Alinhamentos em BioPython

Representação de alinhamentos

- O objeto **MultipleSeqAlignment** contém objetos e funções que permitem lidar com alinhamentos (com duas ou mais sequências);
- Estes objetos são usados para guardar a estrutura dos alinhamentos e não métodos para a sua criação

Objeto Align: exemplo

```
seq1 = MHQAIFIYQIGYPLKSGYIQSIRSPEYDNW
       ||  ||||||||*||||||||||||| ||
seq2 = MH--IFIYQIGYALKSGYIQSIRSPEY-NW
```

```
from Bio import Alphabet
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment
from Bio.Alphabet import IUPAC
from Bio.Seq import Seq

seqr1 = SeqRecord(Seq(seq1,Alphabet.Gapped(IUPAC.protein)),id="seq1")
seqr2 = SeqRecord(Seq(seq2,Alphabet.Gapped(IUPAC.protein)),id="seq2")
alin = MultipleSeqAlignment([seqr1, seqr2])
print (alin.get_alignment_length())
print (alin[:,2])
print (alin)
```

Leitura de alinhamentos: objeto AlignIO

- O objeto AlignIO funciona para os alinhamentos de forma semelhante ao SeqIO para sequências
- Permite ler e escrever alinhamentos em diversos formatos
- Funções para leitura:
  - **Bio.AlignIO.read()**: lê um único alinhamento; retorna um objeto MultipleSeqAlignment
  - **Bio.AlignIO.parse()**: lê um conjunto de alinhamentos, retornando um iterador
  - Em ambos os casos, os parâmetros obrigatórios são um nome de ficheiro (ou handle), ums string a especificar o formato

AlignIO: exemplo com read

Formato Stockholm, usado pelo PFAM

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print (alignment)
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTT...SKA COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTT...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTS...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVV...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVV...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVV...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALV...SRA COATB_BPIF1/22-73
```

AlignIO: exemplo cont.

Formato Stockholm, usado pelo PFAM

```
>>>  from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
 >>> print ("Tam. alinhamento %i" % alignment.get_alignment_length() )
Tam. alinhamento 52
>>> for record in alignment:
 ... print ("%s - %s" % (record.seq, record.id) )
>>> for record in alignment:
 ... if record.dbxrefs:
...         print (record.id, record.dbxrefs )
```

Exemplos de uso do iterador sobre os objetos SeqRecord no alinhamento

AlignIO: exemplo com read

Formato FASTA

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.faa", "fasta")
>>> print (alignment )
>>> print ("tam. alinhamento %i" % alignment.get_alignment_length() )
```

Note que apenas varia o formato … neste caso FASTA

AlignIO: exemplo com parse

Formato Phylip

```
from Bio import AlignIO
alignments = AlignIO.parse("resampled.phy", "phylip")
for alignment in alignments:
    print (alignment)

alignments = list(AlignIO.parse("resampled.phy", "phylip"))
last_align = alignments[-1]
first_align = alignments[0]
```

Note que, neste caso, são lidos vários alinhamentos sendo retornado um iterador sobre objetos MultipleSeqAlignment

AlignIO: escrita de alinhamentos

- A função **Bio.AlignIO.write()** permite escrever alinhamentos em vários formatos
- Argumentos:
  - Lista com objetos MultipleSeqAlignment
  - Nome do ficheiro (ou handle)
  - Formato (string)

AlignIO: exemplo write

```
from Bio.Alphabet import generic_dna
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment

align1 = MultipleSeqAlignment([SeqRecord(Seq("ACTGCTAGC", generic_dna), id="A"),
    SeqRecord(Seq("ACT-CTAGC", generic_dna), id="B"),
    SeqRecord(Seq("ACTGCTAGD", generic_dna), id="C"), ])
align2 = MultipleSeqAlignment([ SeqRecord(Seq("TCAGC-AG", generic_dna), id="D"),
    SeqRecord(Seq("ACAGCTAG", generic_dna), id="E"),
    SeqRecord(Seq("TCAGCTAG", generic_dna), id="F"), ])

my_alignments = [align1, align2]

from Bio import AlignIO
AlignIO.write(my_alignments, "my_example.phy", "phylip")
AlignIO.write(my_alignments, "my_example.sth", "stockholm")
AlignIO.write(my_alignments, "my_example.faa", "fasta")
```

AlignIO: conversão de formatos

```
from Bio import AlignIO
count = AlignIO.convert("PF05371_seed.sth", "stockholm",
        "PF05371_seed.aln", "clustal")
print ("Convertidos %i alinhamentos" % count )
```

Equivalente a

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
AlignIO.write([alignment], "PF05371_seed.aln", "clustal")
```

Slicing de alinhamentos: exemplos

`alignment[3:7]` Dará o alinhamento com as sequências entre a 4ª e a 8ª (todas as colunas)

`alignment[2].seq[6]` Uma única letra: 3ª sequência, 7ª linha do alinhamento

`alignment[:, 6]` Apenas uma coluna (a 7ª); todas as sequências

`alignment[3:6, :6]` Sequências da 4ª à 7ª, primeiras 6 colunas

Módulo Bio.pairwise2

- O BioPython tem uma implementação própria dos métodos de programação dinâmica no módulo Bio.pairwise2
- Esta implementação não está integrada com as interfaces do AlignIO e MultipleSeqAlignment
- Exemplos e help em: http://biopython.org/DIST/docs/api/Bio.pairwise2-module.html

Exemplo – module pairwise2

xx – score conta nº de caracteres iguais (g = 0)

```
>>> from Bio import pairwise2
>>> alignments = pairwise2.align.globalxx("ACCGT", "ACG")
>>> alignments
[('ACCGT', 'AC-G-', 3.0, 0, 5), ('ACCGT', 'A-CG-', 3.0, 0, 5)]
>>> alignments[0]
('ACCGT', 'AC-G-', 3.0, 0, 5)
>>> from Bio.pairwise2 import format_alignment
>>> for a in alignments: print(format_alignment(*a))
...
ACCGT
|||||
AC-G-
  Score=3
...
```

*a equivalente a a[0], a[1], a[2], a[3], a[4]

Exemplo – module pairwise2

```
>>> from Bio.SubsMat import MatrixInfo
>>> matrix = MatrixInfo.blosum62
>>> for a in pairwise2.align.globalds("KEVLA", "EVL", matrix,-4,-1):
...     print(format_alignment(*a))
KEVLA
|||||
-EVL-
  Score=5
>>> for a in pairwise2.align.localds("KEVLAKK", "EVL", matrix,-4,-1):
...     print(format_alignment(*a))
KEVLAKK
 |||
-EVL---
  Score=13
```

ds –matriz de substituição; penalização constante por gap

Análise Filogenética em BioPython

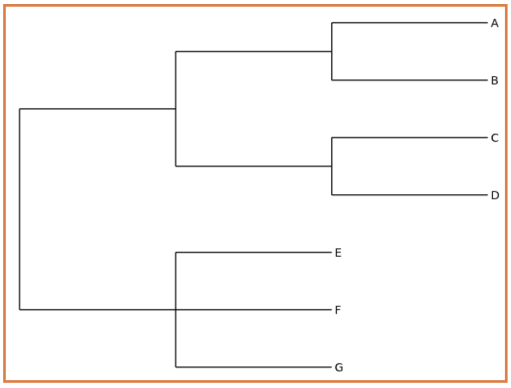
Representação de árvores

Formato Newick

(((A,B),(C,D)),(E,F,G))

Ficheiro “simple.dnd”

Representação gráfica

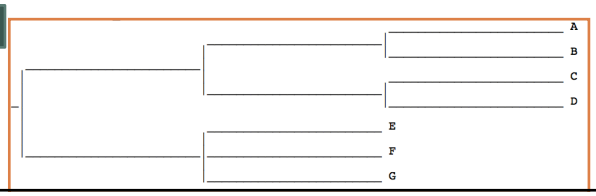


Criar árvore e desenhar

```
>>> from Bio import Phylo
>>> tree = Phylo.read("simple.dnd", "newick")
>>> print (tree)
```

```
Tree(weight=1.0, rooted=False, name="")
    Clade(branch_length=1.0)
        Clade(branch_length=1.0)
            Clade(branch_length=1.0)
                Clade(branch_length=1.0, name="A")
                Clade(branch_length=1.0, name="B")
            Clade(branch_length=1.0)
                Clade(branch_length=1.0, name="C")
                Clade(branch_length=1.0, name="D")
        Clade(branch_length=1.0)
            Clade(branch_length=1.0, name="E")
            Clade(branch_length=1.0, name="F")
            Clade(branch_length=1.0, name="G")
```

```
>>> Phylo.draw_ascii(tree)
```



Objetos Tree e Clade

- Os objetos Tree e Clade permitem representar a estrutura de uma árvore de forma recursiva
  - Tree – informação global da árvore (e.g. se tem raiz)
  - Clade – informação específica de cada nó/ramo, tal como comprimento + informação sobre as suas sub-árvores
- Existem diversos métodos para percorrer e modificar árvores e sub-árvores (ver secção 13.4 do tutorial)

Funções de I/O para árvores

- Tal como no caso do SeqIO e do AlignIO, o biopython disponibiliza com o Phylo um conjunto de funções para ler e escrever árvores em diversos formatos: parse, read, write e convert.
- Os significados de cada uma destas funções são muito semelhantes aos anteriores como se constata nos exemplos seguintes

Funções de I/O para árvores: exemplos

```
>>> from Bio import Phylo
>>> tree = Phylo.read("int_node_labels.nwk", "newick")
>>> print (tree)
>>> Phylo.draw_ascii(tree)

>>> Phylo.convert("int_node_labels.nwk", "newick", "tree.xml",
"phyloxml")

>>> trees = Phylo.parse("tree.xml", "phyloxml")
>>> for tree in trees: print(tree)
```