BioRubyの開発と遺伝子情報解析への応用

大阪大学 遺伝情報実験センター ゲノム情報解析分野　後藤 直久　E-mail: ngoto@gen-info.osaka-u.ac.jp

要旨

生物に関する情報を収集したデータベースは現在既に719件以上存在する。また、実験データやデータベース上のデータを解析するソフトウェアも数多く存在する。これらのデータベースやソフトウェアを有機的に組み合わせた解析を行うことにより、新たな生物学的知見を得ることが期待できる。しかし、そのために必要なソフトウェア環境はまだ十分に整備されていないのが現状である。このため、私はBioRubyプロジェクトに参加し、開発の一翼を担ってきた。BioRubyはバイオインフォマティクスに必要な機能や環境をオブジェクト指向スクリプト言語Rubyによって実装したライブラリである。塩基・アミノ酸配列に対する基本的な処理や解析、データベースのデータ処理、解析ソフトウェアの結果処理など様々な機能を備えている。現在、GenBankなど26種類のデータベースフォーマットと BLAST など15種類のソフトウェアに対応している。これにより、データベースや解析ソフトウェアを複数組み合わせた解析を行うスクリプトを短時間で容易に書くことができる。このポスターでは、BioRubyの特徴・利点や応用について、私が開発に携わった部分を中心に報告する。

BioRubyの概要

バイオインフォマティクスに必要な機能や環境をオブジェクト指向スクリプト言語Rubyによって実装したライブラリ

フリーソフトウェア
☒ ・誰でも自由にコピーや配布ができる
☒ ・ソースを公開しており改造も自由

オープンな開発体制
☒ ・インターネットを活用
☒ ☒ ☒ ☒ http://bioruby.org/

現在の状況

バージョン	0.6.2
ファイル数	130以上
合計行数	37,000行以上
開発に直接関与した人	9名(うち日本国外2名)

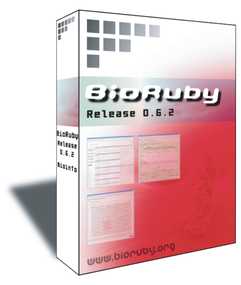
Rubyとは？

・オブジェクト指向スクリプト言語
・Perlと似ているが、より簡潔な文法

オブジェクト指向
☒ ・データ構造を記述しやすい
☒ ・データ構造と処理内容を一括管理可能
☒ ☒ →生物学では大量の構造化されたデータを
☒ ☒ 扱う必要があるため、特に有利
スクリプト言語
☒ ・コンパイル不要ですぐに実行可能

他言語による先行プロジェクト

□ □ Perl□ □ □ BioPerl
□ □ Python□ □ BioPython
□ □ Java□ □ □ BioJava
・言語により得意分野が異なるため、共存可能
・Open Bioinformatics Foundation(OBF) を結成
・データ入出力の標準化(OBDA)



Classes in BioRuby

(Some classes are not listed here.)　NEW : newly added in 2004　updated : significant improvement in 2004

Basic data structures

updated	Bio::Sequence::NA	Nucleic acid sequence
updated	Bio::Sequence::AA	Amino acid sequence
	Bio::Locations	Location
	Bio::Features	Annotations
	Bio::References	Literatures
	Bio::Relation	Binary relation
	Bio::Pathway	Graph / Pathway
● updated	Bio::Alignment	Alignment
updated	Bio::CodonTable	Genetic codes

Wrappers and parsers for bioinformatics tools

updated	Bio::Blast	BLAST (similarity search)
	Bio::Fasta	FASTA (similarity search)
	Bio::HMMER	HMMER (similarity search)
● updated	Bio::ClustalW	CLUSTAL W (multiple alignment)
●	Bio::MAFFT	MAFFT (multiple alignment)
updated	Bio::PSORT	PSORT (protein subcellular localization)
	Bio::TargetP	TargetP (signal peptides)
	Bio::SOSUI	SOSUI (transmembrane helix)
	Bio::TMHMM	TMHMM (transmembrane helix)
updated	Bio::GenScan	GenScan (gene finding)
	Bio::EMBOSS	EMBOSS (analysis package)
● NEW	Bio::Sim4	Sim4 (genomic mapping)
● NEW	Bio::Spidey	Spidey (mRNA-to-genomic alignment)
● NEW	Bio::Blat	BLAT (BLAST-like alignment tool)

Databases and sequence file formats

● updated	Bio::FastaFormat	FASTA format
updated	Bio::GenBank	GenBank / DDBJ
	Bio::RefSeq	RefSeq
updated	Bio::EMBL	EMBL
updated	Bio::SPTR	SwissProt and TrEMBL
●	Bio::NBRF	PIR
● updated	Bio::PDB	Protein Data Bank
updated	Bio::PROSITE	PROSITE motif
updated	Bio::AAindex	AAindex
updated	Bio::GO	Gene Ontology
	Bio::GFF	General Feature Format
updated	Bio::KEGG::GENES	KEGG GENES
updated	Bio::KEGG::GENOME	KEGG GENOME
updated	Bio::KEGG::KO	KEGG Orthology
updated	Bio::KEGG::ENZYME	KEGG ENZYME
updated	Bio::KEGG::COMPOUND	KEGG COMPOUND
NEW	Bio::KEGG::GLYCAN	KEGG GLYCAN
NEW	Bio::KEGG::REACTION	KEGG REACTION
	Bio::KEGG::CELL	KEGG CELL
	Bio::KEGG::Microarrays	KEGG EXPRESSION
	Bio::KEGG::BRITE	Biomolecular Reactions
	Bio::LITDB	Protein/peptide literatures
	Bio::TRANSFAC	Transcription Factor database
●	Bio::FANTOM	Functional annotation of mouse
updated	Bio::MEDLINE	MEDLINE bibliographic database

File, network, and database I/O

	Bio::Registry	OBDA Registry service
	Bio::SQL	OBDA BioSQL RDB schema
	Bio::Fetch	OBDA BioFetch via HTTP
●	Bio::FlatFileIndex	OBDA flat file indexing system
● updated	Bio::FlatFile	Flat file reader with data format autodetection
	Bio::PubMed	NCBI PubMed service
updated	Bio::DAS	Distributed Annotation System
updated	Bio::KEGG::API	KEGG web services
updated	Bio::DDBJ::XML	DDBJ web services

UnitTests

NEW test/all_tests.rb

Command-line applications (installed as br_*.rb)

biogetseq	OBDA Resistry sequence retrieval
biofetch	OBDA BioFetch sequence retrieval
● bioflat	Creates/searches ODBA flat file index
NEW pmfetch	Reference utility

Sample / Miscellaneous

goslim.rb	GO slim histgram
tdiary.rb	Plug-in for tDiary
NEW biofetch.rb	BioFetch server

History

(1995/12/21 Ruby language (0.95) released)
2000/11/21 Bio**Ruby** project started
2001/03/18 mailing list started
2001/06/21 bioruby-0.1
2001/07/19 ISMB/BOSC2001 lightning talks
2001/10/24 bioruby-0.3 w/ CVS repository
2002/01/02 BioHackathon w/ BioFetch server
2002/12/11 MBSJ2002 poster
2002/12/16 GIW2002 software demonstration
2003/01/28 bioruby-0.4.0 released
2003/02/17 BioHackathon 2003
2003/06/25 bioruby-0.5.0 released
2003/06/27 ISMB/BOSC2003 talks
2003/07/16 bioruby-0.5.1 released
2003/08/22 bioruby-0.5.2 released
2003/10/13 bioruby-0.5.3 released
2003/12/10 MBSJ2003 poster
2003/12/15 GIW2003 poster
2004/01/03 CVS repository moved to Open-Bio
2004/07/29 ISMB/BOSC2004 talks
2004/08/24 bioruby-0.6.0 released
2004/08/25 bioruby-0.6.1 released
2004/12/13 bioruby-0.6.2 released
2004/12/13 GIW2004 software demonstration

高速なBLASTパーサの実装

ホモロジー検索ソフトウェアBLASTの実行結果を読み込み、結果の整理や解析などの後処理を加えることは広く行われている。近年のデータベース容量の増大に伴い、BLAST結果のサイズは大きくなることが多い。必然的に後処理にも長時間を要することが多くなり、高速化の重要性は高い。しかし、BLAST自体は並列化などにより高速化が図られているが、結果の後処理に関しては高速化の検討が不十分な場合も多い。

そこで私は、処理速度を重視したBLASTパーサをBioRubyに実装した。高速化のため、BLAST結果出力を大雑把に分解した後で要求された部分だけを詳細に解釈する遅延評価を導入した。さらに、Ruby標準の高速文字列スキャナ(文字列の特定のパターンを検索する仕組み)を使用するなど、様々な高速化の工夫を行った。その結果、広く使われているBioPerlのBLASTパーサと比較すると、処理の内容により異なるが、5倍から20倍以上の処理速度を示した。

このパーサはNCBI BLASTのデフォルト出力形式(-m 0 オプション)を対象とし、通常のBLASTだけでなくPSI/PHI-BLASTにも対応している。拡張モジュールを使用せずRuby(1.8.0以降)本体の機能だけを使用しているため、BioRubyをインストールするだけで利用可能である。

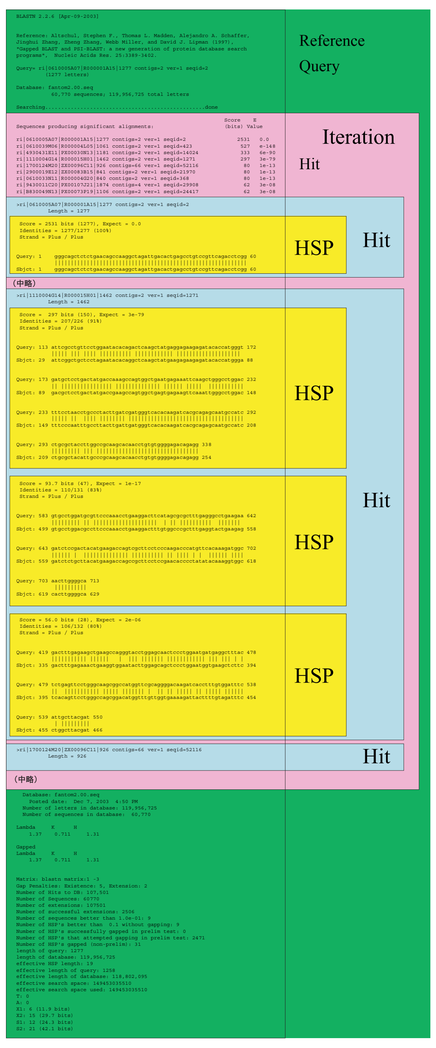
考察

BioRuby, BioPerl それぞれのBLASTパーサと、最近発表された高速なBLASTパーサライブラリZerg(*)の間で機能および速度の比較を行ったのが右表である。BioRubyのBLASTパーサはBioPerlとほぼ同等の機能を持つが、20倍以上の速度を示している。ZergはBioRubyのBLASTパーサよりさらに約15倍高速であるが、そのかわり一部の機能に未対応である。また、Zergはコンパイルやインストールが必要であるため、BioRubyやBioPerlと比較すると手間がかかる場合が多い。

□ このように、BioRubyのBLASTパーサは、最速ではないものの、スクリプト言語標準搭載の機能だけを使用して書かれたパーサとしては非常に高速である。BLAST結果処理にBioRubyを使用することにより、スクリプト言語の柔軟性と処理の高速性の両方を得ることが可能で、解析の効率を上げることが可能であると期待できる。

* Paquola,A.C.M., et al.(2003) Zerg: a very fast BLAST parser library, *Bioinformatics*, **19**, 1035-1036.

BLAST結果出力の構造



BioRubyのクラス

Bio::Blast::Default::Report クラス
BLASTの結果全体。Iterationクラスのインスタンスを内部に保持。

Bio::Blast::Default::Report::Iteration クラス
PSI-BLASTの繰り返し検索1回分の結果を格納するクラス。通常のBLASTでも検索結果の格納に使用。Hitクラスのインスタンスを内部に保持。

Bio::Blast::Default::Report::Hit クラス
検索にヒットした配列に関する情報を格納。HSPクラスのインスタンスを内部に保持。

Bio::Blast::Default::Report::HSP クラス
HSPに関する情報を格納。BLASTによる相同性検索結果の最小単位。

※HSP
High-Scoring Segment Pairの略。閾値を超える類似性が検出された部分配列のペア。

機能比較

	BioRuby (0.5.3)	BioPerl (1.2.1)	Zerg (1.0.3)
使用言語	Ruby	Perl	C (Perl)
NCBI BLAST対応	○	○	○*
HSPのアライメント取得	○	○	×
PSI-BLAST対応	○	○	×
WU-BLAST対応	○*	○	×

* 統計情報の一部には未対応

速度比較

Pentium3 1GHz, メモリ1GB, HDD 27GB, Linux 2.4.18 のマシン上でベンチマークプログラムを10回動作させたときの平均所要時間と処理速度およびBioPerlを基準とした速度比を示した。ベンチマークに使用したデータのサイズは以下のとおりである。

□ BLASTN: 104,921,408バイト, 8014エントリ
□ BLASTX: 104,858,552バイト, 16013エントリ

	BLASTN 所要時間 (s)	BLASTN S.D.	BLASTN 速度 (MB/s)	BLASTN 速度比	BLASTX 所要時間 (s)	BLASTX S.D.	BLASTX 速度 (MB/s)	BLASTX 速度比
BioRuby (Ruby1.8.0)	35.325	0.032	2.83	21.3	44.821	0.084	2.23	23.9
BioRuby (Ruby1.6.7)	49.724	0.048	2.01	15.1	79.857	0.083	1.25	13.4
BioPerl (Perl5.6.1)	751.067	2.915	0.133	1.0	1070.301	5.098	0.0934	1.0
Zerg-C	2.437	0.002	41.1	308	2.685	0.001	37.2	399
Zerg-Perl	2.605	0.002	38.4	288	2.977	0.002	33.6	366
Zerg-Perl2 (Perlオブジェクトを作成)	36.687	0.051	2.73	20.5	57.675	0.222	1.73	18.6

応用例・サンプルプログラム

BLAST結果を読み込み、クエリー, ヒットした配列の名前, アライメント長, e-value, ビットスコアをタブ区切りでHSP毎に表示するプログラム。

```
#!/usr/bin/env ruby
require 'bio'
ff = Bio::FlatFile.auto(ARGF)
print [ 'Query', 'Subject', 'AlignLen',
  'eValue', 'BitScore' ].join("\t"), "\n"
ff.each do |r|
  qdef = r.query_def.split[0]
  r.each_hit do |hit|
    hdef = hit.definition.split[0]
    hit.each do |hsp|
      alen   = hsp.align_len
      evalue = hsp.evalue
      bscore = hsp.bit_score
      print [ qdef, hdef, alen, evalue,
bscore                          ].join("\t"),
"\n"
    end
  end
end
ff.close
```

BLAST結果を読み込み、まったくヒットしていないか、一定の条件以上のヒットが見られないクエリーを表示するプログラム。

```
#!/usr/bin/env ruby
require 'bio'
coverage = 0.5
evalue_max = 1

def min(a,b);     (a > b) ? b : a; end
def abssub(b, a); (b >= a) ? b - a : a - b; end

ff = Bio::FlatFile.open(Bio::Blast::Default::Report, STDIN)
ff.each do |result|
  flag = nil
  STDERR.print "query: #{result.query_def}\nlength: #{result.query_len}\n"
  result.each_hit do |hit|
    qstr = ' ' * result.query_len
    next if hit.hsps.size <= 0
    sstr = ' ' * hit.len
    hit.each do |hsp|
      e = hsp.evalue
      e = '1' + e if e =~ /^e/
      e = e.to_f
      if (e <= evalue_max) then
        hspqstart = min(hsp.query_from, hsp.query_to) - 1
        hspqlen = abssub(hsp.query_from, hsp.query_to) + 1
        qstr[hspqstart, hspqlen] = ('X' * hspqlen)
        hspsstart = min(hsp.hit_from, hsp.hit_to) - 1
        hspslen = abssub(hsp.hit_from, hsp.hit_to) + 1
        sstr[hspsstart, hspslen] = ('X' * hspslen)
      end
    end
    qstr.tr!(' ', '')
    sstr.tr!(' ', '')
    slen = result.query_len
```

（右上へ続く）

（左からの続き）

```
      flag = true if (qstr.length >= result.query_len * coverage) or
         (sstr.length >= hit.len * coverage)
      break if flag
    end
    if !flag then
      print result.query_def.split[0], "\n"
    end
  end
  #ff.close
```


Acknowledgements

□ BioRuby Developers
□ □ □ Toshiaki Katayama
□ □ □ Mitsuteru Nakao
□ □ □ Yoshinori Okuji
□ □ □ Shuichi Kawashima
□ □ □ Masumi Itoh
□ □ □ Alex Gutteridge
□ □ □ Moses Hohman
□ □ and some other contributors on the net.
□ Open Bio* community