=0.97 \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

入力ファイルには、clcleanseqv の実行時に PCR レプリケート法によるノイジー配列・キメラ配列の除去を併用した場合は primarycluster.cleaned.fasta.gz を、併用しなかった場合には primarycluster.denoised.fasta.gz を与えて下さい。

出力フォルダには、以下のファイルが保存されています。

clustered.otu.gz　どの生配列がどの OTU に割り当てられたかを記録しているファイル
clustered.fasta.gz　OTU の代表配列

2.4.2　Assams を用いた旧パイプラインの場合

サンプル内クラスタリングデータを用いてサンプル間でのクラスタリングを行い、全体のクラスタリング結果を得るには以下のコマンドを実行します。1 サンプルしかない場合も、上記のサンプル内配列クラスタリングを行ってからこの処理を行って下さい。

```
> clclassclass \
--minident=0 \
--strand=plus \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ 1 ↓
> clreclassclass \
--minident=0.99 \
--strand=plus \
--numthreads=使用する CPU 数 \
出力フォルダ 1 \
出力フォルダ 2 ↓
> clreclassclass \
```

```
--minident=0.97 \
--strand=plus \
--numthreads=使用する CPU 数 \
出力フォルダ 2 \
出力フォルダ 3 ↓
```

ここでの入力ファイルは前節の出力ファイル*.assembled.fastq.gz です。したがって、実際には以下のように入力することになるでしょう。

```
> clclassclass \
--minident=0 \
--strand=plus \
--numthreads=使用する CPU 数 \
"前節の出力フォルダ/*.assembled.fastq.gz" \
出力フォルダ 1 ↓
> clreclassclass \
--minident=0.99 \
--strand=plus \
--numthreads=使用する CPU 数 \
出力フォルダ 1 \
出力フォルダ 2 ↓
> clreclassclass \
--minident=0.97 \
--strand=plus \
--numthreads=使用する CPU 数 \
出力フォルダ 2 \
出力フォルダ 3 ↓
```

1 つ目の clclassclass でサンプル内配列クラスタリングでまとめられた配列において、不一致が 1 塩基以下の配列をまとめます。2 つ目の clreclassclass で、99% 以上一致する配列をまとめ直します。さらに 3 つ目の clreclassclass で、97% 以上一致する配列をまとめ直します。こうすることで、より高速に、より正確にクラスタリングが行われます。clreclassclass を使用せずに clclassclass に 97% でまとめるよう指示してしまうと、アルゴリズム上「まとめすぎ」が発生しやすくなります。

出力フォルダには、以下のファイルが保存されています。

assembled.contigmembers.gz　どの生配列がどの OTU に割り当てられたかを記録しているファイル
assembled.fastq.gz　OTU のコンセンサス配列
assembled.fasta　OTU のコンセンサス配列

2.5 OTU 代表配列への生配列のマッピング

ノイジー配列・キメラ配列と判定されてしまった配列でも、OTU の代表配列に対して指定した類似度の閾値で貼り付けることが可能なのであれば、復活させることができます。これにより、データの減少を抑えることができます。

2.5.1　VSEARCH を用いた新パイプラインの場合

以下のコマンドを実行して下さい。

```
> clrecoverseqv \
--minident=0.97 \
--centroid=OTU 代表配列のファイル \
--numthreads=使用する CPU 数 \
入力ファイル 1 \
中略 \
入力ファイル N \
出力フォルダ↓
```

OTU 代表配列のファイルには clclassseqv の出力した clustered.fasta.gz を用います。入力ファイルとしては、clcleanseqv の出力した primarycluster.fasta.gz または clcleanseqv での処理や低品質配列の除去を行う前の配列ファイルを使用して下さい。出力フォルダには、clclassseqv と同様のファイル群が作成されます。

2.5.2　Assams を用いた旧パイプラインの場合

旧パイプラインではこのステップはありません。

2.6　結果の要約と後処理

2.6.1　集計表の作成

以下のコマンドで、クラスタリング結果から各サンプルにおける各 OTU に割り当てられた生配列数を表にすることができます。

```
> clsumclass \
--output=Matrix \
入力ファイル \
出力ファイル↓
```

ここでの入力ファイルは、クラスタリング結果の出力フォルダに保存されている、clustered.otu.gz (新パイプライン) または assembled.contigmembers.gz (旧パイプライン) というファイルです。出力されるファイルは表 2.1 のようなタブ区切りのテキストファイルで、Excel などの表計算ソフトや R で読み込むことができます。ただし、表計算ソフトは巨大な行列の全てを読み込むことができない可能性がありますので注意して下さい。このファイルに基づいて、群集生態学的な解析を行うことができます。

samplename	amongsampleotu_1	amongsampleotu_2	amongsampleotu_3
sampleA	2371	0	0
sampleB	0	1518	0
sampleC	1398	0	0
sampleD	0	1436	0
sampleE	0	0	1360
sampleF	0	0	977

表 2.1 集計表の例 — 数値は各サンプルにおける各 OTU の観測配列数です。

2.6.2 集計表からの特定の OTU・サンプルの除去

`clsumclass` では、1 回だけ出現した配列も含めて全て出力されますが、以下のコマンドを用いてサンプルや OTU を選別することができます。入力ファイルは `clsumclass` の出力ファイルです。

```
> clfiltersum \
オプション \
入力ファイル \
出力ファイル↓
```

使用できるオプションは以下の通りです。

`--minnseqotu` 整数値で指定。この値に基づいて「割り当てられた生配列数がどのサンプルでも指定値未満の OTU」を除去します。
`--minpseqotu` 小数値で指定。この値に基づいて「割り当てられた生配列数/サンプルの生配列総数がどのサンプルでも指定値未満の OTU」を除去します。
`--minntotalseqotu` 整数値で指定。この値に基づいて「割り当てられた生配列総数が指定値未満の OTU」を除去します。
`--minnseqsample` 整数値で指定。この値に基づいて「どの OTU の生配列数も指定値未満のサンプル」を除去します。
`--minpseqsample` 小数値で指定。この値に基づいて「そのサンプルのその OTU に割り当てられた生配列数/OTU の生配列総数がどの OTU でも指定値未満のサンプル」を除去します。
`--minntotalseqsample` 整数値で指定。この値に基づいて「生配列総数が指定値未満のサンプル」を除去します。
`--otu` 残したい OTU 名をカンマで区切って指定します。
`--negativeotu` 除去したい OTU 名をカンマで区切って指定します。
`--otulist` ファイル名を指定。1 行に 1 つの OTU 名を記したテキストファイルとして残したい OTU 名を指定します。
`--negativeotulist` ファイル名を指定。1 行に 1 つの OTU 名を記したテキストファイルとして除去したい OTU 名を指定します。
`--otuseq` ファイル名を指定。FASTA 形式の配列ファイルとして残したい OTU 名を指定します。
`--negativeotuseq` ファイル名を指定。FASTA 形式の配列ファイルとして除去したい OTU 名を指定します。
`--sample` 残したいサンプル名をカンマで区切って指定します。
`--negativesample` 除去したいサンプル名をカンマで区切って指定します。
`--samplelist` ファイル名を指定。1 行に 1 つのサンプル名を記したテキストファイルとして残したいサンプル名を指定します。
`--negativesamplelist` ファイル名を指定。1 行に 1 つのサンプル名を記したテキストファイルとして除去したいサンプル名を指定します。
`--replicatelist` ファイル名を指定。PCR レプリケート関係にあるサンプルをタブ区切りで同一行に記述する。出力で 1 つのサンプルに統合される。
`--runname` ラン ID を指定。集計表中のサンプル名に含まれるラン ID が全てこれに置換される。同一サンプル名になったサンプルは統合される。

後述するノイジー配列・キメラ配列の除去や他領域配列の除去を行った上で、残った配列を用いて`--otuseq` などのオプションを用いた OTU の選別を行う場合は、他の選別より先に行って下さい。

2.6.3　PCR レプリケート法によるノイジー配列・キメラ配列の除去

新パイプラインでは clcleanseqv の実行時に行っていますが、旧パイプラインでも PCR レプリケートを用意してノイジー配列・キメラ配列を検出・除去することが可能です。それには、まず同一鋳型 DNA 由来の PCR レプリケートはどのサンプルとどのサンプルなのかを記したファイルを用意する必要があります。以下のような内容で用意して下さい。

```
| sample1  sample2  sample3
| sample4  sample5
| sample6  sample7
```

同一の行内にタブ区切りで書かれたサンプルが、同一鋳型由来の PCR レプリケートであることを表しています。異なる鋳型由来のサンプルは別の行に記述して下さい。PCR レプリケートは 3 つ以上あっても構いません。また、レプリケート数が鋳型ごとに異なっても構いません。

上記ファイルが用意できたら、以下のコマンドでクラスタリング結果からレプリケート間で共通しない配列を除去します。

```
> clfilterseq \
--contigmembers=*.contigmembers.gz \
--replicatelist=PCR レプリケート指示ファイル \
入力ファイル \
出力ファイル↓
```

レプリケートが 3 つ以上であっても、全てのレプリケートで検出された配列のみがノイジーでもキメラでもないと判断されます。また、1 つの OTU が複数の鋳型から検出されていた場合、どれか 1 つの鋳型でノイジーまたはキメラと判定されていれば、他の鋳型でもノイジーまたはキメラとみなして除去します。これらの設定は以下のオプションで変更することができます。

--minnreplicate　整数値で指定。この値以上のレプリケートで観測された OTU はそのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 2。
--minpreplicate　小数値で指定。OTU が観測されたレプリケート数/サンプルの総レプリケート数がこの値以上であれば、そのサンプルではノイジーでもキメラでもないと見なす。デフォルト値は 1。
--minnpositive　整数値で指定。この値以上の配列 (サンプルではない) がノイジーまたはキメラと判定されていれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 1。
--minppositive　小数値で指定。ノイジーまたはキメラと判定された配列数/OTU の総配列数がこの値以上であれば、全サンプルでノイジーまたはキメラと見なす。デフォルト値は 0。
--runname　ラン ID を指定。サンプル名に含まれるラン ID が全てこれに置換される。同一サンプル名になったサンプルは統合される。

--minnreplicate と --minpreplicate の値に基づく 1 サンプル内での判定と、--minnpositive と --minppositive の値に基づく全サンプルにまたがる判定が 2 段階で行われていることに注意して下さい。前者では 2 つの条件を両方満たすとノイジーでもキメラでもないと判定され、後者では 2 つの条件を両方満たすとノイジーまたはキメラであると判定されます。また、同一の OTU が、あるサンプルではノイジーまたはキメラだが、別のサンプルではそうでない、といった判定は行えません。最終的な判定は全サンプル共通です。また、PCR レプリケート指示ファイルに記されて

いないサンプルでは判定が行われません。

ここで作成した FASTA 配列ファイルを第 2.6.2 節の clfiltersum の--otuseq オプションに指定して集計表を加工することで、この配列ファイルから除去された OTU を集計表からも除去することができます。また、--replicatelist オプションと--runname オプションを clfilterseq と同様に指定すれば、PCR レプリケート関係にあるサンプルは 1 つのサンプルに統合されます (セルの値は合算されます)。

2.6.4 UCHIME によるキメラ配列除去

旧パイプラインではクラスタリング前にも UCHIME でキメラ配列除去は行っていますが、クラスタリング後の配列に対してリファレンスなしでのキメラ配列除去とリファレンスありでのキメラ配列除去を追加で行うことができます。両方行う場合は、リファレンスなしでのキメラ配列除去を先に行ってから、リファレンスありでのキメラ配列除去を行います。両方のキメラ配列除去を行う場合は、クラスタリング前のノイジー配列・キメラ配列除去において、キメラ配列除去を無効にしてもいいかもしれません。クラスタリング後のキメラ配列除去の方がずっと高速なので、1 サンプル当たりの配列が膨大でクラスタリング前のキメラ配列除去では長時間を要する場合には、この方法の方が適しています。なお、PCR レプリケート法によるノイジー配列・キメラ配列の除去も併用する場合は、PCR レプリケート法によるノイジー配列・キメラ配列の除去を最初に行います。利用するコマンドは clrunuchime です。このコマンドでは、UCHIME アルゴリズムを実装している VSEARCH というプログラムを呼び出して、キメラ配列除去を行います。

リファレンスなしでのキメラ配列除去

clrunuchime を以下のように実行して下さい。

```
> clrunuchime \
--contigmembers=*.contigmembers.gz \
--otufile=*.otu.gz \
入力ファイル \
出力フォルダ↓
```

旧パイプラインを使用した場合は--contigmembers オプションを、新パイプラインを使用した場合は--otufile オプションを使用します。上記の例は一度に説明するためのもので、両方を指定してはいけません。なお、どちらも指定しなかった場合、*.otu.gz が入力ファイルと同じフォルダに存在すればそれを、*.contigmembers.gz が存在すればそれを見つけ出して使用しますので、普段は何も指定しなくてもいいでしょう。

リファレンスありでのキメラ配列除去

clrunuchime を以下のように実行して下さい。リファレンスなしでのキメラ配列除去を行った場合は、nonchimeras.fasta というファイルがあるはずですので、それを入力ファイルにすればいいでしょう。

```
> clrunuchime \
--referencedb=リファレンスデータベース名 \
入力ファイル \
```

```
出力フォルダ↓
```

UCHIME 用のリファレンスデータベース名と内容は以下の通りです。

rdpgoldv9　細菌 16S 用の RDP Gold データベース (v9 版)
unite20160101　真菌 ITS 用の UNITE データベース (20160101 版)
unite20160101untrim　真菌 ITS 用の UNITE データベース (20160101 トリミングなし版)
unite20160101its1　真菌 ITS 用の UNITE データベース (20160101ITS1 版)
unite20160101its2　真菌 ITS 用の UNITE データベース (20160101ITS2 版)

出力フォルダの内容について

出力フォルダに保存されるファイルは以下の 4 つです。

chimeras.fasta　キメラと判定された配列
nonchimeras.fasta　キメラでないと判定された配列
uchimealns.txt　判定時のアライメント
uchimeout.txt　判定時の親配列やスコアなど

uchimeout.txt の各項目の意味は
http://drive5.com/usearch/manual/uchimeout.html
をご覧下さい。

2.6.5　配列からの低頻度出現配列の除去

クラスタリングの出力フォルダには、clustered.fasta.gz (新パイプライン) または assembled.fasta (旧パイプライン) というファイルとして、最終的に得られた代表配列もしくはコンセンサス配列が保存されています。これを次章の DNA バーコーディングに用いますが、全サンプルを通して少量しか出現しないものまで含めると数が多すぎることがあります。そこで、以下のようにして 5 配列以上出現するものだけを抽出することができます。配列数の閾値を適当に変えてもいいでしょう。この閾値は慎重に決定して下さい。ノイジー配列・キメラ配列の除去を行なっている場合には必要ないかもしれません。

```
> clfilterseq \
--contigmembers=*.contigmembers.gz \
--otufile=*.otu.gz \
--minnseq=5 \
入力ファイル \
出力ファイル↓
```

旧パイプラインを使用した場合は--contigmembers オプションを、新パイプラインを使用した場合は--otufile オプションを使用します。上記の例は一度に説明するためのもので、両方を指定してはいけません。なお、どちらも指定しなかった場合、*.otu.gz が入力ファイルと同じフォルダに存在すればそれを、*.contigmembers.gz が存在すればそれを見つけ出して使用しますので、普段は何も指定しなくてもいいでしょう。ここで作成した FASTA 配列ファイルを

第 2.6.2 節の `clfiltersum` の`--otuseq` オプションに指定して集計表を加工することで、この配列ファイルから除去された OTU を集計表からも除去することができます。

2.6.6 保存領域配列認識による領域分割

ここまでの配列が複数領域にまたがっている場合 (たとえば ITS1・5.8S rRNA・ITS2 といった風に)、領域ごとに分割した方が良いかもしれません。領域間かその付近に保存的な領域があれば、それを境界の目印として領域を分割できます。特にユニバーサルプライマー配列などを使うといいでしょう。これは以下のコマンドで行うことができます。

```
> cldivseq \
--query=保存領域の配列 \
--border=start \
入力ファイル \
出力ファイル 1 \
出力ファイル 2 ↓
```

このコマンドでは、指定された配列を Needleman-Wunsch アルゴリズムを用いたアライメントによって 15% まで不一致を許して探します。該当する部位が見つかったら、その左端を境界として配列を 2 つに分け、それぞれ出力ファイル 1 と出力ファイル 2 に保存します。該当する部位が見つからなかった場合は、そのまま出力ファイル 1 だけに保存されます。信頼度配列ファイルがある場合には、こちらも塩基配列に合わせて分割されます。なお、指定する配列は対象配列と同一のストランドにしておいて下さい。ストランドが異なる場合は、`--reversecomplement` オプションを付加して実行することで、`--query` に指定された配列の逆相補配列をアライメントに用いるようになります。

`--border` オプションを `start` ではなく `end` にすることで、検索に一致した部位の右端を境界として分割することができます。検索した配列が見つからなかった場合は出力ファイル 1 のみに保存され、出力ファイル 2 には配列が保存されませんが、これでは都合が悪い場合は`--makedummy` オプションを付加することでダミーの塩基配列が出力ファイル 2 に保存されます。分割した各領域の配列で別個にホスト生物を同定した上で同定結果を統合する場合など、配列間で対応がとれていないと困る場合にご利用下さい。ここでは 2 分割の例を挙げましたが、分割後にさらに分割を繰り返すことで 3 分割や 4 分割も可能です。

2.6.7 ITSx や Metaxa による ITS・SSU rRNA 配列の抽出

ITSx は核 ITS1・ITS2 領域を認識してその部位だけを取り出すことができるプログラムです (Bengtsson-Palme *et al.*, 2013)。多くの分類群で ITS は非常に変異に富んでいますが、ずっと保守的な SSU・LSU rRNA と隣接しているため、これらの配列が残っていると、その配列の影響で ITS では明確に区別できる遠縁な生物の SSU・LSU 配列が BLAST 検索時にヒットしてしまい、うまく同定できなくなってしまうことがあります。ITSx で ITS 領域のみを切り出して同定することでこのような問題に対処できます。

Metaxa は ITSx と同様に SSU (12S/16S/18S) rRNA を区別してそれぞれの部位だけを取り出すことができるプログラムです (Bengtsson *et al.*, 2011)。SSU rRNA は真核生物のメタバーコーディング・DNA バーコーディングに広く利用されている領域ですが、核だけでなくミトコンドリアや葉緑体、混入した細菌にも含まれているため、核の SSU を標的とする PCR を行った場合にも多少はミトコンドリアや細菌の SSU rRNA が混入してしまいます。そのような配列

は群集生態学的解析の障害となるため、このプログラムで前もって除去しておくと後の解析が楽になります。

いずれのプログラムもインストール方法、使用方法をここでは説明しません。それぞれの Web サイトとマニュアルをご覧下さい。これらのプログラムで作成した FASTA 配列ファイルを第 2.6.2 節の `clfiltersum` の`--otuseq` オプションに指定して集計表を加工することで、この配列ファイルに含まれている OTU だけの集計表を作成することができます。

2.6.8　非ターゲット領域の探索と削除

ITSx や Metaxa は ITS と SSU rRNA 領域にしか使用できませんが、もっと汎用的な非ターゲット領域の探索方法があります。一つは BLAST などで検索して配列がどの遺伝子のものかを判定してくれるプログラムにかけることで、もう一つは多重整列プログラムを利用することです。ClustalW2・ClustalX2 (Larkin *et al.*, 2007) や MAFFT (Katoh and Standley, 2013) には、多重整列した結果を系統的に近いものが近くになるように並び替えて出力することができます。この機能を使って並び替えた上で、多重整列の閲覧ソフトで表示して目で確認すればどれが非ターゲット領域かすぐわかります。非ターゲット領域の OTU を除去した FASTA 配列ファイルを第 2.6.2 節の `clfiltersum` の`--otuseq` オプションに指定して集計表を加工することで、この配列ファイルに含まれている OTU だけの集計表を作成することができます。

2.7　DRA への登録

クラスタリング後の配列は従来通り DDBJ などに登録すればいいですが、新型シーケンサーの生データは DDBJ Sequence Read Archive (DRA) というところへ登録することになっています (サンガー法シーケンサーの生データは Trace Archive)。本書にあるように複数サンプルからの DNA にタグを付けてシーケンスした場合、サンプルごとのデータファイルに分割して登録することを求められます。これは `clsplitseq` によって分割した FASTQ を使えば問題ありません。同一鋳型由来配列ファイルが複数ある場合は、GZIP 圧縮を解除してから連結して再圧縮するだけで構いません。

また、サンプルについての情報 (メタデータ) を記述した XML ファイルを作成しなくてはなりません。データ登録を受け付けている側でも、XML 作成を補助するツールが用意されていますが大量サンプルを扱っている場合は用意するのは大変です。そこで、DRA 登録用 XML 作成を補助する `clmaketsv` および `clmakexml` というコマンドを用意しています。以下でこれらの使用法を説明します。

DRA への登録には、DDBJ が管理する D-way でのユーザーアカウント作成と公開鍵の登録が必要です。詳しくは DRA Handbook
http://trace.ddbj.nig.ac.jp/dra/submission.shtml
をご覧下さい。登録の際、Submission、Study、Experiment、Sample、Run の概念と対応関係を把握している必要がありますので、その部分を特に注意して読んで理解しておいて下さい。

DRA では、Study は BioProject という研究プロジェクトデータベースに登録して、それを参照することになります。BioProject への登録は BioProject Handbook
http://trace.ddbj.nig.ac.jp/bioproject/submission.html

を参照して予め済ませておいて下さい。

さらに、Sample は BioSample という研究サンプルデータベースに登録して、それを参照します。BioSample への登録は BioSample Handbook
http://trace.ddbj.nig.ac.jp/biosample/submission.html
を参照して予め済ませておいて下さい。なお、メタゲノムを鋳型としてユニバーサルプライマーで増幅を行ったシーケンスサンプルの場合、MIMarks-Survey を MIxS のタイプとして指定します。サンプルの採集された高度、水深、温度、湿度、pH など、様々な情報を付加しておくことができます。後世のためにも、面倒がらずにできるだけ多くの情報を付加しておいて下さい。なお、各項目の意味は、Genomic Standards Consortium から提供されているチェックリスト
http://wiki.gensc.org/index.php?title=MIMARKS
を確認して下さい。これを見てもわからない場合は DDBJ に問い合わせて下さい。特に、地下や海底下のサンプルでは、平均海水面からサンプリング地点までの高さ・深さ、地面からサンプリング地点までの高さ・深さ、平均海水面から地面までの高さ・深さを入れる項目があってややこしいのでご注意下さい。

2.7.1 XML 作成用タブ区切りテキストの作成

DRA に登録する XML には、登録する FASTQ ファイルごとにそのファイルに含まれている配列の情報を記述します。これは非常に複雑で大変なため、一旦単純なタブ区切りテキストに書き出した上で、Excel などを用いてそれを編集し、編集後のタブ区切りテキストから XML を作成します。タブ区切りテキストを生成するコマンドは `clmaketsv` です。以下のように使って下さい。

```
> clmaketsv \
入力ファイル 1 \
中略 \
入力ファイル N \
出力ファイル↓
```

入力ファイルは登録するものを指定して下さい。ワイルドカードも使えます。タブ区切りテキストができたら、表計算ソフトなどに読み込ませて、各セルを埋めていって下さい。なお、Excel では特定の文字列を勝手に日時などと認識して変換してしまう機能 (無効にする方法はありません) がありますので、十分注意して下さい。各セル内では、`[Hogehoge,Fugafuga]` とあった場合は `Hogehoge` または `Fugafuga` のどちらかを選んで括弧と選ばなかったものを消して下さい。`<Fill in this cell>`とあった場合は、<>内の指示に従ってセルを埋めて下さい (括弧は残さない)。その他のセルは適当に自分で考えて何とかして下さい。なお、BioProject ID や BioSample ID は、正式なアクセッション番号がまだ発行されていない場合、DDBJ に申請したときの Submission ID (BioProject では PSUB から始まり、BioSample では SSUB から始まるもの) を入力しておけば問題ありません。

2.7.2 タブ区切りテキストからの DRA 登録用 XML ファイルの生成

タブ区切りテキストの編集が終わったら、タブ区切りテキストとして保存した上で以下のように `clmakexml` を実行して下さい。登録に必要な XML ファイル群が生成されます。なお、タブ区切りテキストは一度に複数指定することも可

能です。ただし、各ファイルの最初の 3 行に記述する内容は、最初のファイルのものしか利用されません。2 ファイル目以降の最初の 3 行は無視されます。

```
> clmakexml \
タブ区切りテキストファイル \
DRA から割り振られた submission-ID ↓
```

DRA では、「Create new submission(s)」によって submission-ID が割り振られます。**ユーザー ID-0001** などとなっているはずです。**DRA から割り振られた submission-ID** にはこれを指定します。すると、**submission-ID.*.xml** という名前のファイルが 3 つ生成されます。これを DRA の XML Upload 機能を利用して送信します。その他もろもろの処理が終わると、登録してあるメールアドレスに割り振られたアクセッション番号が送られてきます。論文にはそれを書いておけばいいでしょう。

第 3 章

DNA バーコーディングによる配列のホスト生物同定

DNA バーコーディングは、近年非常に多くの分野で応用が進められている、生物の同定方法です。しかし、既知配列データベースが不十分な上、それを前提とした同定アルゴリズムが欠けていました。そこで、筆者は「問い合わせ配列と最近隣配列との間の変異量＜分類群内の最大変異量」という規準を考案し、これを実現するアルゴリズム QCauto 法 (Tanabe and Toju, 2013) を Claident に実装しました。ただし、その配列のホストの可能性がある全生物が記載済みで、しかもそのバーコード配列もデータベースに登録済みである場合には、そのような難しいことを考えずに最近隣配列と同種とみなせばよいでしょう。そちらの方法も Claident には実装してあります。以下のコマンドは全てターミナルかコンソールで実行して下さい。基本的なターミナルの使い方の知識は持っているものとして話を進めます。

3.1 BLAST 検索による近隣既知配列群の取得

以下のコマンドで、「問い合わせ配列と最近隣配列との間の変異量＜分類群内の最大変異量」という条件を満たすために検討すべき近隣既知配列群 (の GenBank ID) を取得できます。

```
> clidentseq \
--blastdb=overall_genus \
--numthreads=使用する CPU 数 \
入力ファイル \
出力ファイル↓
```

入力ファイルには FASTA 形式の塩基配列ファイルを指定します。`--blastdb` オプションでは、BLAST 検索に用いるデータベース名を指定します。筆者が用意しているデータベースは以下の通りです。

animals_COX1_genus　動物 (Metazoa) の mtDNA *COX1* 配列で属以下の情報があるもの
animals_COX1_species　同上だが種以下の情報があるもの
animals_mt_genus　動物 (Metazoa) の mtDNA 配列で属以下の情報があるもの
animals_mt_species　同上だが種以下の情報があるもの
eukaryota_LSU_genus　真核生物の LSU (28S) rRNA 配列で属以下の情報があるもの
eukaryota_LSU_species　同上だが種以下の情報があるもの
eukaryota_SSU_genus　真核生物の SSU (18S) rRNA 配列で属以下の情報があるもの
eukaryota_SSU_species　同上だが種以下の情報があるもの

fungi_ITS_genus　真菌の ITS 配列で属以下の情報があるもの
fungi_ITS_species　同上だが種以下の情報があるもの
overall_class　NCBI nt の中で綱以下の情報があるもの
overall_order　同上だが目以下の情報があるもの
overall_family　同上だが科以下の情報があるもの
overall_genus　同上だが属以下の情報があるもの
overall_species　同上だが種以下の情報があるもの
plants_matK_genus　緑色植物の cpDNA *matK* 配列で属以下の情報があるもの
plants_matK_species　同上だが種以下の情報があるもの
plants_rbcL_genus　緑色植物の cpDNA *rbcL* 配列で属以下の情報があるもの
plants_rbcL_species　同上だが種以下の情報があるもの
plants_trnH-psbA_genus　緑色植物の cpDNA *trnH–psbA* 配列で属以下の情報があるもの
plants_trnH-psbA_species　同上だが種以下の情報があるもの
prokaryota_16S_genus　原核生物の 16S rRNA 配列で属以下の情報があるもの
prokaryota_16S_species　同上だが種以下の情報があるもの
prokaryota_all_genus　原核生物の配列で属以下の情報があるもの
prokaryota_all_species　同上だが種以下の情報があるもの
semiall_class　`overall_class` から脊椎動物と *Caenorhabditis* と *Drosophila* を除いたもの
semiall_order　`overall_order` から脊椎動物と *Caenorhabditis* と *Drosophila* を除いたもの
semiall_family　`overall_family` から脊椎動物と *Caenorhabditis* と *Drosophila* を除いたもの
semiall_genus　`overall_genus` から脊椎動物と *Caenorhabditis* と *Drosophila* を除いたもの
semiall_species　`overall_species` から脊椎動物と *Caenorhabditis* と *Drosophila* を除いたもの

`overall_*`は非常に巨大なため、多くのメモリを必要としますが、これらは万能なのでどんな配列にも使えますし、想定外の分類群や想定外の遺伝子座の配列の混入があってもとんでもない誤同定を避けることができます。`overall_genus` では属以下の情報がある、つまりそれなりに分類群の情報が信頼できそうな配列を選別していますが、あまりにマイナーな分類群だとそれでは近隣配列が見つからないことがあります。その場合は、綱以下まで同定された配列のデータベース `overall_class` を使ってみて下さい。その他のデータベースは、`overall_*`よりもメモリ占有量がずっと少ないため、メモリの少ないコンピュータでも動作します。

3.1.1　キャッシュデータベースの構築による高速化

`clidentseq` は BLAST 検索は何度も繰り返しますが、それなりの時間と大量のメモリを必要とします。そこで、予め問い合わせ配列と類似した配列の上位 1 万本 (設定で変更可) のヒットした部位のみを BLAST データベースから取得してキャッシュとなる BLAST データベースを作成しておき、`clidentseq` ではこれを対象として検索することで、劇的な高速化が可能です。これを行うのが `clmakecachedb` で、以下のように用います。

```
> clmakecachedb \
--blastdb=overall_genus \
--numthreads=使用する CPU 数 \
入力ファイル \
出力フォルダ↓
```

`clidentseq` の実行時には、`--blastdb` オプションに上記コマンドの出力フォルダを指定することで、キャッシュデータベースが使われます。ただし、`clmakecachedb` と `clidentseq` の入力ファイルは同一である必要があります。メモリの使用量も大幅に減少しますので、`overall_*`を使用してもメモリ 16GB 程度で済みます。

3.1.2 参照配列データベースが全種を網羅している場合

問い合わせ配列のホストの可能性がある全生物が記載済みで、しかもそのバーコード配列も全て参照配列データベースに登録済みである場合には、以下のようにして類似度 99% 以上の上位 1 位 (1 位タイを含む) を近隣配列とすればいいでしょう。

```
> clidentseq \
blastn -task megablast -word_size 16 end \
--method=1,99% \
--blastdb=overall_genus \
--numthreads=使用する CPU 数 \
入力ファイル \
出力ファイル↓
```

類似度 99% 以上の上位 1 位の配列を探し出す場合、BLAST 検索オプションは`-task megablast -word_size 16` としても見逃すことはまずないでしょうから、ここでは高速化のためにこのように指定しています。

3.2 近隣既知配列群に基づく同定

以下のコマンドで、近隣既知配列群の所属分類群が同一になるまで分類階層を上げていくことで配列を同定します。これは lowest common ancestor (LCA) algorithm と呼ばれています (Huson *et al.,* 2007)。

```
> classigntax \
--taxdb=overall_genus \
入力ファイル \
出力ファイル↓
```

入力ファイルには前節で作成した `clidentseq` の出力ファイルを指定して下さい。`--taxdb` は既知配列の所属分類群データベース指定オプションです。BLAST データベースと同じ名前のものがインストールされていますので、それを指定して下さい。

`classigntax` のデフォルト設定では、少なくとも 2 本以上の近隣既知配列がないと同定することができません。前節後半のように `clidentseq` の実行時に`--method=1,99%` を指定していると、近隣既知配列が 1 本しかないために一つも同定できないことになります。この場合は以下のようにして必要な近隣既知配列数を 1 にします。

```
> classigntax \
--taxdb=overall_genus \
--minnsupporter=1 \
入力ファイル \
出力ファイル↓
```

出力ファイルは表 3.1 のような形のタブ区切りのテキストになっています。

query	phylum	genus	species
seqA	Ascomycota	*Chloridium*	*Chloridium virescens*
seqB	Ascomycota	*Chloridium*	*Chloridium virescens*
seqC	Ascomycota	*Chloridium*	*Chloridium virescens*
seqD	Basidiomycota	*Amanita*	*Amanita fuliginea*
seqE	Basidiomycota	*Coltriciella*	*Coltriciella dependens*
seqF	Basidiomycota	*Filobasidium*	*Filobasidium uniguttulatum*
seqG	Basidiomycota	*Laccaria*	*Laccaria bicolor*
seqH	Basidiomycota	*Lactarius*	*Lactarius quietus*
seqI	Basidiomycota	*Russula*	*Russula densifolia*
seqJ	Basidiomycota	*Russula*	*Russula densifolia*
seqK	Basidiomycota	*Russula*	*Russula densifolia*
seqL	Basidiomycota	*Russula*	*Russula vesca*
seqM	Basidiomycota	*Agaricus*	
seqN	Basidiomycota	*Amanita*	
seqO	Basidiomycota	*Amanita*	
seqP	Ascomycota	*Bisporella*	
seqQ	Ascomycota	*Capronia*	
seqR	Ascomycota	*Capronia*	
seqS	Ascomycota	*Cenococcum*	

表 3.1　同定結果の例 — 空欄は unidentified、つまり不明ということです。横幅を抑えるためいくつかの分類階層は省いてあります。

`classigntax` のデフォルト設定では、全ての近隣既知配列の所属分類群が同一になるまで分類階層を上げていきますので、誤同定された配列が混ざっていたりした場合には下位の分類階層はほとんど不明になってしまいます。これは「ほぼ正しいと思われる」結果を得ることをデフォルト設定では優先しているためですが、近隣既知配列が多い場合、多少不一致を許容して誤同定の可能性が増してでもできるだけ下位の分類階層まで同定したい場合があります。そのような場合には以下のように`--maxpopposer` と`--minsoratio` を指定します。

```
> classigntax \
--taxdb=overall_genus \
--maxpopposer=0.05 \
--minsoratio=19 \
入力ファイル \
出力ファイル↓
```

`classigntax` は結果として採用する分類群に該当する配列を supporter 配列とし、該当しない配列を opposer 配列とします。`--maxpopposer` には opposer 配列の存在を許容する割合を百分率で指定します。`--minsoratio` は supporter 配列数と opposer 配列数の比の下限値を指定します。supporter 配列数と opposer 配列数がこれを下回るような同定結果は許容せず分類階層を上げていきます。2 つのオプションが必要なのは、その分類階層の情報がない配列が存在し得るためです。そのような配列は supporter でも opposer でもないので、2 つのオプションが必要になります。上記の例では、近隣既知配列中、5% までの opposer 配列を許容し、opposer 配列の 19 倍以上の supporter 配列を必要とします。

3.3 複数の同定結果の統合

例えば植物の *rbcL* と *matK* など、複数のバーコード領域を併用しなければ同定できないこともありますし、`overall_genus` と `overall_class` での同定結果のいいとこ取りをしたい場合があります。また、厳密な LCA による同定結果と 5% まで opposer を許容する LCA の同定結果の組み合わせでいいとこ取りしたい場合もあるでしょう。そのようなことが、複数の同定結果を統合することで可能になります。

植物の *rbcL* と *matK* を併用して同定する場合は、より深くまで同定できている方の結果を採用するのがいいでしょう。これは以下のコマンドでできます。

```
> clmergeassign \
--priority=equal \
--preferlower \
rbcL での同定結果 \
matK での同定結果 \
出力ファイル↓
```

より保守的に考えるなら、以下のようにして、両方の同定結果が同一か、一方では未同定の場合だけ採用することもできます。ただし、上の階層で不一致だった場合には、同定結果が一方で未同定でも採用されません。

```
> clmergeassign \
--priority=equal \
rbcL での同定結果 \
matK での同定結果 \
出力ファイル↓
```

trnH–psbA も併用して、最も深くまで同定できているものを採用する場合は以下のようにします。

```
> clmergeassign \
--priority=equal \
--preferlower \
rbcL での同定結果 \
matK での同定結果 \
trnH-psbA での同定結果 \
出力ファイル↓
```

`overall_genus` の結果を優先的に採用し、`overall_genus` で同定できている分類階層までは一致しているが `overall_class` の方がより下位まで同定できている場合だけ採用するには、以下のようにします。

```
> clmergeassign \
--priority=descend \
overall_genus での同定結果 \
overall_class での同定結果 \
出力ファイル↓
```

同様に、厳密な LCA による結果を優先し、厳密な LCA で同定できている分類階層まで一致しているが制約を緩めた LCA による結果の方がより下位まで同定できている場合にのみ採用するには、以下のようにします。

```
> clmergeassign \
--priority=descend \
厳密な LCA での同定結果 \
制約を緩めた LCA での同定結果 \
出力ファイル↓
```

引用文献

Andrews, S., 2010, "Software distributed by the author at http://www.bioinformatics.babraham.ac.uk/projects/fastqc/.".

Bengtsson-Palme, J., Ryberg, M., Hartmann, M., Branco, S., Wang, Z., Godhe, A., De Wit, P., Sánchez-García, M., Ebersberger, I., de Sousa, F., Amend, A., Jumpponen, A., Unterseher, M., Kristiansson, E., Abarenkov, K., Bertrand, Y. J. K., Sanli, K., Eriksson, K. M., Vik, U., Veldre, V., and Nilsson, R. H., 2013, "Improved software detection and extraction of ITS1 and ITS2 from ribosomal ITS sequences of fungi and other eukaryotes for analysis of environmental sequencing data", *Methods in Ecology and Evolution*, **4**, No. 10, 914–919.

Bengtsson, J., Eriksson, K. M., Hartmann, M., Wang, Z., Shenoy, B. D., Grelet, G.-A., Abarenkov, K., Petri, A., Rosenblad, M. A., and Nilsson, R. H., 2011, "Metaxa: a software tool for automated detection and discrimination among ribosomal small subunit (12S/16S/18S) sequences of archaea, bacteria, eukaryotes, mitochondria, and chloroplasts in metagenomes and environmental sequencing datasets.", *Antonie Van Leeuwenhoek*, **100**, No. 3, 471–475, Oct.

Edgar, R. C., Haas, B. J., Clemente, J. C., Quince, C., and Knight, R., 2011, "UCHIME improves sensitivity and speed of chimera detection.", *Bioinformatics*, **27**, No. 16, 2194–2200, Aug.

Fadrosh, D. W., Ma, B., Gajer, P., Sengamalay, N., Ott, S., Brotman, R. M., and Ravel, J., 2014, "An improved dual-indexing approach for multiplexed 16S rRNA gene sequencing on the Illumina MiSeq platform.", *Microbiome*, **2**, No. 1, 6.

Hamady, M., Walker, J. J., Harris, J. K., Gold, N. J., and Knight, R., 2008, "Error-correcting barcoded primers for pyrosequencing hundreds of samples in multiplex.", *Nature Methods*, **5**, No. 3, 235–237, Mar.

Huson, D. H., Auch, A. F., Qi, J., and Schuster, S. C., 2007, "MEGAN analysis of metagenomic data.", *Genome Research*, **17**, No. 3, 377–386, Mar.

Illumina corporation, 2013, "16S metagenomic sequencing library preparation.".

Katoh, K. and Standley, D. M., 2013, "MAFFT multiple sequence alignment software version 7: improvements in performance and usability.", *Molecular Biology and Evolution*, **30**, No. 4, 772–780, Apr.

Kunin, V., Engelbrektson, A., Ochman, H., and Hugenholtz, P., 2010, "Wrinkles in the rare biosphere: pyrosequencing errors can lead to artificial inflation of diversity estimates.", *Environmental Microbiology*, **12**, No. 1, 118–123, Jan.

Lange, A., Jost, S., Heider, D., Bock, C., Budeus, B., Schilling, E., Strittmatter, A., Boenigk, J., and Hoffmann, D., 2015, "AmpliconDuo: A Split-Sample Filtering Protocol for High-Throughput Amplicon Sequencing of Microbial Communities.", *PLoS One*, **10**, No. 11, e0141590.

Larkin, M. A., Blackshields, G., Brown, N. P., Chenna, R., McGettigan, P. A., McWilliam, H., Valentin, F., Wallace, I. M., Wilm, A., Lopez, R., Thompson, J. D., Gibson, T. J., and Higgins, D. G., 2007, "Clustal W and Clustal X version 2.0.", *Bioinformatics*, **23**, No. 21, 2947–2948, Nov.

Li, W., Fu, L., Niu, B., Wu, S., and Wooley, J., 2012, "Ultrafast clustering algorithms for metagenomic sequence

analysis.", *Briefings in Bioinformatics*, **13**, No. 6, 656–668, Nov.

Nelson, M. C., Morrison, H. G., Benjamino, J., Grim, S. L., and Graf, J., 2014, "Analysis, optimization and verification of Illumina-generated 16S rRNA gene amplicon surveys.", *PLoS One*, **9**, No. 4, e94249.

Schmieder, R. and Edwards, R., 2011, "Quality control and preprocessing of metagenomic datasets.", *Bioinformatics*, **27**, No. 6, 863–864, Mar.

Stevens, J. L., Jackson, R. L., and Olson, J. B., 2013, "Slowing PCR ramp speed reduces chimera formation from environmental samples.", *Journal of Microbiological Methods*, **93**, No. 3, 203–205, Jun.

Tanabe, A. S. and Toju, H., 2013, "Two new computational methods for universal DNA barcoding: a benchmark using barcode sequences of bacteria, archaea, animals, fungi, and land plants.", *PLoS One*, **8**, No. 10, e76910.

Zhang, J., Kobert, K., Flouri, T., and Stamatakis, A., 2014, "PEAR: a fast and accurate Illumina Paired-End reAd mergeR.", *Bioinformatics*, **30**, No. 5, 614–620, Mar.

付録 A

その他のプログラム・データベースのインストール

A.1　bcl2fastq のインストール

Illumina から提供されている、BCL 形式のベースコールデータから FASTQ を生成するプログラム bcl2fastq は MiSeq・HiSeq 用の v1.8.4 および NextSeq 500・HiSeq X 用の v2.17.1.14 が
http://support.illumina.com/downloads/bcl2fastq_conversion_software.html
からダウンロードできます。v1.8.4 は以下のようにインストールできます。

```
# alien のインストール
$ sudo apt-get install alien
# bcl2fastq v1.8.4 のダウンロード
$ wget \
ftp://webdata:webdata@ussd-ftp.illumina.com/Downloads/Software/bcl2fastq/bcl2fastq-1.8.4-Linux-x86_64.rpm
# .rpm から.deb への変換とインストール
$ sudo alien -i \
bcl2fastq-1.8.4-Linux-x86_64.rpm
# bcl2fastq に必要な追加パッケージのインストール
$ sudo apt-get install libxml-simple-perl xsltproc
```

Ubuntu・Xubuntu・Lubuntu の 14.04LTS では、Perl のバージョンが新しすぎ、より古いバージョンを前提として書かれた bcl2fastq が動作しません。そこで以下のようにして問題の部分を書き換えます。

```
$ sudo perl -i -npe \
's/qw\(ELAND_FASTQ_FILES_PER_PROCESS\)/\("ELAND_FASTQ_FILES_PER_PROCESS"\)/' \
/usr/local/lib/bcl2fastq-1.8.4/perl/Casava/Alignment/Config.pm
$ sudo perl -i -npe \
's/qw\(ELAND_GENOME\)/\("ELAND_GENOME"\)/' \
/usr/local/lib/bcl2fastq-1.8.4/perl/Casava/Alignment/Config.pm
```

テキストエディタで
/usr/local/lib/bcl2fastq-1.8.4/perl/Casava/Alignment/Config.pm

の 747 行目と 751 行目を手動で書き換えても構いません。qw(HOGEHOGE) を ("HOGEHOGE") に書き換えて下さい。

v2.17.1.14 のインストールをする場合は以下のようにコマンドを実行して下さい。

```
# rpm2cpio と cpio のインストール
> sudo apt-get install rpm2cpio cpio ↓
# bcl2fastq2 v2.17 のダウンロード
> wget \
ftp://webdata2:webdata2@ussd-ftp.illumina.com/downloads/software/bcl2fastq/bcl2fastq2-v2.17.1.14-Linux-x86_64.zip
↓
# 圧縮ファイルの展開
> unzip -qq bcl2fastq2-v2.17.1.14-Linux-x86_64.zip ↓
# .rpm を展開してコマンドを抽出
> rpm2cpio \
bcl2fastq2-v2.17.1.14-Linux-x86_64.rpm | cpio -id ↓
# コマンドをインストール
> sudo mv usr/local/bin/bcl2fastq /usr/local/bin/↓
```

付録 B

ターミナルコマンド集

B.1　配列を数え上げる

```
# FASTQ の場合
> grep -P -c '^\+$' inputfile ↓
# GZIP 圧縮 FASTQ の場合
> gzip -dc inputfile | grep -P -c '^\+$' ↓
# BZIP2 圧縮 FASTQ の場合
> bzip2 -dc inputfile | grep -P -c '^\+$' ↓
# XZ 圧縮 FASTQ の場合
> xz -dc inputfile | grep -P -c '^\+$' ↓
# FASTA の場合
> grep -P -c '^>' inputfile ↓
# GZIP 圧縮 FASTA の場合
> gzip -dc inputfile | grep -P -c '^>' ↓
# BZIP2 圧縮 FASTA の場合
> bzip2 -dc inputfile | grep -P -c '^>' ↓
# XZ 圧縮 FASTA の場合
> xz -dc inputfile | grep -P -c '^>' ↓
```

B.2　配列を閲覧する

```
# 無圧縮ファイル内容を画面に出力
> cat inputfile ↓
# 無圧縮ファイル内容をスクロールしながら見る
> less inputfile ↓
# GZIP 圧縮ファイル内容を画面に出力
> gzip -dc inputfile ↓
# GZIP 圧縮ファイル内容をスクロールしながら見る
> gzip -dc inputfile | less ↓
# FASTQ の最初の 1 配列を画面に出力
> head -n 4 inputfile ↓
# GZIP 圧縮 FASTQ の最初の 1 配列を画面に出力
> gzip -dc inputfile | head -n 4 ↓
# FASTQ の特定の配列を検索して画面に出力
> grep -P -A 3 '^\@配列名の正規表現' inputfile ↓
# GZIP 圧縮 FASTQ の特定の配列を検索して画面に出力
```

```
> gzip -dc inputfile | grep -P -A 3 '^\@配列名の正規表現' ↓
```

B.3 圧縮・展開

```
# フォルダ内の全.fastq.gz を展開
> for f in *.fastq.gz ↓
do gzip -d $f ↓
done ↓
# フォルダ内の全.fastq.gz を展開 (サブフォルダ含む)
> for f in `find .  -name *.fastq.gz` ↓
do gzip -d $f ↓
done ↓
# 4 つの CPU を使用してフォルダ内の全.fastq.gz を展開
> ls *.fastq.gz | xargs -L 1 -P 4 gzip -d ↓
# 4 つの CPU を使用してフォルダ内の全.fastq.gz を展開 (サブフォルダ含む)
> find .  -name *.fastq.gz | xargs -L 1 -P 4 gzip -d ↓
# フォルダ内の全.fastq を圧縮
> for f in *.fastq ↓
do gzip $f ↓
done ↓
# フォルダ内の全.fastq を圧縮 (サブフォルダ含む)
> for f in `find .  -name *.fastq` ↓
do gzip $f ↓
done ↓
# 4 つの CPU を使用してフォルダ内の全.fastq を圧縮
> ls *.fastq | xargs -L 1 -P 4 gzip ↓
# 4 つの CPU を使用してフォルダ内の全.fastq を圧縮 (サブフォルダ含む)
> find .  -name *.fastq | xargs -L 1 -P 4 gzip ↓
```

B.4 抽出・保存

```
# FASTQ から最初の 1 万配列を抽出
> head -n 40000 inputfile > outputfile ↓
# FASTQ から最後の 1 万配列を抽出
> tail -n 40000 inputfile > outputfile ↓
# GZIP 圧縮 FASTQ から最初の 1 万配列を抽出して GZIP 圧縮 FASTQ へ保存
> gzip -dc inputfile | head -n 40000 | gzip -c > outputfile ↓
# GZIP 圧縮 FASTQ から最後の 1 万配列を抽出して GZIP 圧縮 FASTQ へ保存
> gzip -dc inputfile | tail -n 40000 | gzip -c > outputfile ↓
# GZIP 圧縮 FASTQ の特定の配列を検索して GZIP 圧縮 FASTQ へ保存
> gzip -dc inputfile | grep -P -A 3 '^\@配列名の正規表現' | gzip -c > outputfile ↓
```