

Main Algorithms and Applications in Bioinformatics

Gregory Nuel

nuel@math-info.univ-paris5.fr

June 30 - July 12, 2008

1 Suffix Trees and Applications

1.1 The Mississippi example

Create `stree.pl` from Listing 1 using your favourite text editor (this Perl script uses [1, 2]). Then execute and redirect in `mississippi.stree` its results in a file (if the program get “stuck”, use CTRL-C to escape):

```
perl stree.pl > mississippi.stree
^C
```

Get the file `stree2dot.cc`¹ and compile it:

```
g++ stree2dot.cc -o stree2dot
```

Use it to convert the `.stree` file to a `.dot` file which can then be visualized through the `dot` program [3].

```
# convert .stree file to .dot file
./stree2dot mississippi.stree > mississippi.dot
# use dot to produce .eps file
dot -Tps -Grankdir=LR mississippi.dot -o mississippi.eps
# vizualize the eps file
gv mississippi.eps
```

Using the graphical representation of the suffix tree, try to:

- locate the occurrences of `ss` in `mississippi`
- find the longest repeated substring in `mississippi`

Create `stree2.pl` from Listing 2, and execute it. What is doing this Perl script ? Could you edit the script to localize the occurrences of `si` and `is` in `mississippi` ?

1.2 Banana time

Build a suffix tree for `bananamanamabamana` and produce its graphical representation. Use this suffix tree to:

- find the number of occurrences of `nam` and their positions;
- find the number of occurrences of `ana` and their positions;
- find the longest repeated substring and the positions where it appears.

¹http://www.math-info.univ-paris5.fr/~nuel/cours/motifs_M1/tp/stree2dot.cc

```
#!/usr/bin/perl
use Tree::Suffix;

$tree=Tree::Suffix->new("mississippi");
$tree->dump();
```

Listing 1: stree.pl

```
#!/usr/bin/perl
use Tree::Suffix;

$tree = Tree::Suffix->new("mississippi");

@loc=$tree->find("ss");
foreach (@loc) {
    $s=${$_}[1];
    $e=${$_}[2];
    print "\"ss\" starts in $s and ends in $e\n";
}

$count=$tree->find("ss");
print "total number of occurrences = $count\n";

$rep=$tree->lrs;
print "longest repeated substring = \"$rep\"\n";
```

Listing 2: stree2.pl

1.3 Now with a FASTA sequence

Get the file `U00093.fasta`² (it contains the complete chromosome VIII of Yeast in FASTA format). Create `fasta.pl` from Listing 3 (which uses [4]) and execute it. What does it do ?

Inspire from `stree2.pl` and `fasta.pl` to build a suffix tree from `U00093`, and use it to find:

- the frequencies of a, c, g, and t in the chromosome;
- the number of occurrences of `gattaca` and their starting positions;
- the number of occurrences of `tgtaatc` and their ending positions;
- the longest exact repetition, and the position of the repeated subsequences.

1.4 Application to genome comparison

Get the EMBL [5] webservice client `dbfetch.pl`³ and use it to obtain three complete genomes of HIV type 1 by retrieving the FASTA sequence corresponding to entries `K03455`, `AJ302646`, and `AJ302647`:

```
perl dbfetch.pl fetchData embl:K03455 fasta > hiv1.fasta
perl dbfetch.pl fetchData embl:AJ302646 fasta> hiv2.fasta
perl dbfetch.pl fetchData embl:AJ302647 fasta > hiv3.fasta
```

²http://www.math-info.univ-paris5.fr/~nuel/cours/motifs_M1/tp/U00093.fasta

³<http://www.ebi.ac.uk/Tools/webservices/downloads/perl/dbfetch.pl>

```
#!/usr/bin/perl
use Bio::Seq;
use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file=>"U00093.fasta",
                             -format=>"Fasta");
$seq_obj = $seqio_obj->next_seq;
$seq = $seq_obj->seq;
print "$seq \n";
```

Listing 3: fasta.pl

Build a suffix tree from the concatenation of these three sequences (using # as separator). Use this suffix tree to find the longest exact repeat between these genomes.

2 Automata and Pattern Matching

2.1 Toy examples

Build the NFA associated to the pattern (or motif) `a[ab]ab` using `spatt [6]` and `dot`:

```
spatt -a "ab" -p "a[ab]ab" --nfa nfa.dot
dot -Tps -Grankdir=LR nfa.dot -o nfa.eps
gv nfa.eps
```

How many states have this NFA ? How many final states ?

Using similar commands, build and visualize the DFA associated to the same motif thanks to the `spatt` option: `--dfa`. How many states and final states have this DFA ?

Complete the following table:

pattern	# of NFA states	# of DFA states
<code>a[ab](1)ab</code>		
<code>a[ab](2)ab</code>		
<code>a[ab](3)ab</code>		
<code>a[ab](4)ab</code>		
<code>a[ab](5)ab</code>		

Create a FASTA file `ab.fasta` which contains a single random sequence over $\mathcal{A} = \{a, b\}$ of length at least 100.

Create `regex.pl` from Listing 4 and execute it.

NB: note that Perl's regex syntax differs from SPatt's pattern syntax by the way to handle repeats. In Perl, `{1,3}` stands for 1 to 3 repeats of the previous character, while in SPatt, the same result is achieved with `(1-3)`.

Edit the script to return also the number and position of occurrences of `aaaba`, `ababa`, `aaaaba`, `aababa`, `abaaba`, and `abbaba`. Compare all these results. Are they consistent ?

2.2 Application: PROSITE signatures

Retrieve from the PROSITE database⁴ [7] the signature of the PDOC01004 entry. Transform this signature into Perl's regex format by applying the following rules:

⁴<http://www.expasy.ch/prosite/>

```
#!/usr/bin/perl
use Bio::Seq;
use Bio::SeqIO;

$regex="a[ab]{1,2}aba";
$file="ab.fasta";

$seqio_obj = Bio::SeqIO->new(-file => $file,
                             -format => "Fasta");

while ($seq_obj = $seqio_obj->next_seq) {
    $seq = $seq_obj->seq;
    $title=$seq_obj->display_name;
    $count=0;
    while ($seq=~/(?=$regex)/g) {
print "$title: $regex starts at position $-[0]\n";
$count++;
    }
    print "Total count of $regex in $title is $count\n";
}
}
```

Listing 4: regex.pl

- remove any white space;
- remove all -;
- replace all x by . (both mean “any letter” in their respective formats);
- use accolade instead of parenthesis for repeat syntax.

Use the resulting regex and previous scripts to count the occurrences and identify the matching proteins in `sample_uniprot_sprot.fasta`⁵

Do the same work for the signature of the PROSITE entry PS01242.

2.3 Counting all words of a given length

We consider here the problem of counting all possible DNA words of length $h = 5$ in the `U00093.fasta` file.

Create `bf.pl` from Listing 5. How many time it takes to this brute force approach to perform the computation (you may add the `time` command at the beginning of your command line) ?

Edit the previous script to first build a suffix tree for the sequence, and then use it to produce the result. Is this script faster or slower than the previous one ?

Also try `wordcount` [8] and `sspat` [9] on the same problem:

```
time wordcount -sequence U00093.fasta -wordsize 5 -outfile stdout
time sspatt U00093.fasta -a acgt -l 5 --all-words
```

What program is the fastest ?

⁵http://www.math-info.univ-paris5.fr/~nuel/cours/motifs_M1/tp/sample_uniprot_sprot.fasta

```

#!/usr/bin/perl
use Bio::Seq;
use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file => "U00093.fasta",
                             -format => "Fasta");
$seq_obj = $seqio_obj->next_seq;
$seq = $seq_obj->seq;

foreach $token5 ("a","c","g","t") {
  foreach $token4 ("a","c","g","t") {
    foreach $token3 ("a","c","g","t") {
      foreach $token2 ("a","c","g","t") {
        foreach $token1 ("a","c","g","t") {
          $word="$token5$token4$token3$token2$token1";
          $count=0;
          while ($seq=~/(?=$word)/g) {
            $count++;
          }
          print "$word $count\n";
        }
      }
    }
  }
}

```

Listing 5: bf.pl

Complete the following running time table:

method	h=3	h=5	h=7	h=9
regex				
suffix tree				
wordcount				
sspatt				

How do these running times grow with h ?

3 Dynamic Programming and Alignment

3.1 Global alignment

Retrieve the human, bovine, and mouse alpha-hemoglobin protein sequences from the uniprot database [10] using the following commands:

```

perl dbfetch.pl fetchData uniprot:Q9NZD4 fasta > human.fasta
perl dbfetch.pl fetchData uniprot:Q865F8 fasta > bovine.fasta
perl dbfetch.pl fetchData uniprot:Q9CY02 fasta > mouse.fasta

```

Use the needle [8] program to perform the three possible global alignments between these three proteins, using the default substitution matrix (here EBLOSUM62), and the default gap costs

(open=10.0, extension=0.5). Propose a phylogenetic tree for these three species which is consistent with your alignment results

Do again these alignments with gap costs reduced to 0.0 (open and extension). What do you observe ?

Copy the EBLOSUM62 file into your working directory under the name of MYBLOSUM62 (you may use the `locate` command to find the EBLOSUM62 file). Edit this substitution matrix by changing a positive term on the identity diagonal by -100, and use this altered substitution matrix to perform once again the alignments. What do you conclude ?

3.2 Local alignment

Use `water` [8] to perform a local alignment of `human.fasta` and `mouse.fasta` (using default parameters). On this particular example, what are the differences between local and global alignments ?

Retrieve a bacterial hemoglobin sequence from the command:

```
perl dbfetch.pl fetchData uniprot:P04252 fasta > bacterial.fasta
```

Compare the global and local alignment of `human.fasta` and `bacterial.fasta` (using default parameters). What do you conclude ?

References

- [1] libstree, a generic suffix tree library written in C, 2008
<http://www.cl.cam.ac.uk/~cpk25/libstree/>
- [2] Tree::Suffix, Perl binding to libstree, 2008
<http://search.cpan.org/dist/Tree-Suffix/>
- [3] Graphviz, open source graph (network) visualization project from AT&T Research, 2008
<http://www.graphviz.org/>
- [4] BioPerl, open source Perl tools for bioinformatics, genomics and life science research, 2008
<http://www.bioperl.org/>
- [5] EMBL, the European Molecular Biology Laboratory nucleotide database, 2008
<http://www.ebi.ac.uk/embl/>
- [6] SPatt version 2.x, Statistics for Patterns, the DFA approach, 2008
<http://stat.genopole.cnrs.fr/spatt>
- [7] PROSITE, database of protein domains, families and functional sites, 2008
<http://www.expasy.ch/prosite/>
- [8] EMBOSS, open source software for molecular biology, 2008
<http://emboss.sourceforge.net/>
- [9] SPatt version 1.x, Statistics for Patterns, the “old school” approach, 2008
<http://stat.genopole.cnrs.fr/spatt>
- [10] UniProt, the Universal Protein Resource, 2008
<http://www.uniprot.org/>