Bioperl1.4 チュートリアル(http://www.bioperl.org/Core/Latest/bptutorial.html)の和訳

(訳注:訳し終わったところの目次は和訳してあります)

NAME

BioPerlTutorial - a tutorial for bioperl

VERSION

1.4

AUTHOR

Written by Peter Schattner <schattner@alum.mit.edu>

Contributions, additions and corrections have been made
to this document by the following individuals:

Jason Stajich
Heikki Lehvaslaiho
Brian Osborne
Hilmar Lapp
Chris Dagdigian
Elia Stupka
Ewan Birney

概要

モジュールのドキュメント、スクリプトの例、"t"テストスクリプトのドキュメントを含む bioperl ドキュメントからのコードと文章の「断片」がこのチュートリアルには含まれています。perl 自身と同じ条件の下でユーザーはことチュートリアルを配布できます。

このドキュメントは perl POD(plain old documentation)フォーマットで書かれています。より使いやすいフォーマットにしたいのであれば pod 形式変換ツール(pod2html、pod2man、pod2text、など)でフォーマットを換えることができます。
(訳注:翻訳後は必ずしも POD に則っておりません)

目次

LargeSeq, RichSeq, SeqWithQuality, SeqI)

- o II.2 Location オブジェクト
- o II.4 インターフェースオブジェクトと実装オブジェクト

* III. bioperl 使用法

- o III.1 ローカルとリモートのデータベース上の配列データへのアクセス
- o III.1.1 リモートデータベースへのアクセス(Bio::DB::GenBank, など)
- o III.1.2 ローカルデータベース (Bio::Index::*, bp_index.pl, bp_fetch.pl, Bio::DB::*)のインデックス化とアクセス
- o III.2 Transforming formats of database/ file records
- o III.2.1 Transforming sequence files (SeqIO)
- o III.2.2 Transforming alignment files (AlignIO)
- o III.3 Manipulating sequences
- o III.3.1 Manipulating sequence data with Seq methods
- o III.3.2 Obtaining basic sequence statistics (SeqStats,SeqWord)
- o III.3.3 Identifying restriction enzyme sites (Bio::Restriction)
- o III.3.4 Identifying amino acid cleavage sites (Sigcleave)
- o III.3.5 Miscellaneous sequence utilities: OddCodes, SeqPattern
- o III.3.6 Converting coordinate systems (Coordinate::Pair, RelSegment)
- o III.4 類似配列の検索
- o III.4.1 BLAST の実行(RemoteBlast.pm を用いて)
- o III.4.2 Parsing BLAST and FASTA reports with Search and SearchIO
- o III.4.3 Parsing BLAST reports with BPlite, BPpsilite, and BPbl2seq
- o III.4.4 Parsing HMM reports (HMMER::Results, SearchIO)
- o III.4.5 Running BLAST locally (StandAloneBlast)
- o III.5 Manipulating sequence alignments (SimpleAlign)
- o III.6 Searching for genes and other structures on genomic DNA (Genscan, Sim4, Grail, Genemark, ESTScan, MZEF, EPCR)
- o III.7 Developing machine readable sequence annotations
- o III.7.1 Representing sequence annotations (SeqFeature,RichSeq,Location)
- o III.7.2 Representing sequence annotations (Annotation::Collection)
- o III.7.3 Representing large sequences (LargeSeq)
- o III.7.4 Representing changing sequences (LiveSeq)
- o III.7.5 Representing related sequences - mutations, polymorphisms (Allele, SeqDiff)
- o III.7.6 Incorporating quality data in sequence annotation (SeqWithQuality)
- o III.7.7 Sequence XML representations - generation and parsing (SeqIO::game, SeqIO::bsml)
- o III.7.8 Representing Sequences using GFF (Bio:DB:GFF)
- o III.8 Manipulating clusters of sequences (Cluster, ClusterIO)
- o III.9 Representing non-sequence data in Bioperl: structures, trees and maps
- o III.9.1 Using 3D structure objects and reading PDB files (StructureI, Structure::IO)
- o III.9.2 Tree objects and phylogenetic trees (Tree::Tree, TreeIO, PAML)
- o III.9.3 Map objects for manipulating genetic maps (Map::MapI, MapIO)
- o III.9.4 Bibliographic objects for querying bibliographic databases (Biblio)
- o III.9.5 Graphics objects for representing sequence objects as images (Graphics)
- o III.10 Bioperl alphabets

I．イントロダクション

I.1 概要

bioperl は、バイオインフォマティクスのアプリケーションとして perl スクリプトで開発された perl のモジュール群です。ですから、多くの市販のパッケージや EntreZ、SRS などのフリーの WEB インターフェースを使うような感じのプログラムではありません。一方、bioperl は配列の操作、データフォーマットを毎の DB へのアクセス、BLAST、clustalw、TCoffee、GenScan、ESTscan、HMMER のような分子生物学のプログラムの結果を実行、切り出しのプログラムなどが再利用が可能な perl モジュールを提供します。つまり、Web ベースのシステムでは普通困難か不可能な多量の配列データの解析を可能とするスクリプトを開発することが perl でできます。

bioperl の利点を用いるために、perl の参照、モジュール、オブジェクト、メソッドの使い方を含む perl 言語の基本的な理解がユーザーに必要です。もしこれらに慣れていなければ perl の初中級の本をどれか参照してください。たとえば、S． Holzmer 著「Perl Core Language」Coriolis Technology Press を薦めます。このチュートリアルでは perl 言語の入門者、未経験者に perl の基本を教えることを目的としていません。一方、objext-oriented perl モジュールをどのように書くかというような perl の応用知識も bioperl を使う上において要求していません。

bioperl はオープンソースソフトウェアであり、現在活発に開発中です。オープンソースソフトウェアの利点については周知のところです。自由にテストができ、ソースコードの改変が可能であり、ソフトウェアの使用量が不要です。しかし、オープンソースソフトウェアなので、普通開発は多くのボランティアのプログラマーによってなされているため、コードは必ずしもまとめられておらず、よく出来た市販の製品ほどにはユーザーインターフェースが標準化されてません。加えて、活発に開発中のプロジェクトでは新規開発分にドキュメントが追いついてないものがあります。つまり、活発な開発に対して追いつく学習曲線はしばしば急勾配です。

このチュートリアルは新規の bioperl ユーザに対して、その学習曲線をなだらかにすることを目的としています。最終的にはこのチュートリアルには下記のものを含みます、

＊bioperl により、バイオインフォマティシャンの仕事をどう扱うことが出来るかを記述しています。

＊その目的を達成するメソッドが bioperl パッケージのどこにあるかを記述しています。

＊補足情報をどこで得ればよいかを示しています。

＊bioperl の能力の多くをデモンストレーションする実行可能なスクリプト、bptutorial.pl があります。実行可能なコード例はたいてい scripts/ フォルダ内か examples/ フォルダ内にあります。これらスクリプトの要約は bioscripts.pod ファイル内（あるいは http://bioperl.org/Core/Latest/bioscripts.html ）にあります。加えて、多くの bioperl モジュールの POD 文書には SYNOPSIS セクション内に実行可能なコードがあ

り、モジュールとメソッドの使い方を示しています。また、FAQ
(http://bioperl.org/Core/Latest/faq.html)内にもいくつかの面白いコード例があります。

このチュートリアルを読み通す間あるいはその前に対話型のデバッガーを使いながら bptutorial.pl スクリプトを実行することにより、bioperl をより理解しやすくなるでしょう。チュートリアルのスクリプトは(むしろこのチュートリアル内のコード断片を使うことよりも)あなたのスクリプト内にカット＆ペーストしやすくしてあります。このチュートリアル内のほとんどのスクリプトはあなたのマシンで動くはずです。もし動かなければ、bioperl に深く関わる前にそれがなぜ動かないかを発見するよい練習となるでしょう！いくつかのデモンストレーションは補助のライブラリあるいは外部のプログラムなどの補足的なモジュールを必要とします。もし必要となる補助のライブラリかプログラムが見つからなければこれらのデモンストレーションは飛ばしてもかまいません。

I.2 クイックスタート

このパッケージを使うことが一番よいかどうかを確かめたい方のために、やさしい手順で bioperl のいくつかの機能を確かめる非常に簡単なモジュールがあります。Bio::Perl モジュールはいくつかの簡単なアクセス機能を提供します。たとえば、以下のスクリプトは swissprot の配列を読み込み、FASTA 形式に書き出します。

```
use Bio::Perl;

# this script will only work with an internet connection
# on the computer it is run on
$seq_object = get_sequence('swissprot',"ROA1_HUMAN");

write_sequence(">roa1.fasta",'fasta',$seq_object);
```

他の例として、NCBI の機能を使い BLAST を実行します。NCBI が提供する BLAST を邪魔しないように個々の検索だけに用いてください(訳注:このスクリプトは NCBI の BLAST 機能に処理を投げているため、ユーザーから多量の検索を実行すると NCBI の BLAST サーバーに負荷がかかるため)。もし多くの BLAST 検索を行いたい場合は BLAST パッケージをローカルにダウンロードしてください。

```
use Bio::Perl;

# this script will only work with an internet connection
# on the computer it is run on

$seq = get_sequence('swissprot',"ROA1_HUMAN");

# uses the default database - nr in this case
$blast_result = blast_sequence($seq);

write_blast(">roa1.blast",$blast_result);
```

他にも Bio::Perl には以下のような使いやすい機能がたくさんあります。

```
get_sequence          - 標準入力、インターネット接続可能なデータベースから配列を取得します
             gets a sequence from standard, internet accessible
            databases
read_sequence         - ファイルから配列を読み込みます
read_all_sequences    - ファイルからすべての配列を読み込みます
new_sequence          - 文字列から bioperl 用配列を作成します
write_sequence        - 1つの配列(sequence)あるいは配列(sequence)の配列(array)をファイルに書き出します
translate             - 配列の翻訳を行います
```

```
translate_as_string - 配列の翻訳を行い、配列を文字列として返します
blast_sequence      - NCBI の標準データベースに対して BLAST を実行します
write_blast         - BLAST 結果をファイルに書き出します
```

Bio::Perl.pm モジュールを使うことにより、以下で述べる Seq や SeqIO オブジェクトを明示的に作らずに bioperl 上で配列操作をすることが可能です。しかし、このモードでは操作は限られたものとなります。

これらの機能についてより知りたい場合は'perldoc Bio::Perl'を実行して Bio::Perl のマニュアルページを参照してください。これらすべてのケースでは、Bio::Perl は bioperl 機能（たとえば bioperl での翻訳では多くの異なる翻訳テーブルを扱い、終止コドンの取扱のための複数のオプションを扱うことが可能です）のサブセットにアクセスします。多くのケースではユーザーはより高度な bioperl オブジェクトを用いるようになるでしょうが、新人あるいは怠惰なプログラマーのために優しい導きの明かりを Bio::Perl は灯しています。また、このモジュールの使い方のより多くの例を示した examples/bioperl.pl を参照してください。

I.3 ソフトウエアに必要なもの

bioperl の実行に必要なものは下記の通りです。

I.3.1 bioperl の最小インストール（bioperl の「コア」インストール）

bioperl の最小インストールのために、bioperl の「コアモジュール」と同様に perl 自体のインストールが必要です。bioperl は perl 5.005、5.6、5.8 で主にテストされています。bioperl の最小インストールでは perl 5.004 でも動くはずです。しかし、CPAN（下記参照）のモジュールを使っている bioperl オブジェクトが増えているので、perl 5.004 環境での bioperl の実行は問題が発生しています。perl 5.004 環境で bioperl を実行し問題があるのなら、perl のバージョンアップを行ってください。

perl のカレントバージョンではくわえて、対話型のデバッガーにアクセスし精通することが初心者には推奨されます。bioperl は複雑な対話型ソフトウエアオブジェクトの大きな集まりです。複雑なソフトウエアシステムが思わぬ動きをするといったようなハプニングが起きたときに、対話型デバッガーを使ってスクリプトを作ることはよい助けとなります。
CPAN の Devel::ptkdb として使用できる、フリーのグラフィカルなデバッガーである ptkdb を特にお勧めします。標準の perl ではコマンドラインインターフェースの強力な対話型デバッガーがあります（"perl -d <script>" として使用できます）。

perl ツールの Data::Dumper は以下の構文で用い、

```
use Data::Dumper;
print Dumper($seqobj);
```

bioperl オブジェクトのデバッグ情報を得るために役だちます。

I.3.2 完全なインストール

bioperl の機能のいくつかは、最小インストールを超えたものが必要です。追加のソフトウエアとしては、CPAN の perl モジュール、bioperl の補助コードレポジトリのパッケージライブラリ、bioperl の xs 拡張、そして標準的なバイオインフォマティクスプログラムが必要です。
Some of the capabilities of bioperl require software beyond that of the minimal installation. This additional software includes perl modules from CPAN, package-libraries from bioperl's auxiliary code-repositories, a bioperl xs-extension, and several standard compiled bioinformatics programs.

Perl - 拡張
Perl - extensions

bioperl で必要とする perl の拡張モジュールの完全なリストについては bioperl パッケージの INSTALL ファイル（あるいは http://bioperl.org/Core/Latest/INSTALL ）を参照してください。

bioperl の補助レポジトリ

bioperl のいくつかの機能では、bioperl の補助コードレポジトリのモジュールを必要とします。これらのモジュールに関するインストールについては第 IV 節とリファレンスを参照してください。

bioperl の C 言語拡張と外部のバイオインフォマティクスプログラム
Bioperl C extensions & external bioinformatics programs

配列アライメントやローカルでの BLAST 検索に bioperl ではいくつかの C 言語プログラムも用います。bioperl のこれらの機能を使うためには、以下のような実際のプログラムソースとともに、ANSI C あるいは GNU C コンパイラが必要です。

Smith-Waterman アライメントでは bioperl-ext-0.6 http://bioperl.org/Core/external.shtml

clustalW アライメントでは ftp://ftp.ebi.ac.uk/pub/software/unix/clustalw/ ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalW/

Tcoffee アライメントでは http://igs-server.cnrs-mrs.fr/~cnotred/Projects_home_page/t_coffee_home_page.html

ローカル BLAST 検索では ftp://ftp.ncbi.nih.gov/blast/executables/release/

EMBOSS アプリケーションでは http://www.emboss.org

I.4 インストール

様々なコンポーネントの実際のインストールでは標準的な手段で完了します。
The actual installation of the various system components is accomplished in the standard manner:

* ネットワーク上でパッケージを見つける
* ダウンロード
* （gunzip あるいはその他解凍ソフトでの）ファイルの解凍
* アーカイブファイルの展開（例: tar -xvf）
* Makefile の作成 "perl Makefile.PL"
* コマンド実行。"make"、"make test"、"make install"。この手順は CPAN モジュール、bioperl-extension、外部モジュールのインストールの度に毎回繰り返されます。お助けモジュールである CPAN.pm は perl モジュールを CPAN からインストールするための手順を自動化してくれます。

上記のモジュールのすべてについて CPAN モジュールを用いると、Bundle::BioPerl モジュールを用いた"bundle"コマンドでインストールが可能です。例えば、

The CPAN module can also be used to install all of the modules listed above in a single step as a ``bundle'' of modules, Bundle::BioPerl, eg

```
$>perl -MCPAN -e shell
cpan>install Bundle::BioPerl
```

```
<installation details....>
cpan>install B/BI/BIRNEY/bioperl-1.2.2.tar.gz
<installation details....>
cpan>quit
```

バージョンの番号が変わると上記は適用できないので注意して下さい。"bundle"のよくないところは、インストールに問題が発生した場合どこが問題かわかりにくいことです。

```
   Be advised that version numbers change regularly, so the number used above
may not apply. A disadvantage of the ``bundle'' approach is that if there's a
problem installing any individual module it may be a bit more difficult to
isolate.
```

詳細はbioperlのINSTALLファイル(あるいはhttp://bioperl.org/Core/Latest/INSTALL)を参照してください。

外部プログラム(clustal、Tcoffee、ncbi-blast)では別のステップです。

* 実行するファイルがあるディレクトリを示す適切な環境変数(CLUSTALDIR、TCOFFEEDIR、BLASTDIR)をたとえば(.bashrc、.tcshrc)のような初期設定ファイルに設定してください。ローカルBLASTを実行するためには、bioperlが認識できるようにローカルBLASTのデータベースディレクトリも設定する必要があります。これはたいてい自動的に行われますが、問題が発生した場合はBio::Tools::Run::StandAloneBlastのmanpageを参照してください。

(少なくともUNIXにおいては)発生するであろう問題としては、ファイルの書き込み権限の問題があります。インストールを修正したり、上書きで再インストールする場合、bioperlの配布物にあるINSTALLファイル(あるいはhttp://bioperl.org/Core/Latest/INSTALL)と、使用する外部プログラムのREADMEファイルを参照してください。

```
The only likely complication (at least on unix systems) that may occur is if you
are unable to obtain system level writing privileges. For instructions on
modifying the installation in this case and for more details on the overall
installation procedure, see the INSTALL file (or http://bioperl.org/Core/Latest/
INSTALL ) in the bioperl distribution as well as the README files for the
external programs you want to use.
```

I.5 非UNIXユーザーのための付記

bioperlは主にLinuxとMacOS Xを含む、UNIX環境で開発されテストされています。くわえて、このチュートリアルはUNIXの観点で書かれています。

bioperlのコア部分はMicrosoft Windowsの多くのバージョン下でのテストされ、動作するはずです。多くのWindowsユーザーはperlとbioperl配布物としてActive State、http://www.activestate.comを用いておりとても便利です。他のWindowsユーザーはCygwin(http://www.cygwin.com)環境下でbioperlを実行しています。詳細についてはパッケージ内のINSTALL.WINファイル(あるいはhttp://bioperl.org/Core/Latest/INSTALL.WIN)を参照のこと。

多くのbioperlの機能はCPANモジュール、拡張コンパイルあるいは外部プログラムを要求します。これらの機能は恐らくいくつかのあるいは他のOS上では動かないかもしれません。もし、非UNIXOS上でこれらの機能にアクセスしようと試みるのであれば、期待する機能が使えない簡単なレポートをするようにbioperlをデザインしてください。しかし、これら非UNIX環境でのbioperlのテストは限られているので、手段を誤るとスクリプトが暴走するかもしれません。

```
Many bioperl features require the use of CPAN modules, compiled extensions or
external programs. These features probably will not work under some or all of
these other operating systems. If a script attempts to access these features
from a non-unix OS, bioperl is designed to simply report that the desired
capability is not available. However, since the testing of bioperl in these
```

environments has been limited, the script may well crash in a less graceful manner.

I.6 補足のドキュメントの場所

このチュートリアルは bioperl のすべてのオブジェクトとメソッドについて書かれた物ではありません。その場合はモジュール毎に含まれるドキュメントを直接読んでください。すべてのモジュールのドキュメントについて探すための使いやすいインターフェースとして http://doc.bioperl.org/bioperl-live/ があります。このインターフェースのリストは bioperl のすべてのモジュールとメソッドについて記述しています。くわえて、初心者がよくする質問については、bioperl の配布物の一番上のディレクトリ内にある FAQ、INSTALL、README ファイル（http://bioperl.org/Core/Latest/faq.html、http://bioperl.org/Core/Latest/INSTALL、http://bioperl.org/Core/Latest/README）を見てください。

異なるモジュールにある複数のメソッドがすべて同じ名前であるということが、正しいドキュメントでありながら問題となりそうな点としてあげられます。くわえて、継承したメソッドが perl では複雑なため、オブジェクトの名前で呼ばれるメソッドの名前がしばしば明白でないことがあります。この問題を解決する手段の一つは付録 V.1 のようにソフトウエアを使うことです。

One potential problem in locating the correct documentation is that multiple methods in different modules may all share the same name. Moreover, because of perl's complex method of inheritance it is not often clear which of the identically named methods is being called by a given object. One way to resolve this question is by using the software described in Appendix V.1.

もしヴィジュアルな記述がよいのであれば、http://bioperl.org/Core/Latest/modules.html からリンクされている、bioperl オブジェクトの相互関係のクラス図（クラス図第 1.0 版）である PDF ファイルも参照してください。

くわえて、bioperl の WEB 上のオンラインコースが http://www.pasteur.fr/recherche/unites/sis/formation/bioperl で利用できます。また、scripts/、examples/ディレクトリ（bioscripts.pod あるいは http://bioperl.org/Core/Latest/bioscripts.html 参照）内にある多くの bioperl スクリプトも参照できます。OBDA アクセス、SeqIO、SearchIO、BioGraphics、Features と Annotation、Trees、PAML そして Biopipe についてのトピックが書かれています。

II. bioperl オブジェクトの概略

実際のバイオインフォマティクスの問題を解決するために bioperl をできるだけ早く使えるようにすることがこのチュートリアルの目的です。bioperl オブジェクト、perl オブジェクトの一般的なプログラミングの構造を説明するものではありません。実際、bioperl オブジェクト間の関係は簡単ではありませんが、パッケージを用いるためには bioperl オブジェクトの詳細を知ることは必須ではありません。

ですが bioperl オブジェクトにある程度精通することは、bioperl を使いこなす上で非常に役立つに違いありません。たとえば少なくとも 8 個の「配列オブジェクト」（Seq, PrimarySeq, LocatableSeq, RelSegment, LiveSeq, LargeSeq, SeqI, SeqWithQuality）があります。これらのオブジェクトの関係（そしてそれらがなぜこれだけあるか）を理解することにより、自分のスクリプトにどのオブジェクトを使うのがよいかがわかるようになるでしょう。

II.1 配列オブジェクト（Seq, PrimarySeq, LocatableSeq, RelSegment, LiveSeq, LargeSeq, RichSeq, SeqWithQuality, SeqI）

この節では bioperl の多様な配列オブジェクトについて述べています。bioperl を使う多くの人は、使っているオブジェクトがどんな種類かについて知らず、知る必要がないでしょう。なぜならファイル、ファイルハンドル、文字列が与えられたときに、正しいタイプのオブジェクトを III.2.1 節で述べる SeqIO モージュールが作成するからです。しかしもし気になるなら、訳あって配列オブジェクトを作る必要があるなら以下を読んでください。

Seq はbioperlの中心的な配列オブジェクトです。どれを使ったらいいかわからないとき、bioperlで DNA、RNA、タンパク質配列を記述するのに使いたいのはたぶんこのオブジェクトです。たいていの配列操作は Seqにて行えます。このオブジェクトの機能については III.3.1 節と III.7.1 節あるいは Bio::Seq manpageに記述されています。

SeqIOオブジェクトを使った配列データを含むファイルを読むときに Seqオブジェクトは自動的に作成されます。この手順は III.2.1 に記述されています。加えて、識別ラベルか配列自身を溜めるとき、Seqオブジェクトはマルチプルアノテーションと、Genbank や EMBL 配列ファイルのような関連する"sequence features"を溜めることができます。この機能は、特に自動アノテーションシステムの開発においてとても便利です。III.7.1 参照。

一方、一度に数百あるいは数千の配列を同時に取り扱うスクリプトが必要な場合、個々の配列にアノテーション付けするオーバーヘッドは重大になります。そんな場合は PrimarySeqオブジェクトを使いたくなるでしょう。 PrimarySeq は Seqの余分なものを廃した版です。配列データ自身といくらかの識別ラベル(ID、アクセッション番号、dna、rna あるいは protein というアルファベット)のみを含み、features を含みません。数百あるいは数千の配列を扱う場合、PrimarySeqオブジェクトはプログラムの実行をスピードアップし、メモリの使用量を減らします。詳細は Bio::PrimarySeq の manpage をご覧ください。

RichSeqオブジェクトは標準的な Seqオブジェクトよりも多くのアノテーション情報を溜めます。もし豊富な配列アノテーションのあるデータを使いたいならば III.7.1 節に記述されているこのオブジェクトを使うようになるでしょう。 GenBank、EMBL、SwissProt フォーマットのファイルを SeqIO で読み込んだときに自動的に RichSeqオブジェクトは作成されます。

phred の出力のような質的データを伴った配列を扱うときに SeqWithQualityオブジェクトは使われます。これらのオブジェクトは III.7.6 節や、Bio::Seq::RichSeqI manpage や、Bio::Seq::SeqWithQuality manpage に記述があります。

LocatableSeqオブジェクトは歴史的理由により"AlignedSeq"オブジェクトと呼ばれています。マルチプル配列アライメントの一部であるのは Seqオブジェクトです。抽出された長い配列での開始と終始位置を示します。その配列が属しているアライメントに対応したギャップの印も持ちます。アライメントオブジェクトである SimpleAlign と SimpleAlignオブジェクト(たとえば AlignIO.pm、pSW.pm)を用いる他のモージュールによってそれは使われます。

たいていは LocatableSeqオブジェクトをどう作るか心配することはありません。なぜなら (pSW、Clustalw、Tcoffee、Lagan、bl2seqを用いた)アライメントを作成するかあるいは、AlignIOを用いてアライメントデータファイルを読み込んだときに自動的に生成されるからです。しかし手動で(たとえば SimpleAlignオブジェクトを生成するとき)配列アライメントを読み込む必要があるときは、LocatableSeqとして配列を読み込む必要があるでしょう。他の情報源として、Bio::LocatableSeq manpage、Bio::SimpleAlign manpage、Bio::AlignIO manpage、Bio::Tools::pSW manpage があります。

RelSegment オブジェクトも bioperl の Seqオブジェクトのひとつです。RelSegment オブジェクトはゲノムの座標系を扱うときに便利です。染色体やコンティグのような非常に長い配列に位置するサブシーケンス(たとえばエクソン)を探すときのような状況のことです。たとえば、グラフィカルなゲノムブラウザーを設計するときなど、このような操作は重要です。そのような機能をコーディングする必要があるとき、詳述されている Bio::DB::GFF::RelSegment manpage を参照してください。

LargeSeqオブジェクトは非常に長い配列(たとえば 100MB 超)を扱うときに使われる特別な Seqオブジェクトです。このような長い配列を扱う必要がある場合、LargeSeqオブジェクトについて記述されている III.7.3 節か Bio::Seq::LargeSeq manpage を参照してください。

LiveSeqオブジェクトは配列データを溜めるためのまた別の特別のオブジェクトです。LiveSeq は長期間に配列の位置が変わるという問題を扱います。たとえば、より高精度な配列データが得られ、変更した新しいゲノム配列上

の遺伝子の位置を溜めるために配列 feature オブジェクトが使われます。LiveSeq オブジェクトは Seq オブジェクトと同じようには扱われず、SeqI インターフェースを実装します（下記参照）。したがって、Seq オブジェクトで使えるほとんどのメソッドは LiveSeq オブジェクトでも扱えるでしょう。III.7.4 節と Bio::LiveSeq manpage では LiveSeq オブジェクトについてより詳しい内容が記述されています。

SeqI オブジェクトは Seq の「インターフェースオブジェクト」（II.4 節と Bio::SeqI manpage 参照）です。SeqI オブジェクトは他のソフトウエアパッケージとの bioperl の互換性を保証しています。SeqI と他のインターフェースオブジェクトはカジュアルな bioperl ユーザーには関係がないかもしれません。

II.2 Location オブジェクト

長い配列の feature を見せるための Sequence Feature オブジェクトに関連して Location オブジェクトは設計されています。位置を記述するために使われるスタンドアロンのオブジェクトとしても Location オブジェクトは使われます。このような簡単なコンセプトから複雑なオブジェクトの集合体に進化した理由は以下の通りです。

1）いくつかのオブジェクトは複数の位置あるいはサブ位置（たとえば遺伝子のエクソンが複数の開始、終始位置を持つこと）を持つこと。2）不完全なゲノムにおいて feature の明白な位置が確実にはわからないこと。

bioperl のいろいろな Location オブジェクトは複雑な問題を解決します。くわえて、もし開始と終始位置が明らかになっていない場合 feature の長さをどのように測るかについて使える CoordinatePolicy オブジェクトがあります。たいていの場合、もし開始と終始位置がはっきりしている単純 afeature を bioperl で扱う場合ではこのような複雑な問題について考える必要がないでしょう。しかし、もしゲノムの一部あるいは不完全なゲノムについて bioperl でアノテーションするかそのようなゲノムのアノテーションを bioperl で読み込む場合にはいろいろな Location オブジェクトを理解することが重要になるでしょう。詳しくは Bio::Locations ディレクトリ配下のいろいろなモジュールのドキュメントや Bio::Location::CoordinatePolicyI manpage あるいは III.7.1 節参照のこと。

II.4 インターフェースオブジェクトと実装オブジェクト

インターフェースと実装オブジェクトを分けることが bioperl の設計のひとつの目標です。それがどのように実装されるかという知識なしに、オブジェクトとして呼び出されるメソッドをインターフェースのみで定義します。実行はオブジェクトの実装として働きます。Java のような言語ではインターフェースは言語の一部として定義されます。perl では自身で演じなければいけません。In Perl, you have to roll your own.

bioperl ではインターフェースオブジェクトはたいてい、インターフェースオブジェクトを示す末尾に I を伴った、Bio::MyObjectI と呼ばれています。（いくつかの例外を除いて）実装についての記述なしに、主にインターフェースが何か、どのように使うかについてのドキュメントをインターフェースオブジェクトは提供します。カジュアルな bioperl ユーザーにとってはインターフェースオブジェクトは実用ではありませんが、Ensembl や biopython や biojava のような他のバイオインフォマティクスプロジェクトやプログラム言語と bioperl プログラムがどのようにやり取り可能かについての基本的な理解をするためにはインターフェースオブジェクトの存在を知っておくと便利です。

設計と開発に関してのこれ以上の議論については bioperl パッケージ内の biodesign.pod ファイルか biodesign.html（http://bioperl.org/Core/Latest/biodesign.html）を参照のこと。

III. bioperl 使用法

バイオインフォマティクスプログラミングでの典型的な処理の多くについてのソフトウエアモジュールを bioperl は提供します。例えば以下のような、

* ローカルとリモートのデータベース上の配列データへのアクセス
* データベース、ファイルのフォーマット変換
* 配列の操作
* 配列の類似性検索
* 配列アライメントの作成と操作
* ゲノム DNA 上の遺伝子と配列構造の発見
* わかりやすい配列アノテーション装置の開発

これらの操作のすべてについて bioperl がどう役立つかを以下の節で述べています。

III.1 ローカルとリモートのデータベース上の配列データへのアクセス

bioperl の多くは配列操作に焦点を当てています。しかし bioperl で配列を扱う前に配列データへアクセスすることが必要です。bioperl の Seq オブジェクトに配列データをダイレクトに入力できます。例えば、

```
$seq = Bio::Seq->new(-seq              => 'actgtggcgtcaact',
           -desc             => 'Sample Bio::Seq object',
           -display_id       => 'something',
           -accession_number => 'accnum',
           -alphabet         => 'dna' );
```

しかし多くの場合、オンライン上のデータファイルやデータベース上の配列データにアクセスすることが望ましいです。「インデックス付けされたフラットファイル」という意味に近い言葉として「データベース」と言う言葉をしばしばバイオインフォマティクスでの基本的な用例として我々は用いることに注意してください。

ローカルデータベースにアクセスするためのインデックスを作成するのと同じようにリモートのデータベースのアクセスを bioperl はサポートします。データベースアクセスをするには二つの一般的な方法があります。どんなタイプの配列データベース（たとえば、flat ファイルか、ローカルのリレーションデータベースか、インターネット経由のリモートのデータベースか）にアクセスするか既知ならばそのデータベース固有のスクリプトを書くことができます。この方法は III.1.1 節と III.1.2 節で、それぞれリモートデータベースとローカルのインデックスされた flat ファイルについて説明しています。bioperl-db ライブラリと BioSQL スキーマモジュールのインストールと設定が必要なローカルのリレーショナルデータベースへの明示的なアクセスに関しては IV.3 節で詳しく説明しています。

OBDA(Open Bioinformatics Data Access)Registory システムを使うアプローチも可能です。OBDA を用いると、データベースがフラットファイルであるかリレーショナル DB であるか、あるいはローカルにあるかネット上でしかアクセスできないかについて知る必要なくデータベースの配列データをインポートすることが可能です。レジストリ設定ファイルの必要な設定の仕方と、レジストリによる配列データへのアクセスの仕方については doc/howto ディレクトリ配下の BIODATABASE_ACCESS に記述しているのでここでは繰り返さない（訳注: doc/howto/html/OBDA_Access.html, doc/howto/pdf/OBDA_Access.pdf ファイルらしい）。

III.1.1 リモートデータベースへのアクセス（Bio::DB::GenBank, など）

主要な分子生物学データベース上の配列データへのアクセスは bioperl では簡単です。配列のアクセッション番号か ID にてデータにアクセス可能です。バッチモードのアクセスでは複数の配列の有効な検索手段もサポートしています。GeBank からのデータ検索では、たとえばコードは下記のようになります。

```
$gb = new Bio::DB::GenBank();
# this returns a Seq object :
$seq1 = $gb->get_Seq_by_id('MUSIGHBA1');
# this returns a Seq object :
$seq2 = $gb->get_Seq_by_acc('AF303112');
# this returns a SeqIO object :
```

```
$seqio = $gb->get_Stream_by_id(["J00522","AF303112","2981014"]);
```

SeqIO オブジェクトの使用法については III.2.1 を参照してください。

bioperl では現在、GenBank、GenPept、RefSeq、SwissProt、EMBL データベースへの配列検索をサポートしています。詳細については、
Bio::DB::GenBank、Bio::DB::GenPept、Bio::DB::SwissProt、Bio::DB::RefSeq、Bio::DB::EBML の manpage を参照してください。データベースのミラーとして別のデータベースを指定することもできます。これは特に ExPaSy で多くのミラーがある SwissProt に関係します。またローカルのファイアウォールの後ろのプロクシサーバーの設定オプションもあります。

EBI のサーバーにクエリーを投げる、Bio::DB::RefSeq モジュールを通して、NCBI の RefSeq 配列の検がサポートされています。RefSeq 検索に関していくつかの警告がありますので使う前に Bio::DB::RefSeq の manpage を参照してください。GenBank での RefSeq の ID
は"NT_"、"NC_"、"NG_"、"NM_"、"NP_"、"XM_"、"XR_"、"XP_"で始まります(詳細は
http://www.ncbi.nlm.nih.gov/LocusLink/refseq.html 参照)。Bio::DB::GenBank はこれらの ID のエントリについて検索できますが、本当の意味での GenBank エントリでないことを心に留めておいてください。"CONTIG"エントリを意味する"NT_"で始まるエントリを検索するについての詳細は Bio::DB::GenBank の manpage を参照してください。

bioperl はリモートの Ace データベース検索もサポートしています。この機能では外部の AcePerl モジュールの存在を要求します。aceperl モジュールを http://stein.cshl.org/AcePerl/からダウンロードしインストールする必要があります。

リモートのデータベースにアクセスするほかのモジュールは、EBI に dbfetch スクリプトのクエリを投げる BioFetch があります。EMBL、GenBank、SWALL データベースが扱え、異なったフォーマットあるいはストリーム(SeqIO オブジェクト)あるいは"tempfiles"で、エントリを検索できます。
詳細は Bio::DB::BioFetch の manpage を参照のこと。

III.1.2 ローカルデータベース (Bio::Index::*, bp_index.pl, bp_fetch.pl, Bio::DB::*)のインデックス化とアクセス

Bio::Index か Bio::DB::Fasta オブジェクトの手段によりローカル配列データファイルに bioperl はそれぞれインデックス付けが行えます。Bio::Index により、GenBank、SwissProt、Pfam、EMBL、FASTA の配列フォーマットに対応しており、記述のリモートデータベースへのアクセスとよく似た方法でアクセスが可能です。たとえば、FASTA ファイルのインデックスされたフラットファイルにアクセスし、そのなかから1ファイルを検索する場合はスクリプトは以下のようになります。

```
# script 1: create the index
use Bio::Index::Fasta; # using fasta file format
use strict; # some users have reported that this is necessary

my $Index_File_Name = shift;
my $inx = Bio::Index::Fasta->new(
  -filename => $Index_File_Name,
  -write_flag => 1);
$inx->make_index(@ARGV);

# script 2: retrieve some files
use Bio::Index::Fasta;
use strict; # some users have reported that this is necessary

my $Index_File_Name = shift;
```

```
my $inx = Bio::Index::Fasta->new($Index_File_Name);
foreach  my $id (@ARGV) {
  my $seq = $inx->fetch($id);  # Returns Bio::Seq object
  # do something with the sequence
}
```

より複雑でフレキシブルなインデックスシステムを作成し使用する場合には bioperl 配布物に含まれる scripts/index ディレクトリ内の二つのサンプルスクリプト、bp_index.PLS、bp_fetch.PLS を参照してください。これらのスクリプトはローカルファイルのインデックスシステム開発のテンプレートとなるでしょう。

FASTA フォーマットファイルをインデックスしクエリーとするために bioperl は Bio::DB::Fasta もサポートしています。Bio::Index::Fasta に似ていますが、多くのメソッドをサポートしています、例えば、

```
use Bio::DB::Fasta;
use strict;

my $db = Bio::DB::Fasta->new($file);  # one file or many files
my $seqstring = $db->seq($id);        # get a sequence as string
my $seqobj = $db->get_Seq_by_id($id); # get a PrimarySeq obj
my $desc = $db->header($id);          # get the header, or description line
```

このモジュールのすべての機能に関しては Bio::DB::Fasta の manpage 参照のこと。

どちらのモジュールも、"gi|4556644|gb|X45555"の文字列内の gi 番号のような ID として FASTA のヘッダーで指定されている文字列を指定することができます。"test.fa"内の下記のような Fasta フォーマット配列では、
Both modules also offer the user the ability to designate a specific string within the fasta header as the desired id, such as the gi number within the string ``gi|4556644|gb|X45555''. Consider the following fasta-formatted sequence, in ``test.fa'':

```
>gi|523232|emb|AAC12345|sp|D12567 titin fragment
MHRHHRTGYSAAYGPLKJHGYVHFIMCVVVSWWASDVVTYIPLLLNNSSAGWKRWWWIIFGGE
GHGHHRTYSALWWPPLKJHGSKHFILCVKVSWLAKKERTYIPKKILLMMGGWWAAWWWI
```

Bio::Index::Fasta と Bio::DB::Fasta はデフォルトでは、検索のキーとして Fasta のヘッダーの最初の「語」を使います。この例では"gi|523232|emb|AAC12345|sp|D12567"です。キーとしてより便利なものはひとつの ID です。"test.fa"ファイルと"test.fa.idx"と呼ばれるインデックスファイルではキーは SwissProt あるいは"sp"であり、コード例を以下に示します。
By default Bio::Index::Fasta and Bio::DB::Fasta will use the first ``word'' they encounter in the fasta header as the retrieval key, in this case ``gi|523232|emb|AAC12345|sp|D12567''. What would be more useful as a key would be a single id. The code below will index the ``test.fa'' file and create an index file called ``test.fa.idx'' where the keys are the Swissprot, or ``sp'', identifiers.

```
$ENV{BIOPERL_INDEX_TYPE} = "SDBM_File";
# look for the index in the current directory
$ENV{BIOPERL_INDEX} = ".";

my $file_name = "test.fa";
my $inx = Bio::Index::Fasta->new( -filename   => $file_name . ".idx",
                  -write_flag => 1 );
# pass a reference to the critical function to the Bio::Index object
$inx->id_parser(\&get_id);
# make the index
```

```
$inx->make_index($file_name);

# here is where the retrieval key is specified
sub get_id {
  my $header = shift;
  $header =~ /^>.*\bsp\|([A-Z]\d{5}\b)/;
  $1;
}
```

ここで配列を検索する方法は、Bio::Seqオブジェクトを用いて、
Here is how you would retrieve the sequence, as a Bio::Seq object:

```
my $seq = $inx->fetch("D12567");
print $seq->seq;
```

SwissProtのIDかGIナンバー、FASTAのヘッダーを用いて配列を検索するとき、複数のGIナンバーとSwissProtのIDはヘッダーにどう連結するか？
What if you wanted to retrieve a sequence using either a Swissprot id or a gi number and the fasta header was actually a concatenation of headers with multiple gi's and Swissprots?

```
>gi|523232|emb|AAC12345|sp|D12567|gi|7744242|sp|V11223 titin fragment
```

その機能を編集し、IDパーサーメソッドに移動するには、
Modify the function that's passed to the id_parser method:

```
sub get_id {
  my $header = shift;
  my (@sps) = $header =~ /^>.*\bsp\|([A-Z]\d{5})\b/g;
  my (@gis) = $header =~ /gi\|(\d+)\b/g;
  return (@sps,@gis);
}
```

Bio::DB::Fastaモジュールは同じ原理を用いるが、文法はすこし異なる。たとえば、
The Bio::DB::Fasta module uses the same principle, but the syntax is slightly different, for example:

```
my $db = Bio::DB::Fasta->new('test.fa', -makeid=>\&make_my_id);
my $seqobj = $db->get_Seq_by_id($id);

sub make_my_id {
  my $description_line = shift;
  $description_line =~ /gi\|(\d+)\|emb\|(\w+)/;
  ($1,$2);
}
```

コアbioperlインストールではリレーショナルデータベースでの配列へのアクセスとデータの格納をサポートしていません。しかし、補助のbioperl-dbライブラリを用いることによりこの機能が可能です。詳細はIV.3節を参照のこと。

III.2 データベースとファイルレコードのフォーマット変換
III.2 Transforming formats of database/ file records

III.2.1 配列ファイルの変換(SeqIO)
III.2.1 Transforming sequence files (SeqIO)

多くの広く使われているデータフォーマット間の配列データの変換が、通常かつ伝統的なバイオインフォマティクスの作業です。しかし、bioperl の SeqIO オブジェクトによりこの雑用がいとも簡単に行えます。多くのフォーマット、Fasta, EMBL, GenBank, Swissprot, PIR, GCG, SCF, phd/phred, Ace, fastq, exp, chado, or raw (plain sequence)に関してひとつのあるいはマルチプルファイルの配列のストリームを SeqIO は読むことができます。また、トレースファイルである、alf, ztr, abi, ctf, そして ctr フォーマットもパースすることができます。SeqIO に一度配列が読み込まれると、元の配列ソースに依存して、Seq、PrimarySeq、RichSeq オブジェクト形式として bioperl で扱うことができます。くわえて
A common - and tedious - bioinformatics task is that of converting sequence data among the many widely used data formats. Bioperl's SeqIO object, however, makes this chore a breeze. SeqIO can read a stream of sequences - located in a single or in multiple files - in a number of formats: Fasta, EMBL, GenBank, Swissprot, PIR, GCG, SCF, phd/phred, Ace, fastq, exp, chado, or raw (plain sequence). SeqIO can also parse tracefiles in alf, ztr, abi, ctf, and ctr format Once the sequence data has been read in with SeqIO, it is available to bioperl in the form of Seq, PrimarySeq, or RichSeq objects, depending on what the sequence source is. Moreover, the sequence objects can then be written to another file (again using SeqIO) in any of the supported data formats making data converters simple to implement, for example:

```
use Bio::SeqIO;
$in  = Bio::SeqIO->new(-file => "inputfilename",
            -format => 'Fasta');
$out = Bio::SeqIO->new(-file => ">outputfilename",
            -format => 'EMBL');
while ( my $seq = $in->next_seq() ) {$out->write_seq($seq); }
```

In addition, the perl ``tied filehandle'' syntax is available to SeqIO, allowing you to use the standard <> and print operations to read and write sequence objects, eg:

```
$in  = Bio::SeqIO->newFh(-file => "inputfilename" ,
             -format => 'fasta');
$out = Bio::SeqIO->newFh(-format => 'embl');
print $out $_ while <$in>;
```

If the ``-format'' argument isn't used then Bioperl will try to determine the format based on the file's suffix, in a case-insensitive manner. If there's no suffix available then SeqIO will attempt to guess the format based on actual content. Here is the current set of suffixes:

Format	Suffixes	Comment
fasta	fasta|fast|seq|fa|fsa|nt|aa	Fasta
genbank	gb|gbank|genbank|gbs|gbk	Genbank
scf	scf	SCF tracefile
pir	pir	PIR
embl	embl|ebl|emb|dat	EMBL
raw	txt	plain
gcg	gcg	GCG
ace	ace	ACeDB
bsml	bsm|bsml	BSML XML
game		GAME XML
swiss	swiss|sp	SwissProt

phd	phd|phred	Phred
fastq	fastq	Fastq
Locuslink		LL_tmpl format
qual		Phred quality file
chado		Chado XML
tinyseq		NCBI TinySeq XML
exp	exp	Staden experiment file
abi*	abi	ABI tracefile
alf*	alf	ALF tracefile
ctf*	ctf	CTF tracefile
ztr*	ztr	ZTR tracefile
pln*	pln	Staden plain tracefile

* These formats require the bioperl-ext package and the io_lib library from the Staden package

For more information see the Bio::SeqIO manpage or the SeqIO HOWTO (http://bioperl.org/HOWTOs/html/SeqIO.html).

III.2.2 Transforming alignment files (AlignIO)

Data files storing multiple sequence alignments also appear in varied formats. AlignIO is the bioperl object for conversion of alignment files. AlignIO is patterned on the SeqIO object and its commands have many of the same names as the commands in SeqIO. Just as in SeqIO the AlignIO object can be created with ``-file'' and ``-format'' options:

```
use Bio::AlignIO;
my $io = Bio::AlignIO->new(-file   => "receptors.aln",
             -format => "clustalw" );
```

If the ``-format'' argument isn't used then Bioperl will try and determine the format based on the file's suffix, in a case-insensitive manner. Here is the current set of suffixes:

Format	Suffixes	Comment
bl2seq		
clustalw	aln	
emboss*	water|needle	
fasta	fasta|fast|seq|fa|fsa|nt|aa	
maf	maf	
mase		Seaview
mega	meg|mega	
meme	meme	
metafasta		
msf	msf|pileup|gcg	GCG
nexus	nexus|nex	
pfam	pfam|pfm	
phylip	phylip|phlp|phyl|phy|phy|ph	interleaved
prodom		
psi	psi	PSI-BLAST
selex	selex|slx|selx|slex|sx	HMMER
stockholm		

*water, needle, matcher, stretcher, merger, and supermatcher See IV.2.1 on EMBOSS for more information

Unlike SeqIO AlignIO cannot create output files in every format. AlignIO currently supports output in these 6 formats: fasta, mase, selex, clustalw, msf/gcg, and phylip (interleaved).

Another significant difference between AlignIO and SeqIO is that AlignIO handles IO for only a single alignment at a time but SeqIO.pm handles IO for multiple sequences in a single stream. Syntax for AlignIO is almost identical to that of SeqIO:

```
use Bio::AlignIO;
$in  = Bio::AlignIO->new(-file => "inputfilename" ,
            -format => 'fasta');
$out = Bio::AlignIO->new(-file => ">outputfilename",
            -format => 'pfam');
while ( my $aln = $in->next_aln() ) { $out->write_aln($aln); }
```

The only difference is that the returned object reference, $aln, is to a SimpleAlign object rather than to a Seq object.

AlignIO also supports the tied filehandle syntax described above for SeqIO. See the Bio::AlignIO manpage, the Bio::SimpleAlign manpage, and section III.5 on SimpleAlign for more information.

III.3 Manipulating sequences

Bioperl contains many modules with functions for sequence analysis. And if you cannot find the function you want in bioperl you may be able to find it in EMBOSS or PISE , which are accessible through the bioperl-run auxiliary library (see IV.2.1).

III.3.1 Manipulating sequence data with Seq methods

OK, so we know how to retrieve sequences and access them as sequence objects. Let's see how we can use sequence objects to manipulate our sequence data and retrieve information. Seq provides multiple methods for performing many common (and some not-so-common) tasks of sequence manipulation and data retrieval. Here are some of the most useful:

These methods return strings or may be used to set values:

```
$seqobj->display_id();       # the human read-able id of the sequence
$seqobj->seq();              # string of sequence
$seqobj->subseq(5,10);       # part of the sequence as a string
$seqobj->accession_number(); # when there, the accession number
$seqobj->alphabet();         # one of 'dna','rna','protein'
$seqobj->primary_id();       # a unique id for this sequence irregardless
              # of its display_id or accession number
$seqobj->desc();             # a description of the sequence
```

It is worth mentioning that some of these values correspond to specific fields of given formats. For example, the display_id method returns the LOCUS name of a Genbank entry, the (\S+) following the > character in a Fasta file, the ID from a SwissProt file, and so on. The desc() method will return the DEFINITION line of a Genbank file, the line following the display_id in a Fasta file, and the DE field in a SwissProt file.

The following methods return an array of Bio::SeqFeature objects:

```
 $seqobj->get_SeqFeatures;      # The 'top level' sequence features
 $seqobj->get_all_SeqFeatures;  # All sequence features, including sub-
                # seq features
```

For a comment annotation, you can use:

```
 use Bio::Annotation::Comment;
 $seq->annotation->add_Annotation('comment',
   Bio::Annotation::Comment->new(-text => 'some description');
```

For a reference annotation, you can use:

```
 use Bio::Annotation::Reference;
 $seq->annotation->add_Annotation('reference',
   Bio::Annotation::Reference->new(-authors  => 'author1,author2',
                  -title    => 'title line',
                  -location => 'location line',
                  -medline  => 998122 );
```

Sequence features will be discussed further in section III.7 on machine-readable sequence annotation. A general description of the object can be found in the Bio::SeqFeature::Generic manpage, and a description of related, top-level annotation is found in the Bio::Annotation::Collection manpage.

Additional sample code for obtaining sequence features can be found in the script gb2features.pl in the subdirectory examples/DB. Finally, there's a HOWTO on features and annotations (http://bioperl.org/HOWTOs/html/Feature-Annotation.html) and there's a section on features in the FAQ (http://bioperl.org/Core/Latest/faq.html#5).

The following methods returns new sequence objects, but do not transfer the features from the starting object to the resulting feature:

```
 $seqobj->trunc(5,10);  # truncation from 5 to 10 as new object
 $seqobj->revcom;       # reverse complements sequence
 $seqobj->translate;    # translation of the sequence
```

Note that some methods return strings, some return arrays and some return objects. See the Bio::Seq manpage for more information.

Many of these methods are self-explanatory. However, bioperl's flexible translation methods warrant further comment. Translation in bioinformatics can mean two slightly different things:

1. Translating a nucleotide sequence from start to end.
2. Taking into account the constraints of real coding regions in mRNAs.

The bioperl implementation of sequence-translation does the first of these tasks easily. Any sequence object which is not of alphabet 'protein' can be translated by simply calling the method which returns a protein sequence object:

```
 $translation1 = $my_seq_object->translate;
```

However, the translate method can also be passed several optional parameters to modify its behavior. For example, the first two arguments to translate() can be used to modify the characters used to represent stop (default '*') and unknown amino acid ('X'). (These are normally best left untouched.) The third argument

determines the frame of the translation. The default frame is ``0''. To get translations in the other two forward frames, we would write:

```
$translation2 = $my_seq_object->translate(undef,undef,1);
$translation3 = $my_seq_object->translate(undef,undef,2);
```

The fourth argument to translate() makes it possible to use alternative genetic codes. There are currently 16 codon tables defined, including tables for 'Vertebrate Mitochondrial', 'Bacterial', 'Alternative Yeast Nuclear' and 'Ciliate, Dasycladacean and Hexamita Nuclear' translation. These tables are located in the object Bio::Tools::CodonTable which is used by the translate method. For example, for mitochondrial translation:

```
$human_mitochondrial_translation = $seq_obj->translate(undef,undef,undef,2);
```

If we want to translate full coding regions (CDS) the way major nucleotide databanks EMBL, GenBank and DDBJ do it, the translate method has to perform more tricks. Specifically, 'translate' needs to confirm that the sequence has appropriate start and terminator codons at the beginning and the end of the sequence and that there are no terminator codons present within the sequence. In addition, if the genetic code being used has an atypical (non-ATG) start codon, the translate method needs to convert the initial amino acid to methionine. These checks and conversions are triggered by setting the fifth argument of the translate method to evaluate to ``true''.

If argument 5 is set to true and the criteria for a proper CDS are not met, the method, by default, issues a warning. By setting the sixth argument to evaluate to ``true'', one can instead instruct the program to die if an improper CDS is found, e.g.

```
$protein_object = $cds->translate(undef,undef,undef,undef,1,'die_if_errors');
```

See the Bio::Tools::CodonTable manpage for related details.

III.3.2 Obtaining basic sequence statistics (SeqStats,SeqWord)

In addition to the methods directly available in the Seq object, bioperl provides various helper objects to determine additional information about a sequence. For example, SeqStats object provides methods for obtaining the molecular weight of the sequence as well the number of occurrences of each of the component residues (bases for a nucleic acid or amino acids for a protein.) For nucleic acids, SeqStats also returns counts of the number of codons used. For example:

```
use SeqStats;
$seq_stats  = Bio::Tools::SeqStats->new($seqobj);
$weight = $seq_stats->get_mol_wt();
$monomer_ref = $seq_stats->count_monomers();
$codon_ref = $seq_stats->count_codons();  # for nucleic acid sequence
```

Note: sometimes sequences will contain ambiguous codes. For this reason, get_mol_wt() returns a reference to a two element array containing a greatest lower bound and a least upper bound of the molecular weight.

The SeqWords object is similar to SeqStats and provides methods for calculating frequencies of ``words'' (e.g. tetramers or hexamers) within the sequence. See the Bio::Tools::SeqStats manpage and the Bio::Tools::SeqWords manpage for more information.

III.3.3 Identifying restriction enzyme sites (Bio::Restriction)

Another common sequence manipulation task for nucleic acid sequences is locating restriction enzyme cutting sites. Bioperl provides the Bio::Restriction::Enzyme, Bio::Restriction::EnzymeCollection, and Bio::Restriction::Analysis objects for this purpose. These modules replace the older module Bio::Tools::RestrictionEnzyme. A new collection of enzyme objects would be defined like this:

```
 use Bio::Restriction::EnzymeCollection;
 my $all_collection = Bio::Restriction::EnzymeCollection;
```

Bioperl's default Restriction::EnzymeCollection object comes with data for more than 500 different Type II restriction enzymes. A list of the available enzyme names can be accessed using the available_list() method, but these are just the names, not the functional objects. You also have access to enzyme subsets. For example to select all available Enzyme objects with recognition sites that are six bases long one could write:

```
my $six_cutter_collection = $all_collection->cutters(6);
foreach my $enz ($six_cutter_collection){
  print $enz->name,"\t",$enz->site,"\t",$enz->overhang_seq,"\n";
  # prints name, recognition site, overhang
}
```

There are other methods that can be used to select sets of enzyme objects, such as unique_cutters() and blunt_enzymes(). You can also select a Enzyme object by name, like so:

```
my $ecori_enzyme = $all_collection->get_enzyme('EcoRI');
```

Once an appropriate enzyme has been selected, the sites for that enzyme on a given nucleic acid sequence can be obtained using the fragments() method. The syntax for performing this task is:

```
 use Bio::Restriction::Analysis;
 my $analysis = Bio::Restriction::Analysis->new(-seq => $seq);
 # where $seq is the Bio::Seq object for the DNA to be cut
 @fragments =  $analysis->fragments($enzyme);
 # and @fragments will be an array of strings
```

To get information on isoschizomers, methylation sites, microbe source, vendor or availability you will need to create your EnzymeCollection directly from a REBASE file, like this:

```
use Bio::Restriction::IO;
my $re_io = Bio::Restriction::IO->new(-file=>$file,-format=>'withrefm');
my $rebase_collection = $re_io->read;
```

A REBASE file in the correct format can be found at ftp://ftp.neb.com/pub/rebase, it will have a name like ``withrefm.308''. If need be you can also create new enzymes, like this:

```
my $re = new Bio::Restriction::Enzyme(-enzyme=>'BioRI',-seq=>'GG^AATTCC');
```

For more informatation see the Bio::Restriction::Enzyme manpage, the Bio::Restriction::EnzymeCollection manpage, the Bio::Restriction::Analysis manpage, and the Bio::Restriction::IO manpage.

III.3.4 Identifying amino acid cleavage sites (Sigcleave)

For amino acid sequences we may be interested to know whether the amino acid sequence contains a cleavable signal sequence for directing the transport of the protein within the cell. SigCleave is a program (originally part of the EGCG molecular biology package) to predict signal sequences, and to identify the cleavage site based on the von Heijne algorithm.

The threshold setting controls the score reporting. If no value for threshold is passed in by the user, the code defaults to a reporting value of 3.5. SigCleave will only return score/position pairs which meet the threshold limit.

There are 2 accessor methods for this object. signals() will return a perl hash containing the sigcleave scores keyed by amino acid position. pretty_print() returns a formatted string similar to the output of the original sigcleave utility.

The syntax for using Sigcleave is as follows:

```
 # create a Seq object, for example:
 $seqobj = Bio::Seq->new(-seq => "AALLHHHHHHGGGGPPRTTTTTVVVVVVVVVVVVVVV");

 use Bio::Tools::Sigcleave;
 $sigcleave_object = new Bio::Tools::Sigcleave
   ( -seq       => $seqobj,
    -threshold => 3.5,
    -desc      => 'test sigcleave protein seq',
    -type      => 'AMINO'
   );
 %raw_results      = $sigcleave_object->signals;
 $formatted_output = $sigcleave_object->pretty_print;
```

Note that the ``type'' in the Sigcleave object is ``amino'' whereas in a Seq object it would be called ``protein''. Please see the Bio::Tools::Sigcleave manpage for details.

III.3.5 Miscellaneous sequence utilities: OddCodes, SeqPattern

OddCodes:

For some purposes it's useful to have a listing of an amino acid sequence showing where the hydrophobic amino acids are located or where the positively charged ones are. Bioperl provides this capability via the module Bio::Tools::OddCodes.

For example, to quickly see where the charged amino acids are located along the sequence we perform:

```
 use Bio::Tools::OddCodes;
 $oddcode_obj = Bio::Tools::OddCodes->new($amino_obj);
 $output = $oddcode_obj->charge();
```

The sequence will be transformed into a three-letter sequence (A,C,N) for negative (acidic), positive (basic), and neutral amino acids. For example the ACDEFGH would become NNAANNC.

For a more complete chemical description of the sequence one can call the chemical() method which turns sequence into one with an 8-letter chemical alphabet { A (acidic), L (aliphatic), M (amide), R (aromatic), C (basic), H (hydroxyl), I (imino), S (sulfur) }:

```
$output = $oddcode_obj->chemical();
```

In this case the sample sequence ACDEFGH would become LSAARAC.

OddCodes also offers translation into alphabets showing alternate characteristics of the amino acid sequence such as hydrophobicity, ``functionality'' or grouping using Dayhoff's definitions. See the documentation in the Bio::Tools::OddCodes manpage for further details.

SeqPattern:

The SeqPattern object is used to manipulate sequences using perl regular expressions. A key motivation for SeqPattern is to have a way of generating a reverse complement of a nucleic acid sequence pattern that includes ambiguous bases and/or regular expressions. This capability leads to significant performance gains when pattern matching on both the sense and anti-sense strands of a query sequence are required. Typical syntax for using SeqPattern is shown below. For more information, there are several interesting examples in the script seq_pattern.pl in the examples/tools directory.

```
use Bio::Tools::SeqPattern;
$pattern     = '(CCCCT)N{1,200}(agggg)N{1,200}(agggg)';
$pattern_obj = new Bio::Tools::SeqPattern(-SEQ  => $pattern,
                      -TYPE => 'dna');
$pattern_obj2 = $pattern_obj->revcom();
$pattern_obj->revcom(1); # returns expanded rev complement pattern.
```

More detail can be found in the Bio::Tools::SeqPattern manpage.

III.3.6 Converting coordinate systems (Coordinate::Pair, RelSegment)

Coordinate system conversion is a common requirement, for example, when one wants to look at the relative positions of sequence features to one another and convert those relative positions to absolute coordinates along a chromosome or contig. Although coordinate conversion sounds pretty trivial it can get fairly tricky when one includes the possibilities of switching to coordinates on negative (i.e. Crick) strands and/or having a coordinate system terminate because you have reached the end of a clone or contig. Bioperl has two different approaches to coordinate-system conversion (based on the modules Bio::Coordinate::Pair and Bio::DB::GFF::RelSegment, respectively).

The Coordinate::Pair approach is somewhat more ``low level''. With it, you define an input coordinate system and an output coordinate system, where in each case a coordinate system is a triple of a start position, end position and strand. The end position is especially important when dealing with unfinished assemblies where the coordinate system ends when one reaches the end of the sequence of a clone or contig. Once one has defined the two coordinate systems, one defines a Coordinate::Pair to map between them. Then one can map positions between the coordinates systems with code such as this:

```
$input_coordinates = Bio::Location::Simple->new
(-seq_id => 'propeptide', -start => 1000, -end => 2000, -strand=>1 );
$output_coordinates = Bio::Location::Simple->new
(-seq_id => 'peptide', -start => 1100, -end => 2100, -strand=>1 );
$pair = Bio::Coordinate::Pair->new
(-in => $input_coordinates ,  -out => $output_coordinates   );
$pos = Bio::Location::Simple->new (-start => 500, -end => 500 );
$res = $pair->map($pos);
$converted_start = $res->start;
```

```
In this example $res is also a Bio::Location object, as you'd expect. See the
documentation for Bio::Coordinate::Pair and Bio::Coordinate::GeneMapper for more
details.

The Bio::DB::GFF::RelSegment approach is designed more for handling coordinate
transformations of sequence features rather than for transforming arbitrary
coordinate systems. With Bio::DB::GFF::RelSegment you define a coordinate system
relative to a specific feature (called the ``refseq''). You also have access to
the absolute coordinate system (typically of the entire chromosome.) You can
determine the position of a feature relative to some other feature simply by
redefining the relevant reference feature (i.e. the ``refseq'') with code like
this:

 $db = Bio::DB::GFF->new(-dsn     => 'dbi:mysql:elegans',
             -adaptor => 'dbi:mysqlopt');

 $segment = $db->segment('ZK909');
 $relative_start = $segment->start;  # $relative_start = 1;

 # Now retrieve the start position of ZK909 relative to feature ZK337
 $segment->refseq('ZK337');
 $relative_start = $segment->start;

 # Now retrieve the start position of ZK909 relative to the entire chromosome
 $absolute_start =  $segment->abs_start;

This approach is convenient because you don't have to keep track of coordinates
directly, you just keep track of the name of a feature which in turn marks the
coordinate-system origin. However, this approach does require that you have
stored all the sequence features in GFF format. Moreover,
Bio::DB::GFF::RelSegment has been principally developed and tested for
applications where all the sequence features are stored in a Bioperl-db
relational database. However, if one wants to use the Bio:DB::GFF machinery
(including its coordinate transformation capabilities) without building a local
relational database, this is possible by defining the 'database' as having an
adaptor called 'memory', e.g.

 $db = Bio::DB::GFF->new( '-adaptor' => 'memory' );

For more details on coordinate transformations and other GFF-related
capabilities in Bioperl see the Bio::DB::GFF::RelSegment manpage, the
Bio::DB::GFF manpage, and the test file t/BioDBGFF.t.
```

III.4 類似配列の検索

何らかの方法で興味ある配列と類似する配列を同定することが分子生物学の基本的な作業 NCBI によってオリジナルに開発された BLAST プログラムはそのような配列の同定について広く用いられています。BLAST を実行したり、BLAST により出力される膨大な結果を切り出したりするためのいくつかのモジュールを bioperl と bioperl の実行パッケージは提供します。

III.4.1 BLAST の実行(RemoteBlast.pm を用いて)

RemoteBlast オブジェクトを使って NCBI 上の BLAST のリモート実行を bioperl はサポートします。

リモート BLAST 実行のためのスクリプトの骨子は以下のような感じです。

```
 $remote_blast = Bio::Tools::Run::RemoteBlast->new (
```

```
     -prog => 'blastp',-data => 'ecoli',-expect => '1e-10' );
$r = $remote_blast->submit_blast("t/data/ecolitst.fa");
while (@rids = $remote_blast->each_rid ) {
  foreach $rid ( @rids ) {$rc = $remote_blast->retrieve_blast($rid);}
}
```

リモートジョブのいくつかのパラメーターを変えたい場合、たとえばマトリクスを代えたい場合の例は、

```
$Bio::Tools::Run::RemoteBlast::HEADER{'MATRIX_NAME'} = 'BLOSUM25';
```

多くの CGI パラメーターの記述法については、

```
http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html
```

上記のスクリプトが2つの部分に分かれていることに注目してください。BLAST の実行と結果の取出しです。NCBI BLAST を多量に用いるときは BLAST 実行と結果を得る間隔を開けることが重要です。

$rc オブジェクトは BLAST 結果を含み、Bio::Tools::BPlite あるいは Bio::SearchIO にて切り出すことができます。バージョン 1.0 以降ではデフォルトのオブジェクトは SearchIO を返します。オブジェクトタイプを変えるには readmethod パラメータを変える必要がありますが BLAST 結果の切り出しには Bio::SearchIO が推奨されます。他のオブジェクトは今後のバージョンでは保証されません。BLAST 実行が成功したかを確認するコードか、連続実行したときに NCBI サーバーを待つための"sleep"ループを追加することがこのスクリプトを実際に使いやすくするためにはひつようです。詳細は、付録のデモスクリプトの例 22 か Bio::Tools:Run::RemoteBlast manpage を参照のこと。

リモート BLAST 工場を作成するための構文としては、StandAloneBlast、Custalw、T-Coffee 工場を作るときとはちょっと違うということにも注意すべきです。特に RemotoBlast では、'-prog' => 'blastp'のようなハイフンを伴ったパラメータが必要であり、他のプログラムではハイフンを伴ったパラメーターはないということです。

III.4.2 Search と SearchIO を用いた BLAST と FASTA 結果の切り出し

どのように BLAST 検索を実行(ローカルであるいはリモートで、また、perl インターフェースを用いるか用いないか)しても、取捨選択するには十分すぎる多量なデータが返ってくる。Blast 結果をパースするために、bioperl は Search.pm や SearchIO.pm、BPlite.pm(そのマイナー修正版である、BPpsilite、BPbl2seq)のようないくつかの異なるオブジェクトを用意しています。Blast と FASTA 結果のパース用の bioperl の基本的なインターフェースである Search と SearchIO オブジェクトについてはこの節に記述しています。古い版の BPlite については III.4.3 節で記述しています。将来の版でもサポートがされる SearchIO を使うことを推奨します。

```
The Search and SearchIO modules provide a uniform interface for parsing
sequence-similarity-search reports generated by BLAST (in standard and BLAST XML
formats), PSI-BLAST, RPS-BLAST, bl2seq and FASTA. The SearchIO modules also
provide a parser for HMMER reports and in the future, it is envisioned that the
Search/SearchIO syntax will be extended to provide a uniform interface to an
even wider range of report parsers including parsers for Genscan.

Parsing sequence-similarity reports with Search and SearchIO is straightforward.
Initially a SearchIO object specifies a file containing the report(s). The
method next_result reads the next report into a Search object in just the same
way that the next_seq method of SeqIO reads in the next sequence in a file into
a Seq object.

Once a report (i.e. a SearchIO object) has been read in and is available to the
script, the report's overall attributes (e.g. the query) can be determined and
its individual hits can be accessed with the next_hit method. Individual high-
```

scoring segment pairs for each hit can then be accessed with the next_hsp method. Except for the additional syntax required to enable the reading of multiple reports in a single file, the remainder of the Search/SearchIO parsing syntax is very similar to that of the BPlite object it is intended to replace. Sample code to read a BLAST report might look like this:

```
 # Get the report
 $searchio = new Bio::SearchIO (-format => 'blast',
                 -file   => $blast_report);
 $result = $searchio->next_result;
 # Get info about the entire report
 $result->database_name;
 $algorithm_type =  $result->algorithm;
 # get info about the first hit
 $hit = $result->next_hit;
 $hit_name = $hit->name ;
 # get info about the first hsp of the first hit
 $hsp = $hit->next_hsp;
 $hsp_start = $hsp->query->start;
```

For more details there is a good description of how to use SearchIO at http://www.bioperl.org/HOWTOs/html/SearchIO.html or in the docs/howto subdirectory of the distribution. Additional documentation can be found in the Bio::SearchIO::blast manpage, the Bio::SearchIO::psiblast manpage, the Bio::SearchIO::blastxml manpage, the Bio::SearchIO::fasta manpage, and the Bio::SearchIO manpage. There is also sample code in the examples/searchio directory which illustrates how to use SearchIO. And finally, there's a section with SearchIO questions in the FAQ (http://bioperl.org/Core/Latest/faq.html#3).

III.4.3 Parsing BLAST reports with BPlite, BPpsilite, and BPbl2seq

Bioperl's older BLAST report parsers - BPlite, BPpsilite, BPbl2seq and Blast.pm - are no longer supported but since legacy Bioperl scripts have been written which use these objects, they are likely to remain within Bioperl for some time.

Much of the user interface of BPlite is very similar to that of Search. However accessing the next hit or HSP uses methods called next_Sbjct and next_HSP, respectively - in contrast to Search's next_hit and next_hsp.

BPlite

The syntax for using BPlite is as follows where the method for retrieving hits is now called nextSbjct() (for ``subject''), while the method for retrieving high-scoring-pairs is called nextHSP():

```
 use Bio::Tools::BPlite;
 $report = new Bio::Tools::BPlite(-fh=>\*STDIN);
 $report->query;
 while(my $sbjct = $report->nextSbjct) {
    $sbjct->name;
    while (my $hsp = $sbjct->nextHSP) { print $hsp->score,"\n"; }
 }
```

A complete description of the module can be found in the Bio::Tools::BPlite manpage.

BPpsilite

BPpsilite and BPbl2seq are objects for parsing (multiple iteration) PSIBLAST

reports and Blast bl2seq reports, respectively. They are both minor variations on the BPlite object. See the Bio::Tools::BPbl2seq manpage and the Bio::Tools::BPpsilite manpage for details.

The syntax for parsing a multiple iteration PSIBLAST report is as shown below. The only significant additions to BPlite are methods to determine the number of iterated blasts and to access the results from each iteration. The results from each iteration are parsed in the same manner as a (complete) BPlite object.

```
use Bio::Tools::BPpsilite;
$report = new Bio::Tools::BPpsilite(-fh=>\*STDIN);
$total_iterations = $report->number_of_iterations;
$last_iteration = $report->round($total_iterations)
while(my $sbjct =  $last_iteration ->nextSbjct) {
   $sbjct->name;
   while (my $hsp = $sbjct->nextHSP) {$hsp->score; }
}
```

See the Bio::Tools::BPpsilite manpage for details.

BPbl2seq

BLAST bl2seq is a program for comparing and aligning two sequences using BLAST. Although the report format is similar to that of a conventional BLAST, there are a few differences. Consequently, the standard bioperl parser BPlite ia unable to read bl2seq reports directly. From the user's perspective, one difference between bl2seq and other blast reports is that the bl2seq report does not print out the name of the first of the two aligned sequences. Consequently, BPbl2seq has no way of identifying the name of one of the initial sequence unless it is explicitly passed to constructor as a second argument as in:

```
use Bio::Tools::BPbl2seq;
$report = Bio::Tools::BPbl2seq->new(-file => "t/data/dblseq.out",
                  -queryname => "ALEU_HORVU");
$hsp = $report->next_feature;
$answer=$hsp->score;
```

In addition, since there will only be (at most) one subject (hit) in a bl2seq report one should use the method $report->next_feature, rather than $report->nextSbjct->nextHSP to obtain the next high scoring pair. See the Bio::Tools::BPbl2seq manpage for more details.

Blast.pm

The Bio::Tools::Blast parser has been removed from Bioperl as of version 1.1. Consequently, the BPlite parser (described in the section III.4.3) or the Search/SearchIO parsers (section III.4.2) should be used for BLAST parsing within bioperl. SearchIO is the preferred approach and will be formally supported in future releases.

III.4.4 Parsing HMM reports (HMMER::Results, SearchIO)

Blast is not the only sequence-similarity-searching program supported by bioperl. HMMER is a Hidden Markov Model (HMM) program that (among other capabilities) enables sequence similarity searching, from http://hmmer.wustl.edu. Bioperl does not currently provide a perl interface for running HMMER. However, bioperl does provide 2 HMMER report parsers, the recommended SearchIO HMMER parser and an older parser called HMMER::Results.

SearchIO can parse reports generated both by the HMMER program hmmsearch - which

searches a sequence database for sequences similar to those generated by a given HMM - and the program hmmpfam - which searches a HMM database for HMMs which match domains of a given sequence. Sample usage for parsing a hmmsearch report might be:

```
 use Bio::SearchIO;

 $in = new Bio::SearchIO(-format => 'hmmer',-file => '123.hmmsearch');
 while ( $res = $in->next_result ){
  # get a Bio::Search::Result::HMMERResult object
  print $res->query_name, " for HMM ", $res->hmm_name, "\n";
  while ( $hit = $res->next_hit ){
   print $hit->name, "\n";
   while ( $hsp = $hit->next_hsp ){
    print "length is ", $hsp->length, "\n";
   }
  }
 }
```

Purists may insist that the term ``hsp'' is not applicable to hmmsearch or hmmpfam results and they may be correct - this is an unintended consequence of using the flexible and extensible SearchIO approach. See the Bio::Search::Result::HMMERResult manpage for more information.

For documentation on the older, unsupported HMMER parser, look at the Bio::Tools::HMMER::Results manpage.

III.4.5 Running BLAST locally (StandAloneBlast)

There are several reasons why one might want to run the Blast programs locally - speed, data security, immunity to network problems, being able to run large batch runs, wanting to use custom or proprietary databases, etc. The NCBI provides a downloadable version of blast in a stand-alone format, and running blast locally without any use of perl or bioperl is completely straightforward. However, there are situations where having a perl interface for running the blast programs locally is convenient.

The module Bio::Tools::Run::StandAloneBlast offers the ability to wrap local calls to blast from within perl. All of the currently available options of NCBI Blast (e.g. PSIBLAST, PHIBLAST, bl2seq) are available from within the bioperl StandAloneBlast interface. Of course, to use StandAloneBlast, one needs to have installed BLAST from NCBI locally as well as one or more blast-readable databases.

Basic usage of the StandAloneBlast.pm module is simple. Initially, a local blast factory object is created.

```
 @params = (program  => 'blastn',
       database => 'ecoli.nt');
 $factory = Bio::Tools::Run::StandAloneBlast->new(@params);
```

Any parameters not explicitly set will remain as the BLAST defaults. Once the factory has been created and the appropriate parameters set, one can call one of the supported blast executables. The input sequence(s) to these executables may be fasta file(s), a Seq object or an array of Seq objects, eg

```
 $input = Bio::Seq->new(-id  =>"test query",
             -seq =>"ACTAAGTGGGGG");
 $blast_report = $factory->blastall($input);
```

The returned blast report will be in the form of a bioperl parsed-blast object. The report object may be either a SearchIO, BPlite, BPpsilite, BPbl2seq or Blast object depending on the type of blast search - the SearchIO object is returned by default. The raw blast report is also available.

The syntax for running PHIBLAST, PSIBLAST and bl2seq searches via StandAloneBlast is also straightforward. See the Bio::Tools::Run::StandAloneBlast manpage documentation for details. In addition, the script standaloneblast.pl in the examples/tools directory contains descriptions of various possible applications of the StandAloneBlast object. This script shows how the blast report object can access the SearchIO blast parser directly, e.g.

```
 while (my $hit = $blast_report->next_hit){
   while (my $hsp = $sbjct->next_hsp){
     print $hsp->score," ",$hit->name,"\n";
   }
 }
```

See the sections III.4.2 and III.4.3 for more details on parsing BLAST reports.

III.5 Manipulating sequence alignments (SimpleAlign)

Once one has identified a set of similar sequences, one often needs to create an alignment of those sequences. Bioperl offers several perl objects to facilitate sequence alignment: pSW, Clustalw.pm, TCoffee.pm, dpAlign.pm and the bl2seq option of StandAloneBlast. As of release 1.2 of bioperl, using these modules (except bl2seq) requires a bioperl auxiliary library (bioperl-ext for pSW and dpAlign, bioperl-run for the others) and are therefore described in section IV. Here we describe only the module within the bioperl core package for manipulating previously created alignments, namely the SimpleAlign module.

The script aligntutorial.pl in the examples/align/ subdirectory is another good source of information of ways to create and manipulate sequence alignments within bioperl.

SimpleAlign objects are produced by bioperl-run alignment creation objects (e.g. Clustalw.pm, BLAST's bl2seq, TCoffee.pm, Lagan.pm, or pSW and dpAlign from the bioperl-ext package) or they can be read in from files of multiple-sequence alignments in various formats using AlignIO.

Some of the manipulations possible with SimpleAlign include:

  *
   slice(): Obtaining an alignment ``slice'', that is, a subalignment inclusive of specified start and end columns. Sequences with no residues in the slice are excluded from the new alignment and a warning is printed.

  *
   column_from_residue_number(): Finding column in an alignment where a specified residue of a specified sequence is located.

  *
   consensus_string(): Making a consensus string. This method includes an optional threshold parameter, so that positions in the alignment with lower percent-identity than the threshold are marked by ``?'''s in the consensus

  *
   percentage_identity(): A fast method for calculating the average percentage

identity of the alignment

  *
   consensus_iupac(): Making a consensus using IUPAC ambiguity codes from DNA and RNA.

Skeleton code for using some of these features is shown below. More detailed, working code is in bptutorial.pl example 13 and in align_on_codons.pl in the examples/align directory. Additional documentation on methods can be found in the Bio::SimpleAlign manpage and the Bio::LocatableSeq manpage.

```
 use Bio::SimpleAlign;
 $aln = Bio::SimpleAlign->new('t/data/testaln.dna');
 $threshold_percent = 60;
 $consensus_with_threshold = $aln->consensus_string($threshold_percent);
 $iupac_consensus = $aln->consensus_iupac();   # dna/rna alignments only
 $percent_ident = $aln->percentage_identity;
 $seqname = '1433_LYCES';
 $pos = $aln->column_from_residue_number($seqname, 14);
```

III.6 Searching for genes and other structures on genomic DNA (Genscan, Sim4, Grail, Genemark, ESTScan, MZEF, EPCR)

Automated searching for putative genes, coding sequences, sequence-tagged-sites (STS's) and other functional units in genomic and expressed sequence tag (EST) data has become very important as the available quantity of sequence data has rapidly increased. Many feature searching programs currently exist. Each produces reports containing predictions that must be read manually or parsed by automated report readers.

Parsers for six widely used gene prediction programs - Genscan, Sim4, Genemark, Grail, ESTScan and MZEF - are available in bioperl. The interfaces for these parsers are all similar. We illustrate the usage for Genscan and Sim4 here. The syntax is relatively self-explanatory; see the Bio::Tools::Genscan manpage, the Bio::Tools::Genemark manpage, the Bio::Tools::Grail manpage, the Bio::Tools::ESTScan manpage, the Bio::Tools::MZEF manpage, and the Bio::Tools::Sim4::Results manpage for further details.

```
 use Bio::Tools::Genscan;
 $genscan = Bio::Tools::Genscan->new(-file => 'result.genscan');
 # $gene is an instance of Bio::Tools::Prediction::Gene
 # $gene->exons() returns an array of Bio::Tools::Prediction::Exon objects
 while($gene = $genscan->next_prediction())
   { @exon_arr = $gene->exons(); }
 $genscan->close();
```

See the Bio::Tools::Prediction::Gene manpage and the Bio::Tools::Prediction::Exon manpage for more details.

```
 use Bio::Tools::Sim4::Results;
 $sim4 = new Bio::Tools::Sim4::Results(-file => 't/data/sim4.rev',
                     -estisfirst => 0);
 # $exonset is-a Bio::SeqFeature::Generic with Bio::Tools::Sim4::Exons
 # as sub features
 $exonset = $sim4->next_exonset;
 @exons = $exonset->sub_SeqFeature();
 # $exon is-a Bio::SeqFeature::FeaturePair
```

```
$exon = 1;
$exonstart = $exons[$exon]->start();
$estname = $exons[$exon]->est_hit()->seqname();
$sim4->close();
```

See the Bio::SeqFeature::Generic manpage and the Bio::Tools::Sim4::Exons manpage for more information.

A parser for the ePCR program is also available. The ePCR program identifies potential PCR-based sequence tagged sites (STSs) For more details see the documentation in the Bio::Tools::EPCR manpage. A sample skeleton script for parsing an ePCR report and using the data to annotate a genomic sequence might look like this:

```
use Bio::Tools::EPCR;
use Bio::SeqIO;
$parser = new Bio::Tools::EPCR(-file => 'seq1.epcr');
$seqio = new Bio::SeqIO(-format => 'fasta', -file => 'seq1.fa');
$seq = $seqio->next_seq;
while( $feat = $parser->next_feature ) {
   # add EPCR annotation to a sequence
   $seq->add_SeqFeature($feat);
}
```

III.7 Developing machine readable sequence annotations

Historically, annotations for sequence data have been entered and read manually in flat-file or relational databases with relatively little concern for machine readability. More recent projects - such as EBI's ENSEMBL project and the efforts to develop an XML molecular biology data specification - have begun to address this limitation. Because of its strengths in text processing and regular-expression handling, perl is a natural choice for the computer language to be used for this task. And bioperl offers numerous tools to facilitate this process - several of which are described in the following sub-sections.

III.7.1 Representing sequence annotations (SeqFeature,RichSeq,Location)

In Bioperl, most sequence annotations are stored in sequence-feature (SeqFeature) objects, where the SeqFeature object is associated with a parent Seq object. A SeqFeature object generally has a description (e.g. ``exon'', ``promoter''), a location specifying its start and end positions on the parent sequence, and a reference to its parent sequence. In addition, a Seq object can also have an Annotation object associated with it, which could be used to store database links, literature references and comments. Creating a new SeqFeature and Annotation and associating it with a Seq is accomplished with syntax like:

```
$feat = new Bio::SeqFeature::Generic(-start   => 40,
                  -end     => 80,
                  -strand  => 1,
                  -primary => 'exon',
                  -source  => 'internal' );
$seqobj->add_SeqFeature($feat); # Add the SeqFeature to the Seq object
```

Once the features and annotations have been associated with the Seq, they can be with retrieved, eg:

```
@topfeatures = $seqobj->top_SeqFeatures(); # just top level, or
@allfeatures = $seqobj->all_SeqFeatures(); # descend into sub features
```

```
 $disease_annotation = $annotations->get_Annotations('disease');
```

The individual components of a SeqFeature can also be set or retrieved with methods including:

```
 # methods which return numbers
 $feat->start          # start position
 $feat->end            # end position

 $feat->strand         # 1 means forward, -1 reverse, 0 not relevant

 # methods which return strings
 $feat->primary_tag    # the main 'name' of the sequence feature,
            # eg, 'exon'
 $feat->source_tag     # where the feature comes from, e.g. 'BLAST'

 # methods which return Bio::PrimarySeq objects
 $feat->seq            # the sequence between start and end
 $feat->entire_seq     # the entire sequence
 $feat->spliced_seq    # the "joined" sequence, when there are
            # multiple sub-locations

 # other useful methods include
 $feat->overlap($other)  # do SeqFeature $feat and SeqFeature $other overlap?
 $feat->contains($other) # is $other completely within $feat?
 $feat->equals($other)   # do $feat and $other completely agree?
 $feat->sub_SeqFeatures  # create/access an array of subsequence features
```

It is worth mentioning that one can also retrieve the start and end positions of a feature using a Bio::LocationI object:

```
 $location = $feat->location # $location is a Bio::LocationI object
 $location->start;           # start position
 $location->end;             # end position
```

This is useful because one can use a Bio::Location::SplitLocationI object in order to retrieve the split coordinates inside the Genbank or EMBL join() statements (e.g. ``CDS join(51..142,273..495,1346..1474)''):

```
 if ( $feat->location->isa('Bio::Location::SplitLocationI') &&
       $feat->primary_tag eq 'CDS' )  {
  foreach $loc ( $feat->location->sub_Location ) {
   print $loc->start,"..",$loc->end,"\n";
  }
 }
```

See the Bio::LocationI manpage and the Bio::Location::SplitLocationI manpage for more information.

If more detailed information is required than is currently available in Seq objects the RichSeq object may be used. It is applicable in particular to database sequences (EMBL, GenBank and Swissprot) with detailed annotations. Sample usage might be:

```
  @secondary   = $richseq->get_secondary_accessions;
```

```
  $division    = $richseq->division;
  @dates       = $richseq->get_dates;
  $seq_version = $richseq->seq_version;
```

See the Bio::Seq::RichSeqI manpage for more details.

III.7.2 Representing sequence annotations (Annotation::Collection)

Much of the interesting description of a sequence can be associated with sequence features but in sequence objects derived from Genbank or EMBL entries there can be useful information in other ``annotation'' sections, such as the COMMENTS section of a Genbank entry. In order to access this information you'll need to create an Annotation::Collection object. For example:

```
 $db = new Bio::DB::GenBank;
 $seqobj = $db->get_Seq_by_acc("NM_125788");
 $ann_coll = $seqobj->annotation;
```

This Collection object is just a container for other specialized objects, and its methods are described in the Bio::Annotation::Collection manpage. You can find the desired object within the Collection object by examining the ``tagnames'':

```
 foreach $ann ($ann_coll->get_Annotations) {
  print "Comment: ",$ann->as_text if ($ann->tagname eq "comment");
 }
```

Other possible tagnames include ``date_changed'', ``keyword'', and ``reference''. Objects with the ``reference'' tagname are Bio::Annotation::Reference objects and represent scientific articles. See the Bio::Annotation::Reference manpage for descriptions of the methods used to access the data in Reference objects. There is also a HOWTO on features and annotation (http://bioperl.org/HOWTOs/html/Feature-Annotation.html).

III.7.3 Representing large sequences (LargeSeq)

Very large sequences present special problems to automated sequence-annotation storage and retrieval projects. Bioperl's LargeSeq object addresses this situation.

A LargeSeq object is a SeqI compliant object that stores a sequence as a series of files in a temporary directory (see sect II.1 or the Bio::SeqI manpage for a definition of SeqI objects). The aim is to enable storing very large sequences (e.g. > 100 MBases) without running out of memory and, at the same time, preserving the familiar bioperl Seq object interface. As a result, from the user's perspective, using a LargeSeq object is almost identical to using a Seq object. The principal difference is in the format used in the SeqIO calls. Another difference is that the user must remember to only read in small chunks of the sequence at one time. These differences are illustrated in the following code:

```
 $seqio = new Bio::SeqIO(-format => 'largefasta',
             -file   => 't/data/genomic-seq.fasta');
 $pseq = $seqio->next_seq();
 $plength = $pseq->length();
 $last_4 = $pseq->subseq($plength-3,$plength);  # this is OK

 # On the other hand, the next statement would
 # probably cause the machine to run out of memory
```

```
 # $lots_of_data = $pseq->seq();  # NOT OK for a large LargeSeq object
```

III.7.4 Representing changing sequences (LiveSeq)

Data files with sequences that are frequently being updated present special problems to automated sequence-annotation storage and retrieval projects. Bioperl's LiveSeq object is designed to address this situation.

The LiveSeq object addresses the need for a sequence object capable of handling sequence data that may be changing over time. In such a sequence, the precise locations of features along the sequence may change. LiveSeq deals with this issue by re-implementing the sequence object internally as a ``double linked chain.'' Each element of the chain is connected to other two elements (the PREVious and the NEXT one). There is no absolute position like in an array, hence if positions are important, they need to be computed (methods are provided). Otherwise it's easy to keep track of the elements with their ``LABELs''. There is one LABEL (think of it as a pointer) to each ELEMENT. The labels won't change after insertions or deletions of the chain. So it's always possible to retrieve an element even if the chain has been modified by successive insertions or deletions.

Although the implementation of the LiveSeq object is novel, its bioperl user interface is unchanged since LiveSeq implements a PrimarySeqI interface (recall PrimarySeq is the subset of Seq without annotations or SeqFeatures - see section II.1 or the Bio::PrimarySeq manpage). Consequently syntax for using LiveSeq objects is familiar although a modified version of SeqIO called Bio::LiveSeq::IO::Bioperl needs to be used to actually load the data, e.g.:

```
 $loader = Bio::LiveSeq::IO::BioPerl->load(-db   => "EMBL",
                       -file => "t/data/factor7.embl");
 $gene = $loader->gene2liveseq(-gene_name => "factor7");
 $id = $gene->get_DNA->display_id ;
 $maxstart = $gene->maxtranscript->start;
```

See the Bio::LiveSeq::IO::BioPerl manpage for more details.

III.7.5 Representing related sequences - mutations, polymorphisms (Allele, SeqDiff)

A Mutation object allows for a basic description of a sequence change in the DNA sequence of a gene. The Mutator object takes in mutations, applies them to a LiveSeq gene and returns a set of Bio::Variation objects describing the net effect of the mutation on the gene at the DNA, RNA and protein level.

The objects in Bio::Variation and Bio::LiveSeq directory were originally designed for the ``Computational Mutation Expression Toolkit'' project at European Bioinformatics Institute (EBI). The result of using them to mutate a gene is a holder object, 'SeqDiff', that can be printed out or queried for specific information. For example, to find out if restriction enzyme changes caused by a mutation are exactly the same in DNA and RNA sequences, we can write:

```
 use Bio::LiveSeq::IO::BioPerl;
 use Bio::LiveSeq::Mutator;
 use Bio::LiveSeq::Mutation;

 $loader = Bio::LiveSeq::IO::BioPerl->load(-file => "$filename");
 $gene = $loader->gene2liveseq(-gene_name => $gene_name);
 $mutation = new Bio::LiveSeq::Mutation (-seq =>'G',
                       -pos => 100 );
```

```
$mutate = Bio::LiveSeq::Mutator->new(-gene      => $gene,
                  -numbering => "coding"  );
$mutate->add_Mutation($mutation);
$seqdiff = $mutate->change_gene();
$DNA_re_changes = $seqdiff->DNAMutation->restriction_changes;
$RNA_re_changes = $seqdiff->RNAChange->restriction_changes;
$DNA_re_changes eq $RNA_re_changes or print "Different!\n";
```

For a complete working script, see the change_gene.pl script in the examples/liveseq directory. For more details on the use of these objects see the Bio::LiveSeq::Mutator manpage and the Bio::LiveSeq::Mutation manpage as well as the original documentation for the ``Computational Mutation Expression Toolkit'' project at http://www.ebi.ac.uk/mutations/toolkit/.

III.7.6 Incorporating quality data in sequence annotation (SeqWithQuality)

SeqWithQuality objects are used to describe sequences with very specific annotations - that is, data quality annotations. Data quality information is important for documenting the reliability of base calls in newly sequenced or otherwise questionable sequence data. The quality data is contained within a Bio::Seq::PrimaryQual object. Syntax for using SeqWithQuality objects is as follows:

```
# first, make a PrimarySeq object
$seqobj = Bio::PrimarySeq->new( -seq => 'atcgatcg',
                -id  => 'GeneFragment-12',
                -accession_number => 'X78121',
                -alphabet => 'dna');
# now make a PrimaryQual object
$qualobj = Bio::Seq::PrimaryQual->new(-qual => '10 20 30 40 50 50 20 10',
                   -id   => 'GeneFragment-12',
                   -accession_number => 'X78121',
                   -alphabet => 'dna');
# now make the SeqWithQuality object
$swqobj = Bio::Seq::SeqQithQuality->new(-seq  => $seqobj,
                    -qual => $qualobj);
# Now we access the sequence with quality object
$swqobj->id(); # the id of the SeqWithQuality object may not match the
        # id of the sequence or of the quality
$swqobj->seq(); # the sequence of the SeqWithQuality object
$swqobj->qual(); # the quality of the SeqWithQuality object
```

A SeqWithQuality object is created automatically when phred output, a *phd file, is read by SeqIO, e.g.

```
$seqio = Bio::SeqIO->new(-file=>"my.phd",-format=>"phd");
# or just 'Bio::SeqIO->new(-file=>"my.phd")'
$seqWithQualObj = $seqio->next_seq;
```

See the Bio::Seq::SeqWithQuality manpage for a detailed description of the methods, the Bio::Seq::PrimaryQual manpage, and the Bio::SeqIO::phd manpage.

III.7.7 Sequence XML representations - generation and parsing (SeqIO::game, SeqIO::bsml)

The previous subsections have described tools for automated sequence annotation

by the creation of an object layer on top of a traditional database structure. XML takes a somewhat different approach. In XML, the data structure is unmodified, but machine readability is facilitated by using a data-record syntax with special flags and controlled vocabulary.

In order to transfer data with XML in biology, one needs an agreed upon a vocabulary of biological terms. Several of these have been proposed and bioperl has at least some support for three: GAME, BSML and AGAVE.

Once a vocabulary is agreed upon, it becomes possible to convert sequence XML sequence features can be turned into bioperl Annotation and SeqFeature objects. Conversely Seq object features and annotations can be converted to XML so that they become available to any other systems. Typical usage with GAME or BSML are shown below. No special syntax is required by the user. Note that some Seq annotation will be lost when using XML in this manner since generally XML does not support all the annotation information available in Seq objects.

```
$str = Bio::SeqIO->new(-file   => 't/data/test.game',
            -format => 'game');
$seq = $str->next_primary_seq();
$id = $seq->id;
@feats = $seq->all_SeqFeatures();
$first_primary_tag = $feats[0]->primary_tag;

$str = Bio::SeqIO->new(-file   => 'bsmlfile.xml',
            -format => 'bsml');
$seq = $str->next_primary_seq();
$id = $seq->id;
@feats = $seq->all_SeqFeatures();
$first_primary_tag = $feats[0]->primary_tag;
```

III.7.8 Representing Sequences using GFF (Bio:DB:GFF)

Another format for transmitting machine-readable sequence-feature data is the Genome Feature Format (GFF). This file type is well suited to sequence annotation because it allows the ability to describe entries in terms of parent-child relationships (see http://www.sanger.ac.uk/software/GFF for details). Bioperl includes a parser for converting between GFF files and SeqFeature objects. Typical syntax looks like:

```
$gffio = Bio::Tools::GFF->new(-fh => \*STDIN, -gff_version => 2);
 # loop over the input stream
while ($feature = $gffio->next_feature()) {
 # do something with feature
}
$gffio->close();
```

Further information can be found at the Bio::Tools::GFF manpage. Also see examples/tools/gff2ps.pl, examples/tools/gb_to_gff.pl, and the scripts in scripts/Bio-DB-GFF. Note: this module shouldn't be confused with the module Bio::DB::GFF which is for implementing relational databases when using bioperl-db.

III.8 Manipulating clusters of sequences (Cluster, ClusterIO)

Sequence alignments are not the only examples in which one might want to manipulate a group of sequences together. Such groups of related sequences are generally referred to as clusters. Examples include Unigene clusters and gene clusters resulting from clustering algorithms being applied to microarray data.

The bioperl Cluster and ClusterIO modules are available for handling sequence clusters. Currently, cluster input/output modules are available only for Unigene clusters. To read in a Unigene cluster (in the NCBI XML format) and then extract individual sequences for the cluster for manipulation might look like this:

```
my $stream = Bio::ClusterIO->new(-file => "Hs.data", -format => "unigene");
while ( my $in = $stream->next_cluster ) {
  print $in->unigene_id() . "\n";
  while ( my $sequence = $in->next_seq ) {
    print $sequence->accession_number . "\n";
  }
}
```

See the Bio::Cluster::UniGene manpage for more details.

III.9 Representing non-sequence data in Bioperl: structures, trees and maps

Though bioperl has its roots in describing and searching nucleotide and protein sequences it has also branched out into related fields of study, such as protein structure, phylogenetic trees and genetic maps.

III.9.1 Using 3D structure objects and reading PDB files (StructureI, Structure::IO)

A StructureIO object can be created from one or more 3D structures represented in Protein Data Bank, or pdb, format (see http://www.rcsb.org/pdb for details).

StructureIO objects allow access to a variety of related Bio:Structure objects. An Entry object consist of one or more Model objects, which in turn consist of one or more Chain objects. A Chain is composed of Residue objects, which in turn consist of Atom objects. There's a wealth of methods, here are just a few:

```
$structio = Bio::Structure::IO->new( -file => "1XYZ.pdb");
$struc = $structio->next_structure; # returns an Entry object
$pseq = $struc->seqres;             # returns a PrimarySeq object, thus
$pseq->subseq(1,20);                # returns a sequence string
@atoms = $struc->get_atoms($res);   # Atom objects, given a Residue
@xyz = $atom->xyz;                  # the 3D coordinates of the atom
```

These lines show how one has access to a number of related objects and methods. For examples of typical usage of these modules, see the scripts in the examples/ structure subdirectory. Also see the Bio::Structure::IO manpage, the Bio::Structure::Entry manpage, the Bio::Structure::Model manpage, the Bio::Structure::Chain manpage, the Bio::Structure::Residue manpage, and the Bio::Structure::Atom manpage for more information.

III.9.2 Tree objects and phylogenetic trees (Tree::Tree, TreeIO, PAML)

Bioperl Tree objects can store data for all kinds of computer trees and are intended especially for phylogenetic trees. Nodes and branches of trees can be individually manipulated. The TreeIO object is used for stream I/O of tree objects. Currently only phylip/newick tree format is supported. Sample code might be:

```
$treeio = new Bio::TreeIO( -format => 'newick', -file => $treefile);
$tree = $treeio->next_tree;             # get the tree
@nodes = $tree->get_nodes;              # get all the nodes
$tree->get_root_node->each_Descendent;  # get descendents of root node
```

See the Bio::TreeIO manpage and the Bio::Tree::Tree manpage for details.

Using the Bio::Tools::Phylo::PAML module one can also parse the results of the PAML tree-building programs codeml, baseml, basemlg, codemlsites and yn00. See the Bio::Tools::Phylo::PAML manpage or the PAML HOWTO (http://bioperl.org/HOWTOs/html/PAML.html) for more information.

III.9.3 Map objects for manipulating genetic maps (Map::MapI, MapIO)

Bioperl map objects can be used to describe any type of biological map data including genetic maps, STS maps etc. Map I/O is performed with the MapIO object which works in a similar manner to the SeqIO, SearchIO and similar I/O objects described previously. In principle, Map I/O with various map data formats can be performed. However currently only mapmaker format is supported. Manipulation of genetic map data with Bioperl Map objects might look like this:

```
$mapio = new Bio::MapIO(-format => 'mapmaker', -file => $mapfile);
$map = $mapio->next_map;  # get a map
$maptype =  $map->type ;
foreach $marker ( $map->each_element ) {
 $marker_name = $marker->name ;  # get the name of each map marker
}
```

See the Bio::MapIO manpage and the Bio::Map::SimpleMap manpage for more information.

III.9.4 Bibliographic objects for querying bibliographic databases (Biblio)

Bio::Biblio objects are used to query bibliographic databases, such as MEDLINE. The associated modules are built to work with OpenBQS-compatible databases (see http://industry.ebi.ac.uk/openBQS). A Bio::Biblio object can execute a query like:

```
my $collection = $biblio->find ('brazma', 'authors');
while ( $collection->has_next ) {
  print $collection->get_next;
}
```

See the Bio::Biblio manpage, the scripts/biblio/biblio.PLS script, or the examples/biblio/biblio_examples.pl script for more information.

III.9.5 Graphics objects for representing sequence objects as images (Graphics)

A user may want to represent sequence objects and their SeqFeatures graphically. The Bio::Graphics::* modules use Perl's GD.pm module to create a PNG or GIF image given the SeqFeatures (Section III.7.1) contained within a Seq object.

These modules contain numerous methods to dictate the sizes, colors, labels, and line formats within the image. For information see the excellent Graphics-HOWTO (http://bioperl.org/HOWTOs/html/Graphics-HOWTO.html) or in the docs/howto subdirectory. Additional documentation can be found in the Bio::Graphics manpage, the Bio::Graphics::Panel manpage, and in the scripts in the examples/biographics/ and scripts/graphics directories in the Bioperl package.

III.10 Bioperl alphabets

Bioperl modules use the standard extended single-letter genetic alphabets to represent nucleotide and amino acid sequences.

In addition to the standard alphabet, the following symbols are also acceptable

in a biosequence:

? (a missing nucleotide or amino acid)
- (gap in sequence)

III.10.1 Extended DNA / RNA alphabet

(includes symbols for nucleotide ambiguity)

Symbol	Meaning	Nucleic Acid
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
U	U	Uracil
M	A or C	
R	A or G	
W	A or T	
S	C or G	
Y	C or T	
K	G or T	
V	A or C or G	
H	A or C or T	
D	A or G or T	
B	C or G or T	
X	G or A or T or C	
N	G or A or T or C	

IUPAC-IUB SYMBOLS FOR NUCLEOTIDE NOMENCLATURE:
Cornish-Bowden (1985) Nucl. Acids Res. 13: 3021-3030.

III.10.2 Amino Acid alphabet

Symbol	Meaning
A	Alanine
B	Aspartic Acid, Asparagine
C	Cystine
D	Aspartic Acid
E	Glutamic Acid
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
K	Lysine
L	Leucine
M	Methionine
N	Asparagine
P	Proline

Q	Glutamine
R	Arginine
S	Serine
T	Threonine
V	Valine
W	Tryptophan
X	Unknown
Y	Tyrosine
Z	Glutamic Acid, Glutamine
*	Terminator

IUPAC-IUP AMINO ACID SYMBOLS:
Biochem J. 1984 Apr 15; 219(2): 345-373
Eur J Biochem. 1993 Apr 1; 213(1): 2

distribution:**配布物**