Bioperl

Thierry Lecroq

Université de Rouen
FRANCE

Plan

1. Généralités sur Bioperl
2. Manipuler des fichiers de séquences
3. Collecter des séquences
4. Séquences et données associées
5. Blast

Bioperl

- Ensemble de modules pour manipuler des objets biologiques (séquences, annotations, ...)
- Accéder à des séquences
  - Depuis des fichiers ou des bases de données
  - Locales ou distantes
  - Transformer les formats des séquences
- Manipuler les séquences
- Créer / Manipuler des alignements de séquences
- Rechercher des gènes ou des structures dans des séquences
- Annoter les séquences

Exemple de Module Bio::Seq

- Bio::Seq : stocker et manipuler des séquences et des annotations
- Un objet de type Bio::Seq
  - contient des données : la séquence elle-même, le type de séquence (dna, rna, protein), le numéro d'accession, des *features*, des annotations, etc ...
  - fournit des méthodes pour travailler sur cet objet : subseq(), revcom(), translate (), species (), annotations (), get_SeqFeatures()

Modules, objets, méthodes, ...

- Module Perl : ensemble de fonctionnalités
  - Peut être juste un ensemble de fonctions (librairie)
  - Le plus souvent, c'est un objet
- Objet (au sens informatique)
  - Structure de données + des méthodes (= fonctions) pour manipuler ces données
  - Utilité : séparer l'implémentation et l'interface
  - Implémentation : la façon dont les données sont organisées
  - Interface : la façon d'accéder à ces données
- Ex Bio::Seq
  - On sait que cet objet contient des annotations
  - On ne sait pas comment sont stockées ces annotions (liste, tableau, ... ?)
  - On utilise la méthode annotations() pour y accéder
  - La méthode de stockage (= l'implémentation) peut changer
  - Les méthodes d'accès (= l'interface) ne changeront pas

Utilisation des modules objet

```
#!/usr/bin/perl
use XYZ;
$objet1 = XYZ->new($argument1, $argument2);
$var1 = $objet1->methode1();
$var2 = $objet1->methode2(argument1, argument2);
$var3 = $objet1->methode3(argument1, argument2);
$var4 = $objet1->methode4(argument1, argument2);
```

Important : connaître le type de la valeur retournée par une méthode
Cf la documentation de chaque module

Utilisation des modules

```
#!/usr/bin/perl
use Bio::Seq;
$seq1 = Bio::Seq->new(-id => 'ma seq', -seq => 'ATTCGTTACCGGAC
$id = $seq1->id() ; # chaine
$dna = $seq1->seq() ; # chaine
$seq2 = $seq1->revcom() ; # Bio::Seq
$seq3 = $seq2->translate() ; # Bio::Seq
$seq4 = $seq3->subseq(3, 6) ; # chaine
$seq5 = $seq1->revcom()->translate()->subseq(3, 6) ;

print "seq1=", $seq1->seq, " seq2=", $seq2->seq, "\n";
print "seq3=", $seq3->seq, " seq4=$seq4 seq5=$seq5\n";
print ref($seq1), "\n";
```

```
seq1=ATTCGTTACCGGAGTCTA seq2=TAGACTCCGGTAACGAAT
seq3=*TPVTN seq4=PVTN seq5=PVTN
Bio::Seq
```

Modules (ou Packages) Bioperl

- Core package
  - *Parsers* pour divers formats
  - Accès à des bases de données
  - Manipulation de séquences
  - Manipulation d'alignements
- Run package
  Wrappers pour divers programmes
  - Blast
  - Genescan
  - HMMER
  - Sim4
- Bio :: Biblio
- Bio :: Graphics

Parser, Wrapper ?

- *Parser* = conversion entre un format de fichier et un objet (i.e. une structure de données)
  Permet de
  - lire/écrire des données dans un fichier selon un format spécifique (ex genbank, EMBL, fasta)
  - stocker ces données dans un format interne (indépendant du format de fichier)
- *Wrapper* = encapsulation d'un programme existant (ex Blast)
  - Permet de lancer blast en cachant les détails techniques

Où trouver des informations ?

- Biblio
  - Bioperl course, Catherine Letondal
  - BioPerl Tutorial, Peter Schattner
  - Mastering Perl for Bioinformatics, James D. Tisdall (chapitre 9)
- Documentation en ligne
- Documentation de la distribution

Documentation en ligne

- Bioperl site `http://bioperl.org`
  - HOWTO Ex : Beginners, SeqIO, SearchIO, Feature-Annotation, Graphics, local Databases, Eutilities, Getting Genomic Sequences
  - Bioperl Docs `http://doc.bioperl.org/bioperl-live/` : Documentation complète de chaque module avec ses méthodes
- CPAN `http://search.cpan.org/`
  - `http://search.cpan.org/~cjfields/BioPerl-1.6.901/Bio/Seq.pm`

Documentation de la distribution

- man Bio::Seq = perldoc Bio::Seq
- man -k <expression>
  - Recherche dans la description courte et le nom des pages de manuel
  - Exemple pour man -k GFF

```
bed2gff3.pl (1p) - Convert UCSC Genome Browser-format gene files i
Bio::Graphics::Browser2::GFFhelper (3pm) - - Helps gbrowse plugins
Bio::Graphics::Glyph::gene (3pm) - A GFF3-compatible gene glyph
Bio::Graphics::Glyph::rainbow_gene (3pm) - A GFF3-compatible gene
load_genbank.pl (1p) - Load a Bio::DB::GFF database from GENBANK f
ucsc_genes2gff.pl (1p) - Convert UCSC Genome Browser-format gene f
Bio::DB::GFF (3pm) - - Storage and retrieval of sequence annotatio
Bio::DB::GFF::Aggregator::alignment (3pm) - - Alignment aggregator
Bio::DB::GFF::Feature (3pm) - - A relative segment identified by a
Bio::DB::GFF::Homol (3pm) - - A segment of DNA that is homologous
bp_bulk_load_gff (1p) - Bulk-load a Bio::DB::GFF database from GFF
bp_genbank2gff3 (1p) - - Genbank->gbrowse-friendly GFF3
bp_generate_histogram (1p) - - Generate a histogram of Bio::DB::GF
```

(1) programme, (3) fonction ou module

Scripts & exemples

- Ensemble de scripts développés par des contributeurs
  `http://www.bioperl.org/`

Par où commencer ?

- Bio :: Perl + bptutorial . pl : utilisation sans objets
  - Functional access to BioPerl for people who don't know objects
  - Démontre les possibilités
  - Programme exécutable + documentation
  - Limite : ne peut pas être étendu
    Cf chap 9 Mastering Perl for Bioinformatics
- Utiliser les objets
  - Howto
  - Documentation de chaque module
  - Tutorial sur les types d'objets
    Cf BioPerl Tutorial, Peter Schatter

Plan

- Lire un fichier fasta
- Manipuler des séquences
- Lire et écrire des fichiers fasta
- Chercher et récupérer des séquences
  - Module genbank, NCBI Entrez utilities
- Lire des fichiers genbank
  - Annotation, *features*
- Écrire des fichiers GFF
- Parser des sorties blast

Plan

1. Généralités sur Bioperl
2. Manipuler des fichiers de séquences
3. Collecter des séquences
4. Séquences et données associées
5. Blast

Manipuler des fichiers de séquences

- Créer une séquence et afficher sa longueur
- Lire une séquence d'un fichier fasta
- Lire toutes les séquences d'un fichier fasta
- Lire 3 fichiers fasta
- Lire tous les fichiers fasta passés en argument
- + concaténer les séquences dans un nouveau fichier fasta
- + filtrer les séquence $> 500bp$
- + filtrer les séquences $> 500bp$ ET avec un identifiant contenant TEST

```
use Bio::Seq;
$seq = Bio::Seq->new(-id => 'ma sequence',
-seq => 'ATTCGTTACCGGAGTCTA');
$seq->display_id(); # the human readable id of the sequence
$seq->id() ; # idem
$seq->primary_id() ; # a unique id for this sequence
$seq->accession_number() ; # the accession number
$seq->alphabet(); # one of dna,rna,protein
$seq->seq(); # the dna/rna/protein sequence
$seq->length() ; # number of symbols in the sequence
$seq->subseq(5,10); # part of the sequence as a string
$seq->trunc(5,10); # truncation from 5 to 10 as new object
$seq->revcom(); # reverse complements sequence
$seq->translate(); # translation of the sequence
```

Toutes les méthodes :
http://bioperl.org/howtos/Beginners_HOWTO.html#item13

Exercice : p1.pl

Objectif : utiliser Bio::Seq

- Créer une séquence
- Afficher son nom
- Afficher sa longueur
- Utiliser les méthodes id() et length() de ce module pour obtenir les info sur la séquence

Lire un fichier fasta

- Module Bio::SeqIO : parsers pour différents formats de fichiers
- 42 formats : fasta, embl, fastq, genbank, interpro, kegg, locuslink, pir, qual, swissprot, etc...
- `http://bioperl.org/howtos/Beginners_HOWTO.html#item8`
- Format : minuscules / majuscules équivalentes : 'FASTA', 'Fasta' and 'fasta' OK

```
use Bio::SeqIO ;
$in = Bio::SeqIO->new(-file => "f1.fasta", -format => Fasta);
$seq = $in->next_seq():
print $seq->id() ;
$in->close();
```

Exercice : p2.pl

Objectif : Utiliser Bio::SeqIO

- Lire le fichier f1.fasta
- Afficher le nom et la longueur de la 1ère séquence
- Utiliser la fonction next_seq() du module Bio::SeqIO pour lire le fichier
- Utiliser les méthodes id() et length() du module Bio::Seq pour obtenir les info sur la séquence

Parcourir tout le fichier : Boucle while

- while (condition) { bloc }
- close

```
use Bio::SeqIO ;
$in = Bio::SeqIO->new(-file => "f1.fasta", -format => Fasta);
while ($seq = $in->next_seq()) {
  print $seq->seq();
}
$in->close();
```

Exercice : p3.pl

Objectif : utiliser une boucle while

Reprendre le programme p2.pl et faire les modifications suivantes :

- traiter toutes les séquences
- utiliser la fonction next_seq() du module Bio::SeqIO pour lire le fichier

Exercice : p4.pl

Objectif : Utiliser des variables

Reprendre le programme p3.pl et faire les modifications suivantes :

- pour chaque séquence
  - compter le nombre de séquences dans une variable count
  - cumuler la longueur totale en bp dans une variable size
- afficher count et size
- pour additionner 2 valeurs
  - utiliser l'opérateur +, par exemple
    - `$cout = $count + 1;`
    - `$size = $size + $seq->length() ;`
  - ou bien utiliser les opérateurs ++ et +=, ex
    - `$count += 1;`

      ou

      `$count++;`
    - `$size += $seq->length();`

Boucle for ou foreach

```
for $variable (@liste) { bloc }
```

```
for $file ("file1", "file2", "file3") {
  print $file, " \n";
}

@files = ("file1", "file2", "file3");
for $file (@files) {
  print $file, " \n";
}
```

Exercice : p5.pl

Objectif : Utiliser une boucle for

Reprendre le programme p4.pl et faire les modifications suivantes :

- lire 3 fichiers f1.fasta, f2.fasta et f3.fasta
- afficher
  - le nombre total de séquences
  - la longueur totale en bp
- utiliser une boucle for pour traiter les 3 fichiers

Nom du programme et paramètres de la ligne de commande

- On peut lancer un programme sans argument :
  - p5.pl
- Ou bien avec un ou plusieurs arguments :
  - p6.pl f1.fasta
  - p6.pl f1.fasta f3.fasta f55.fasta
  - p6.pl f*.fasta
- Lorsque le programme démarre, il trouve
  -
  - son nom dans la variable $0
  - ses arguments dans la liste @ARGV (vecteur d'arguments)
- Donc :

p6.pl	f1.fasta	f3.fasta	f55.fasta	etc...
$0	$ARGV[0]	$ARGV[1]	$ARGV[2]	etc...

Exercice : p6.pl

Objectif : utiliser les arguments de la ligne de commande

Reprendre le programme p5.pl et faire les modifications suivantes :

- lire tous les fichiers .fasta donnés en argument
- utiliser une boucle avec @ARGV pour traiter tous les fichiers de la ligne de commande
- exemples d'utilisation de ce programme
  - p6.pl f1.fasta
  - p6.pl f1.fasta f3.fasta
  - p6.pl f*.fasta

Exercice : p7.pl

Objectif : utiliser if et next

Reprendre le programme p6.pl et faire les modifications suivantes :

- ignorer les séquences $< 500bp$
- afficher le nombre de séquences totales et la longueur totale
- tester 2 solutions pour tester la condition :
  - la structure de contrôle if
  - la structure de contrôle next

Exercice : p8.pl

Objectif : combiner plusieurs conditions

Reprendre le programme p7.pl et faire les modifications suivantes :

- ignorer les séquences dont la longueur
  - est $< 500bp$
  - est $> 1500bp$
- tester 2 solutions pour tester les conditions :
  - la structure de contrôle if
  - la structure de contrôle next

Exercice : p9.pl

Objectif : utiliser une expression rationnelle

Reprendre le programme p9.pl et apporter les modifications suivantes :

- ignorer les séquences
  - dont la longueur $< 500bp$
  - dont l'identifiant commence par TEST_
- utiliser un expression rationnelle pour la seconde condition

Lire et écrire des fichiers fasta

- Module Bio::SeqIO
- Attention au nom du fichier en écriture
  - doit être précédé par >
- Méthode write_seq

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "f1.fa", -format => Fasta);
$out = Bio::SeqIO->new(-file => ">f2.fa", -format => Fasta);
while ($seq = $in->next_seq()) {
  $out->write_seq($seq);
}
```

Exercice : p10.pl

Objectif : écrire un fichier fasta

Reprendre le programme p9.pl et apporter les modifications suivantes :

- ignorer les séquences $< 200bp$
- écrire les séquences traduites dans le fichier select.fasta
- utiliser la méthode translate()

Exercice : p11.pl

Objectifs : utiliser une fonction, déclarer des variables locales, utiliser use strict

Reprendre le programme p10.pl et faire les modifications suivantes :

- déclarer une fonction main
- utiliser la directive use strict
- déclarez vos variables locales
- n'oubliez pas d'appeler la fonction main

Opérateurs de test sur les fichiers

- −f nom : nom est un fichier
- −d nom : nom est un répertoire
- −e nom : nom existe
- −z nom : nom est vide
- −s nom : nom est non vide

Exercice p12.pl

Objectif : utiliser les tests sur fichiers

Rependre le programme p11.pl et faire les modifications suivantes :

- tester si le fichier select.fasta existe
- si c'est le cas
  - afficher un message d'erreur
  - et terminer le programme
  - ainsi le fichier ne sera pas écrasé

Exercice p13.pl

Objectifs : créer un répertoire, ≪ fabriquer ≫ des noms de fichiers

Reprendre le programme p12.pl et faire les modifications suivantes :

- créer un fichier fasta par séquence, dans le dossier fasta
  - en utilisant l'identifiant comme nom de fichier
- utiliser mkdir pour créer le répertoire s'il n'existe pas
- utiliser une variable pour formater le nom du fichier
  - dans le nom du fichier remplacer tout caractère autre que minuscule, majuscule, chiffre ou _ par _ en utilisant une expression régulière

Exercice p14.pl

Objectif : utiliser un tableau associatif

Reprendre le programme p13.pl et faire les modifications suivantes :

- extraire la liste de séquences données dans un tableau associatif
- mettre 3 séquences dans le tableau associatif
  - exemple_du_cours_1, exemple_du_cours_2 et exemple_du_cours_3

Exercice p15.pl

Objectif : fichier → tableau associatif

Reprendre le programme p14.pl et faire les modifications suivantes :

- initialiser le tableau associatif
  - à partir du fichier list_id
- pour lire le fichier list_id , utiliser :
  - les fonctions open() et close ()
  - une boucle while
  - l'opérateur <>

Exercice p16.pl

Objectif : écrire dans un fichier

Reprendre le programme p15.pl et faire les modifications suivantes :

- écrire un résumé d'exécution dans le fichier trace
- chaque ligne contiendra :
  - le nom de la séquence à extraire
  - sa longueur, si la séquence a été trouvée
  - « Missing » sinon
- Pour écrire le fichier trace utiliser :
  - les fonctions open() et close()
  - une boucle while
  - la fonction print

Plan

1 Généralités sur Bioperl

2 Manipuler des fichiers de séquences

3 Collecter des séquences

4 Séquences et données associées

5 Blast

Partie 2

Collecter des séquences

- à partir de genbank
- avec NCBI Eutilities

Obtenir une séquence genbank

- Module Bio::DB::Genbank
- Database Object interface to genbank
- Méthode : get_Seq_by_acc
- Retourne une séquence

```
use Bio::DB::GenBank;
use Bio::SeqIO;
$gb = new Bio::DB::GenBank();
$seq = $gb->get_Seq_by_acc("NC_001284");
$out = Bio::SeqIO->new(-file => ">f1.gbk", -format => 'genbank
$out->write_seq($seq);
```

Exercice : 2a_gb_get_genome.pl

- Récupérer la séquence NC_001284
- Génome mitochondrial d'*Arabiposis thaliana*
- Le mettre dans un fichier au format genbank nommé At.gbk
- Ce fichier devrait contenir

```
LOCUS NC_001284 366924 bp DNA circular PLN 31-JUL-2008
DEFINITION Arabidopsis thaliana mitochondrion, complete genome
ACCESSION NC_001284
VERSION NC_001284.2 GI:26556996
KEYWORDS complete genome.
SOURCE mitochondrion Arabidopsis thaliana (thale cress)
ORGANISM Arabidopsis thaliana
Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheoph
Spermatophyta; Magnoliophyta; eudicotyledons; core eudicotyled
rosids; malvids; Brassicales; Brassicaceae; Camelineae;
Arabidopsis.
```

Obtenir un ensemble de séquences genbank

Modules

- Bio::DB::Genbank
- Bio::Query::Genbank

```
use Bio::DB::GenBank;
use Bio::DB::Query::GenBank;
$query = "Arabidopsis[ORGN] AND topoisomerase[TITL]
                AND 0:3000[SLEN]";
$query_obj = Bio::DB::Query::GenBank->new(-db => 'nucleotide',
  -query => $query );
print $query_obj->count,"\n";
$gb_obj = Bio::DB::GenBank->new;
$stream_obj = $gb_obj->get_Stream_by_query($query_obj);
while ($seq_obj = $stream_obj->next_seq) {
  # do something with the sequence object
  print $seq_obj->display_id, "\t", $seq_obj->length, "\n";
}
```

Exercice : 2b_gb_search_and_get_genomes.pl

- Récupérer les séquences de tous les génomes mitochondriaux de *Brassicaceae* en utilisant l'expression
  - Brassicaceae[Orgn] AND mitochondrion[TITL] AND "complete genome"
- Écrire ces séquences dans un fichier genbank nommé all_from_gb .gbk
- Vous devez obtenir 27 génomes séquence

```
grep LOCUS all_from_gb.gbk
LOCUS       KU831325                219975 bp    DNA     circula
```

NCBI Entrez – cross database search

- Système intégré d'accès à environ 40 base de données
  - Nucleotid, protein, gene, genome, homologene, GEO, SRA, epigenomics etc...)
- Recherche à partir de mots clés et d'opérateurs booléens (and, or, not)
- Extraction de données dans de nombreux formats (fasta, genbank), unique ou par lot
- Liens entre données des DB (séquence/abstract, séquence/protéine, protéine/structure 3D)
- Utilisable via le web `http://www.ncbi.nlm.nih.gov/Entrez`

NCBI Entrez Utilities (eutils) 1/2

- Objectif : pouvoir utiliser NCBI Entrez directement par des programmes
- « fonctions » ou « services »

Einfo (DB → info) : informations sur les bases (date de mise à jour, liste des champs, liens vers d'autres bases).
Ex : DB nucleotide

Esearch (texte → IDs) : recherche à partir de mots clés et d'opérateurs booléens (and, or, not), renvoie des listes d'ID.
Ex : Arabidopsis[ORGN] AND topoisomerase[TITL]

Esummary (IDs → record info) : obtenir des informations sur les enregistrements à partir d'une liste d'ID

Efetch (IDs → records) : télécharger des enregistrements à partir d'une liste d'ID

Elink (IDs DB1 → IDs DB2) : obtenir les liens vers une autre base à partir d'identifiants
Ex : liste d'id de la base protein → liste d'id correspondant dans la base nucleotid

NCBI Entrez Utilities (eutils) 2/2

- Ces services sont lancés par des URL
- Exemples :
  - Liste de toutes les bases `http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi`
  - Statistiques sur Entrez Protein `http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=protein`
  - PubMed ID pour les articles à propos de breast cancer publiés dans Science en 2015 `http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=science[journal]+AND+breast+cancer+AND+2015[pdat]`
- Problèmes :
  - URL complexes à générer
  - données retournées en XML

Bio::BD:: EUtilities

- Module Bioperl pour accéder aux services Eutilities
- Méthodes pour « fabriquer » des URL, lancer les requêtes et récupérer les résultats
- Permet de créer des programmes pour des tâches complexes impossible à faire en directe avec le site web
- Exemples
  - Esearch term + Efetch all records
  - Esearch in DB1 + elink on DB2 + esummary on DB2

Récupérer des identifiants avec Bio::DB:: EUtilities

- webagent which interacts with and retrieves data from NCBI's Entrez Utilities Web Services
- Cf Entrez Programming Utilities Help
  http://www.ncbi.nlm.nih.gov/books/NBK25501/

```
use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil => 'esearch',
 -db => 'protein', -term => 'BRCA1 AND human',
 -email => 'mymail@foo.bar', -retmax => 5000);
# query terms are mapped; what's the actual query?
print "Query translation: ",$factory->get_query_translation,"\n";
print "Count = ",$factory->get_count,"\n";# query hits
my @ids = $factory->get_ids ; # UIDs
```

```
Query translation: BRCA1[All Fields] AND ("Homo sapiens"[Organism] O
human[All Fields])
Count = 2363
```

Récupérer des identifiants avec Bio::DB:: EUtilities

```
use Bio::DB::EUtilities;
use Bio::SeqIO;
my @ids = qw(28096 28094);# GI
my $factory = Bio::DB::EUtilities->new(-eutil => 'efetch',
 -db => 'nucleotide',
 -id => \@ids,
 -email => 'thierry.lecroq@univ-rouen.fr',
 -rettype => 'gb');
my $file = 'myseqs.gb';
$factory->get_Response(-file => $file);
```

Exercice : 2b_eu_search_and_get_genomes.pl

- Récupérer les séquences de tous les génomes mitochondriaux de *Brassicaceae* en utilisant l'expression
  - Brassicaceae[Orgn] AND mitochondrion[TITL] AND "complete[genome" [TITL]
- Écrire ces séquences dans un fichier genbank nommé all_from_eu .gbk
- Vous devez obtenir les mêmes 27 génomes
- Utilisez d'abord esearch pour obtenir les ID, puis efetch pour obtenir les séquences

Exercice : 3a_read_genbank.pl

- Lire le fichier genbank all_from_eu .gbk
- Afficher le nom des séquences et leur longueur

Exemple de sortie

```
KP030753:232407
KP161618:221867
KM454975:223412
KM454974:223412
KF442616:247696
KJ716484:244054
KJ820683:219962
KJ461445:219919
KC193578:239186
JQ083668:239723
NC_018551:258426
NC_016123:219766
NC_016125:219747
```

Exercice : 3a_read_genbank.pl

Le type des séquences lues dépend du fichier d'entrée :

- fichier genbank $\rightarrow$ seqBio::Seq::RichSeq
- fichier fasta $\rightarrow$ seqBio::Seq

Exercice : 3b_genbank2fasta.pl

- Lire le fichier genbank all_from_eu .gbk
- Écrire les séquences au format fasta dans le fichier all_from_eu . fasta

Exemple de sortie

```
>KP030753 Brassica nigra mitochondrion, complete genome.
ATTCCTAGGTTTTAGAAGAAGGATGGAGCAGAAAGATAGAAAGATCAGTTGCTGGAAATC
AAATATGTAAATAGTCGAGTTATTGCATTCCTCTTCACAAACTATCAATTTCATAAGAGA
AGAAAGATCGTTTTTCGAAACTATCAATTTCATAAGAGAAGAAAGATCGTTTTTAGAAAA
GAAAGAAGGTTTTGATTCGAGGCGGATCCCTTTTTTCTTTGCCTTTACGAGTAAGAAAGT
...
```

Plan

1 Généralités sur Bioperl

2 Manipuler des fichiers de séquences

3 Collecter des séquences

4 Séquences et données associées

5 Blast

Séquences et données associées

- *Features*
- Annotations
- Références bibliographiques
- Species
- Filtrer les *features*
- Exporter les *features* au format GFF
- Visualiser les *features*

Séquences et données associées

Genbank contient beaucoup de données associées aux séquences

- Annotations : données associées à la séquence
  - Date, version, commentaires, références biblio
- *Features* :
  - Données associées à une partie de la séquence, déterminée par une localisation (CDS, exon, séquence répétée, ...)
  - Possèdent des ≪ sub-features ≫ ou tag (Ex nom du gene, protein_id , EC_number, db_xref (lien vers une autre DB ex UniProtKB/Swiss-Prot, GOA, GI))

```
LOCUS       NC_001284          366924 bp    DNA     circular PLN 31-JUL-2008
DEFINITION  Arabidopsis thaliana mitochondrion, complete genome.
ACCESSION   NC_001284
VERSION     NC_001284.2  GI:26556996
KEYWORDS    complete genome.
SOURCE      mitochondrion Arabidopsis thaliana (thale cress)
  ORGANISM  Arabidopsis thaliana
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta; ...
REFERENCE   1  (bases 1 to 366924)
  AUTHORS   Giege,P. and Brennicke,A.
  TITLE     RNA editing in Arabidopsis mitochondria effects 441 C to U changes in ORFs
  JOURNAL   Proc. Natl. Acad. Sci. U.S.A. 96 (26), 15324-15329 (1999)
   PUBMED   10611383
COMMENT     PROVISIONAL REFSEQ: This record has not yet been subject to final  NCBI review.
            The reference sequence was derived from Y08501.  COMPLETENESS: full length
```

annotations

fichier genbank

```
FEATURES             Location/Qualifiers
     source          1..366924
                     /mol_type="genomic DNA"
                     /db_xref="taxon:3702"
                     /organelle="mitochondrion"
                     /ecotype="Columbia"
                     /organism="Arabidopsis thaliana"
     gene            complement(join(327890..333105,79740..81297))
                     /locus_tag="ArthMp026"
                     /db_xref="GeneID:3371312"
                     /gene="nad2"
     CDS             complement(join(327890..328078,329735..330306,
                     332945..333105,79740..80132,81113..81297))
                     /locus_tag="ArthMp026"
                     /protein_id="NP_085584.1"
                     /gene="nad2"
                     /db_xref="GI:13449314"
                     /db_xref="GOA:O05000"
                     /db_xref="UniProtKB/Swiss-Prot:O05000"
                     /translation="MKAEFVRILPHMFNLFLAVSPEIFIINATSILLIHGVVFSTSKK ....
                     /product="NADH dehydrogenase subunit 2"
```

primary_tags ou features

tag

source

gene

localisation

CDS

features

Objets Bio : :Specie

Méthodes : node_name, classification

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "all_from_eu.gbk",
                                        -format => 'genbank');
while ($seq = $in->next_seq()) {
  $species = $seq->species->node_name;
  # get all taxa from the ORGANISM section in an array
  @classification = reverse($seq->species->classification);
  print $seq->id, " : $species [@classification]\n";
}
```

```
KP030753 : Brassica nigra [Eukaryota Viridiplantae Streptophyt
KP161618 : Brassica napus [Eukaryota Viridiplantae Streptophyt
KM454975 : Brassica napus [Eukaryota Viridiplantae Streptophyt
KM454974 : Brassica napus [Eukaryota Viridiplantae Streptophyt
KF442616 : Eruca vesicaria subsp. sativa [Eukaryota Viridiplan
KJ716484 : Raphanus sativus [Eukaryota Viridiplantae Streptoph
```

Annotations

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "f1.gbk", -format => 'genbank')
while ($seq = $in->next_seq()) {
  $anno_collection = $seq->annotation;
  for $key ($anno_collection->get_all_annotation_keys) {
    @annotations = $anno_collection->get_Annotations($key);
    for $value (@annotations) {
      print $value->tagname, ":";
      print "\t", $value->display_text, "\n";
    }
  }
}
```

```
keyword: complete genome
comment: PROVISIONAL REFSEQ: This record has not yet been subj
reference: RNA editing in Arabidopsis mitochondria effects 441
reference: The mitochondrial genome of Arabidopsis thaliana co
```

Annotations filtrées

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "f1.gbk", -format => 'genbank')
while ($seq = $in->next_seq()) {
  $anno_collection = $seq->annotation;
  @annotations = $anno_collection->get_Annotations('reference
  for $value (@annotations) {
    print $value->tagname, ":";
    print "\t", $value->display_text, "\n";
  }
}
```

```
reference: RNA editing in Arabidopsis mitochondria effects 441
reference: The mitochondrial genome of Arabidopsis thaliana co
reference: Genomic recombination of the mitochondrial atp6 ger
...
```

Features

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "f1.gbk",
-format => 'genbank');
while ($seq = $in->next_seq()) {
  for $feature ($seq->get_SeqFeatures) {
    print "feature : ", $feature->primary_tag, "\n";
    for $tag ($feature->get_all_tags) {
      print " tag: ", $tag, "\n";
      for $value ($feature->get_tag_values($tag)) {
        print " value: ", $value, "\n";
      }
    }
  }
}
```

Features

- gene utilisé pour primary key et tag
- tag sans valeur (trans_splicing)
- tags avec plusieurs valeurs (db_xref)

```
feature : source
 tag: db_xref
 value: taxon:3702
 tag: ecotype
 value: Columbia
 tag: mol_type
 value: genomic DNA
 tag: organelle
 value: mitochondrion
 tag: organism
 value: Arabidopsis thaliana
feature : gene
 tag: db_xref
```

Features filtrés

```
use Bio::SeqIO;
$in = Bio::SeqIO->new(-file => "f1.gbk", -format => 'genbank')
while ($seq = $in->next_seq()) { # Bio::Seq::RichSeq
  for $feature ($seq->get_SeqFeatures) { # feature=Bio::SeqFea
    if ($feature->has_tag('db_xref')) {
      @db_xrefs = $feature->get_tag_values('db_xref');
    }
    print $feature->primary_tag, ": @db_xrefs\n";
  }
}
```

```
source: taxon:3702
gene: GeneID:3371312
CDS: GI:13449314 GOA:O05000 UniProtKB/Swiss-Prot:O05000 GeneID
gene: GeneID:3371336
CDS: GI:13449291 GOA:P93275 UniProtKB/Swiss-Prot:P93275 GeneID
regulatory: GI:13449291 GOA:P93275 UniProtKB/Swiss-Prot:P93275
```

Exercice : 3 c_features . pl

- Lire le fichier genbank f1 .gbk
- Afficher le nombre de feature de chaque type et la liste des gènes

Exemple de sortie

```
117 CDS
1 STS
32 exon
131 gene
21 intron
455 misc_feature
3 rRNA
14 regulatory
6 repeat_region
1 source
21 tRNA
49 genes : atp1 atp6-1 atp6-2 atp9 ccb203 ccb206 ccb256 ccb382
```

Écriture de *features* en GFF

```
use Bio::SeqIO ;
use Bio::Tools::GFF;
$in = Bio::SeqIO->new(-file => "f1.gbk", -format => 'genbank');
$out = new Bio::Tools::GFF(-file => "> f1_gene.gff",
  -gff_version => 3);
while ($seq = $in->next_seq()) {
  for $feature ($seq->get_SeqFeatures) {
    if ($feature->primary_tag eq "CDS") {
      if ($feature->has_tag('gene')) {
        $out->write_feature($feature);
      }
    }
  }
}
```

```
##gff-version 3
NC_001284 EMBL/GenBank/SwissProt CDS 327890 328078 . -.
eC_number=1.6.99.3;codon_start=1;db_xref=GI:13449314,GOA:O050
```

Exercice : 3 d_features_gff .pl

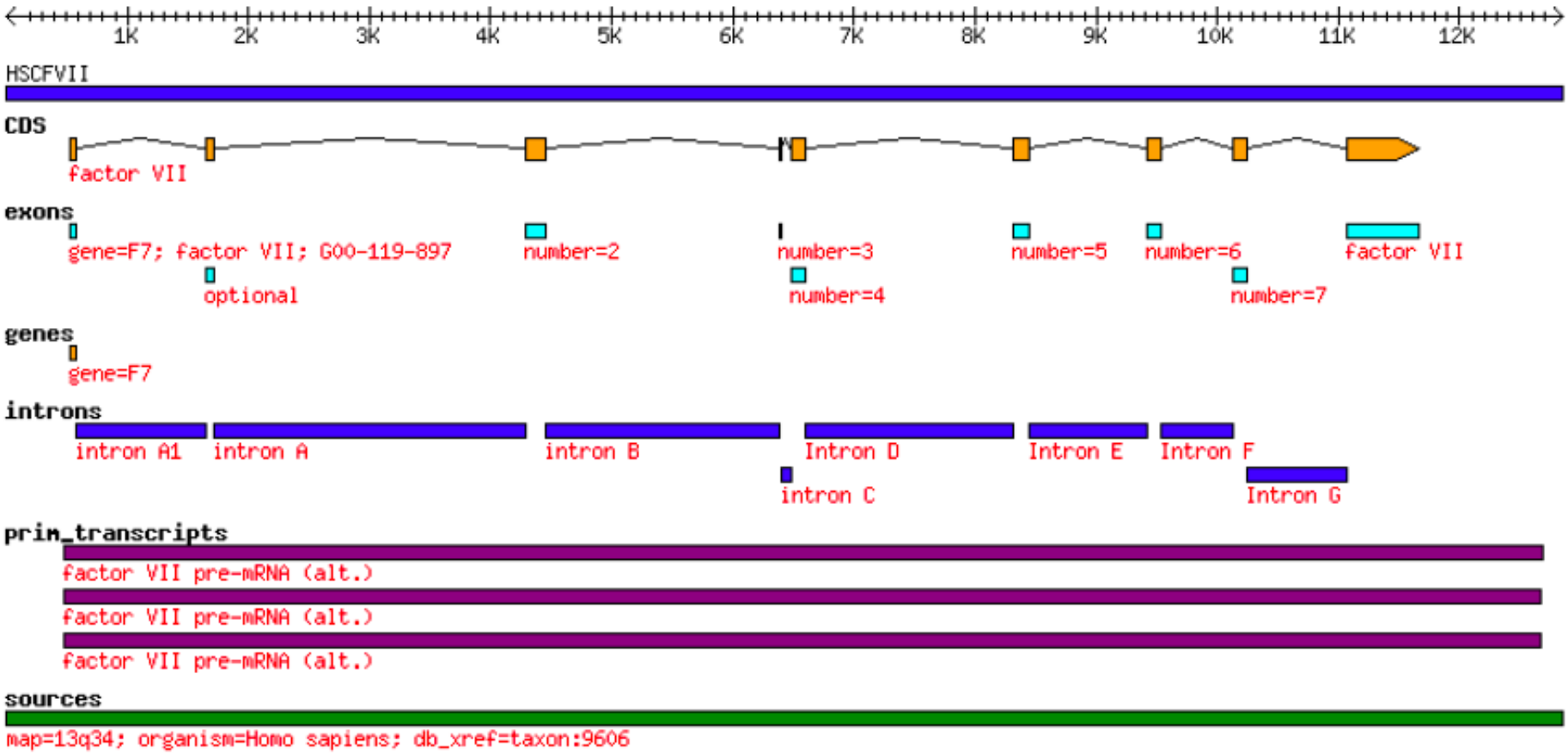
- Lire le fichier genbank f1.gbk
- Écrire les *features* CDS et gene au format GFF

Exemple de sortie

```
##gff-version 3
NC_001284 EMBL/GenBank/SwissProt CDS 327890 328078 . -.
eC_number=1.6.99.3;codon_start=1;db_xref=GI:13449314,GOA:O050
```

Visualisation des *features*

Use Bio::Graphics ;



Visualisation des *features* 1/2

```
# Rassembler les features de type CDS dans un tableau
use Bio::SeqIo ;
$in = Bio::SeqIO->new(-file => "f1.gbk", -format => 'genbank');
$seq = $in->next_seq();
for $feature ($seq->get_SeqFeatures) {
  next unless ($feature->primary_tag eq "CDS");
  push @CDS, $feature;
}
$in->close();

# Créer 1 feature pour la séquence entière
use Bio::SeqFeature::Generic;
$wholeseq = Bio::SeqFeature::Generic->new(
 -start => 1,
 -end => $seq->length,
 -display_name => $seq->display_name
);

# Créer un cadre graphique
use Bio::Graphics;
$panel = Bio::Graphics::Panel->new(
 -length => $seq->length,
 -key_style => 'between',
 -width => 800,
);
```

Visualisation des *features* 1/2

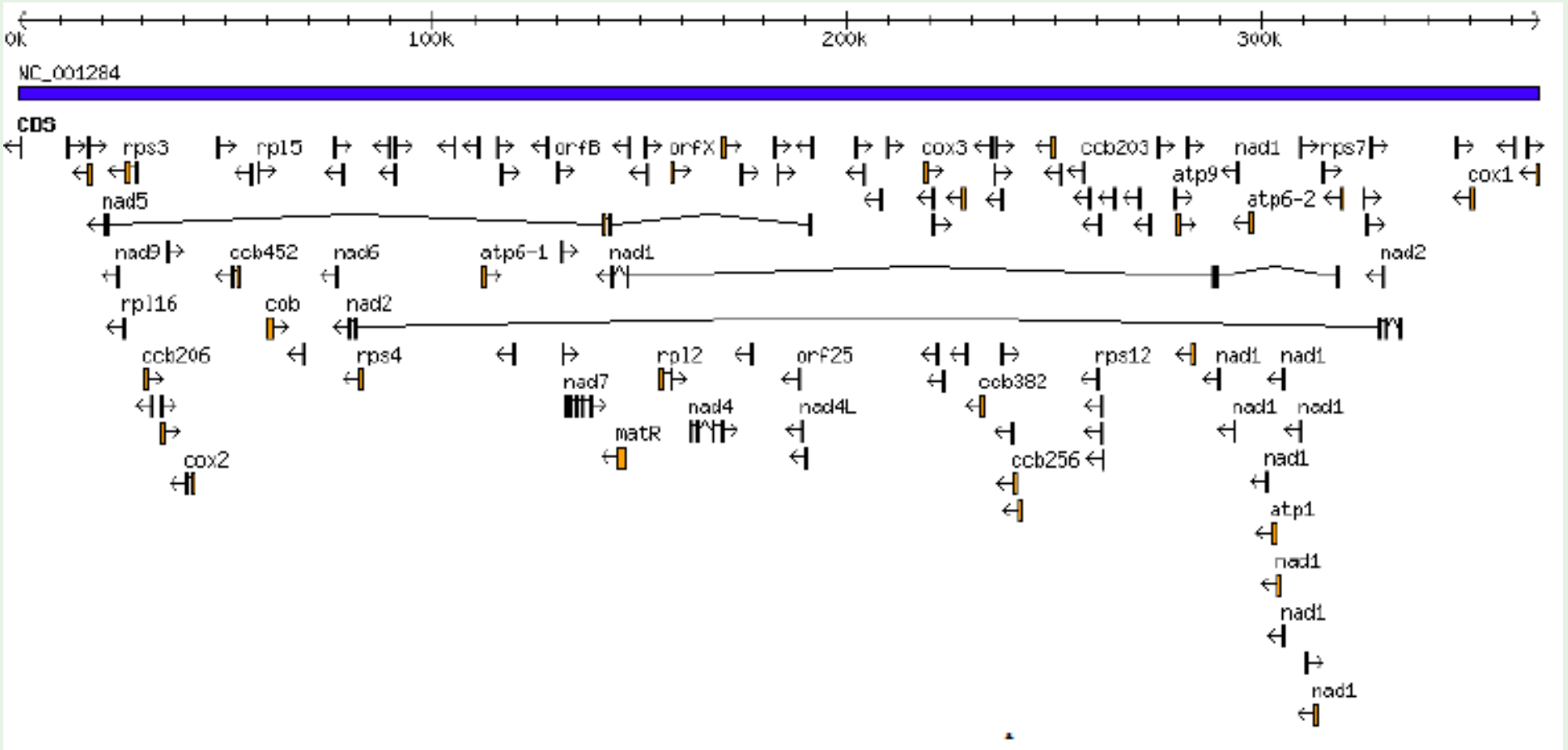
```
# Création des pistes
$panel->add_track($wholeseq,
 -glyph => 'arrow',
 -bump => 0,
 -double => 1,
 -tick => 2);
$panel->add_track($wholeseq,
 -glyph => 'generic',
 -bgcolor => 'blue',
 -label => 1);
$panel->add_track(\@CDS,
 -glyph => 'transcript2',
 -bgcolor => 'orange',
 -fgcolor => 'black',
 -font2color => 'red',
 -key => 'CDS',
 -bump => +1,
 -height => 12);

# Enregistrer le graphique en format png
open(FHO, ">f1.png") or die("f1.png");
print FHO $panel->png;
close(FHO);
```

Exercice : 3 e_features_graphic .pl

- Lire le fichier genbank f1.gbk
- Visualiser les *features* CDS dans un fichier f1.png

Exemple de sortie



Plan

1. Généralités sur Bioperl
2. Manipuler des fichiers de séquences
3. Collecter des séquences
4. Séquences et données associées
5. Blast

Blast

- Parcourir les résultats
- Filtrer les résultats sur différents critères
- Extraire les résultats dans un fichier excel
- Visualiser les résultats dans un graphique

Parcourir les résultats d'un blast

```
use Bio::SearchIO;
$in = new Bio::SearchIO(-format => 'blast',
  -file => "out_blast.txt");
while ($result = $in->next_result) {
  ## $result is a Bio::Search::Result::ResultI compliant object
  while ($hit = $result->next_hit) {
    ## $hit is a Bio::Search::Hit::HitI compliant object
    while ($hsp = $hit->next_hsp) {
      ## $hsp is a Bio::Search::HSP::HSPI compliant object
      print $result->query_name . "\t" . $hsp->start('query')
        . "\t" . $hsp->end('query')  . "\t" . $hit->name
        . "\t" . $hsp->start('subject') . "\t" . $hsp->end('subject
        . "\t" . $hsp->score . "\t" . $hsp->evalue
        . "\t" . $hsp->length('hit') . "/" . $hsp->length('total')
        . "\t" . sprintf("%3d", $hsp->percent_identity)
        . "\n";
    }
  }
}
```

Filtrer les résultats d'un blast

```
use Bio::SearchIO;
$in = new Bio::SearchIO(-format => 'blast', -file => "out_blas
while ($result = $in->next_result) {
  while ($hit = $result->next_hit) {
    while ($hsp = $hit->next_hsp) {
      next if ($result->query_name eq " XYZ " );
      next if ($hit->name =~ /LIB_A/) ;
      next if ($hsp->percent_identity <= 90);
      print $result->query_name . "\t" . $hit->name . "\t" . $
    }
  }
}
```

Exercice : 4 _parse_blast_txt .pl

- Filtrer les résultats de blast
  - hsp $> 100bp$
  - pourcentage d'identité > 80
  - Evalue < 0.05
- Lire dans le fichier blast1_out.txt
- Écrire les résultats filtrés dans un fichier texte blast1_ok.txt
- Écrire les colonnes suivantes :
  - q_name, q_start, q_end,
  - h_name, h_desc, h_start, h_end,
  - score, e−value, length, id, id_pct

Exemple de sortie

```
91 hits selected / 107 hits from blast1_out.txt to blast1_ok.t
query      q_start q_end  h_name h_desc   h_start h_end score e
NC_001284 218083  219169 1_0    contig_1 12034   13122 703   0
NC_001284 23608   24276  1_0    contig_1 18986   19653 472   0
NC_001284 349840  350321 1_0    contig_1 1       482   374   0
```

Écrire les résultats d'un blast dans un fichier excel

```
use Bio::SearchIO;
use Spreadsheet::WriteExcel;
my $fileName = "res_blast.xls";
my $in = new Bio::SearchIO(-format => 'blast', -file => "out_blast.txt");
my $xlsdoc = Spreadsheet::WriteExcel->new($fileName);
my $format_95 = $xlsdoc->add_format();
$format_95->set_bg_color(53);
my $sheetName = "resultats blast";
my $sheet = $xlsdoc->add_worksheet($sheetName); my $row = 0 ; my $col = 0 ;
$sheet->write($row, $col, [" query name", " hit name score"]); $row++;
while ($result = $in->next_result) {
  while ($hit = $result->next_hit) {
    while ($hsp = $hit->next_hsp) {
      next if ($hsp->percent_identity <= 90);
      $data = [$result->query_name, $hit->name, $hsp->score] ;
      if ($hsp->score > 95) {
        $sheet->write($row, $col, $data, $format_95); $row++;
      }
      else {
        $sheet->write($row, $col, $data); $row++;
      }
    }
  }
}
```

Exercice : 4_parse_blast_xls.pl

- Modifier le programme 4_parse_blast_txt.pl
- Pour écrire les résultats dans un fichier excel
- Colorer en fonction du pourcentage d'identité

q_name	q_start	q_end	h_name	h_desc	h_start	h_end	score	e-value	length	id	id_pct
NC_001284	218083	219169	1_0	contig_1	12034	13122	703	0	1089	1099	90
NC_001284	23608	24276	1_0	contig_1	18986	19653	472	0	668	675	92
NC_001284	349840	350321	1_0	contig_1	1	482	374	0	482	482	94
NC_001284	129909	130270	1_0	contig_1	14095	14456	254	1E-142	362	362	92
NC_001284	320747	321147	1_0	contig_1	20403	20828	241	1E-135	426	430	88
NC_001284	320260	320496	1_0	contig_1	20150	20402	155	2E-083	253	253	89
NC_001284	250043	250174	1_0	contig_1	18440	18567	84	5E-041	128	133	92
NC_001284	231921	233153	2_0	contig_2	16612	17847	815	0	1236	1242	91
NC_001284	76603	77367	2_0	contig_2	728	1500	581	0	773	774	93
NC_001284	21492	22181	2_0	contig_2	11407	12109	573	0	703	703	95
NC_001284	20494	21378	2_0	contig_2	12227	13132	555	0	906	910	91
NC_001284	240001	240732	2_0	contig_2	7102	7837	538	0	736	736	93
NC_001284	256662	257442	2_0	contig_2	15774	16562	502	0	789	794	91
NC_001284	81985	82189	2_0	contig_2	160	364	165	2E-089	205	205	95
NC_001284	240938	241124	2_0	contig_2	954	1140	147	1E-078	187	187	94
NC_001284	140656	142097	3_0	contig_3	15825	17266	1212	0	1442	1442	95
NC_001284	78906	80238	3_0	contig_3	1433	2777	805	0	1345	1353	89

Visualiser les résultats de blast

```
# Lire une séquence
use Bio::SearchIO;
$in = new Bio::SearchIO(-format => 'blast', -file => $fileIn);
$result = $in->next_result() or die "no result";
$in->close();

# Créer un objet graphique
use Bio::Graphics;
$panel = Bio::Graphics::Panel->new(-length => $result->query_length);

# Créer une feature pour toute la séquence
use Bio::SeqFeature::Generic;
$full_length = Bio::SeqFeature::Generic->new(
 -start => 1,
 -end => $result->query_length,
 -display_name => $result->query_name);

# Ajouter la feature « séquence complète » à l'objet graphique
$panel->add_track($full_length,
 -glyph => 'arrow',
 -tick => 2,
 -fgcolor => 'black',
 -double => 1,
 -label => 1) ;
```

Visualiser les résultats de blast

```
# Ajouter à l'objet graphique une piste pour les hits (cette
#  piste est vide au départ)
$track = $panel->add_track(
 -glyph => 'graded_segments',
 -label => 1,
 -connector => 'dashed',
 -bgcolor => 'blue',
 -font2color => 'red',
 -sort_order => 'high_score',
 -description => sub {
   my $feature = shift;
   "score=" . $feature->score;
  },
) ;
```

Visualiser les résultats de blast

```
# for each Hit
while ($hit = $result->next_hit) {
  # Créer une feature pour le hit
  $feature = Bio::SeqFeature::Generic->new(
   -score => $hit->raw_score,
   -display_name => $hit->name);
  # Ajouter les HSP à la feature
  while ($hsp = $hit->next_hsp) {
    $feature->add_sub_SeqFeature($hsp,'EXPAND');
  }
  # Ajouter la feature à la piste des hits
  $track->add_feature($feature);
}

# Enregistrer le graphique en format png
open(FHO, ">$fileOut") or die("cannot create $fileOut");
print FHO $panel->png;
close(FHO);
```