Genome 540

Introduction to Computational Molecular Biology:

Genome and protein sequence analysis
Today’s Lecture

• Course overview

• Administrative details

• Finding exact matches in sequences using suffix arrays
Computational Molecular Biology

Molecular biology

Probability and statistics

Computer science
Course Lecture Content

• DNA and protein sequences
  – Algorithms
    • Dynamic programming
  – Probability models
    • HMMs
    • Information theory
We do not cover:

• ‘Non-linear’ (non-sequence based) computational biology
  – protein structure, expression arrays, metabolic pathways, models for interacting molecules …

• ‘Machine learning’ applications

• Existing software tools
Course Prerequisites

• You *must*
  – be able to write programs for data analysis
  – have access to a computer where you can write & run your programs
    HW assignment # 1 will be a good test!

• Some previous familiarity with
  – probability and statistics
  – molecular biology

  is highly desirable
  – (if you lack it, you will have to work harder!)
Course Requirements

• Homework
• No tests or exams
• Attendance at discussion section strongly encouraged but not required
• Ask questions!
  – in lecture
  – at discussion section
  – by email
  – (via message board)
Homework

• Due weekly, Sunday at midnight
  – Posted on web site approx 1.5 weeks in advance
  – Each is 10 pts, late penalty of 1 pt/day (max penalty 3 pts)
  – Can redo

• write computer program to analyze genomic data set
  – “From scratch”, i.e. not using prewritten routines from elsewhere
  – Run on your own computer
  – Programming language is up to you – but a compiled language (e.g. C, C++) is recommended for efficiency reasons
    • Python + Cython also works
    • Interpreted language may work, but risky!
• Also: readings (in textbooks, or journal articles)

• turn in results of analysis, and your program, with (in some cases) a written interpretation of the results;
  – all to be submitted by email in computer-readable format
Course Info

• Instructors (contact info is on web page):
  – Phil Green
  – TA: Serena Liu

• Office hours by appointment (send Serena or me an email)

• if you did not receive the email I sent yesterday, send me (phg@uw.edu) your email address today (whether or not you are registered!)
• Lectures: TuTh 10:30-11:50, Foege S-110
• Weekly discussion section:
  – discuss homework, answer questions
  – review background material
  – related topics (next-gen sequencing?)

*Tentative* time/place: Th 12-1, Foege S-040
  – If you have a conflict, *email me* your schedule of unavailable times & we will try to find another

  – will post HW assignments, copies of slides here
  – has link to last year’s site – for approx syllabus & slides
Texts (will follow only loosely):


- *Statistical Methods in Bioinformatics: An Introduction (Statistics for Biology and Health)* by Ewens & Grant. Hardbound, ~$105. N.B. This is the 2D edition!

- available from UW Bookstore (South Campus Center branch) or from Amazon or Barnes & Noble
Finding perfectly matching subsequences of a sequence

- Idea (*much* more efficient than ‘brute force’ approach):
  - *suffix array* (Manber & Myers, 1990)
  - make list of pointers to all positions in sequence
  - lexicographically sort list of strings that are pointed to
  - process the list: adjacent entries are “maximally agreeing”
Suffix array step 1:
List of Pointers to Suffixes

```
<table>
<thead>
<tr>
<th>p</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₂</td>
<td>CCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₃</td>
<td>CTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₄</td>
<td>TGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₅</td>
<td>GCACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₆</td>
<td>CACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₇</td>
<td>ACTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₈</td>
<td>CTAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₉</td>
<td>TAAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₁₀</td>
<td>AAACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₁₁</td>
<td>AACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
<tr>
<td>p₁₂</td>
<td>ACCGTACACTGGGTTCAAGAGATTTCCC</td>
</tr>
</tbody>
</table>
```
Suffix array step 2: View as Strings to be Compared

ACCTGCACTAAACCCTACACTGGGTTCAGAGATTTCCC

\[ p_1 \] ACCTGCACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_2 \] CCTGCACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_3 \] CTGCACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_4 \] TGCACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_5 \] GCACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_6 \] CACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_7 \] ACTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_8 \] CTAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_9 \] TAAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_{10} \] AAACCCTACACTGGGTTCAGAGATTTCCC
\[ p_{11} \] ACCCTACACTGGGTTCAGAGATTTCCC
\[ p_{12} \] ACCCTACACTGGGTTCAGAGATTTCCC

\ldots
Suffix array step 3: Sort the Pointers Lexicographically

ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC

<table>
<thead>
<tr>
<th>p</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>AAACCGTACACGTGGGTTCAGAGATTCCC</td>
</tr>
<tr>
<td>11</td>
<td>AAACCGTACACGTGGGTTCAGAGATTCCC</td>
</tr>
<tr>
<td>12</td>
<td>AAGAGATTCCC</td>
</tr>
<tr>
<td>17</td>
<td>ACACTGGGTTCAGAGATTCCC</td>
</tr>
<tr>
<td>28</td>
<td>ACACTGGGTTCAGAGATTCCC</td>
</tr>
<tr>
<td>29</td>
<td>ACCTGCACTAAACCGTACACGTGGGTTCAGAGATTCCC</td>
</tr>
<tr>
<td>31</td>
<td>AGATTTCCC</td>
</tr>
<tr>
<td>33</td>
<td>ATTTCCC</td>
</tr>
<tr>
<td>27</td>
<td>CAAGAGATTCCC</td>
</tr>
</tbody>
</table>
Finding Matching Subsequences Using the Sorted List of Pointers

• Perfectly matching subsequences
  – (more precisely – the pointers to the starts of those subsequences)
  are “near” each other in the sorted list

• For a given subsequence, a longest perfect match to it is adjacent to it in the sorted list
  – (there may be other, equally long matches which are not adjacent, but they are nearby).
(Average Case) Complexity Analysis

- If $N =$ sequence length, sorting can be done with
  - $O(N\log(N))$ comparisons,
  - each requiring $O(\log(N))$ steps on average,
for an overall complexity of $O(N(\log(N))^2)$.
  - (Processing the sorted list requires an additional $O(N)$ steps which does not affect the overall complexity).

- Manber & Myers (1990) have more efficient algorithm ($O(N\log(N))$)
- several $O(N)$ algorithms are now known – but the best implementations are not as fast as $O(N\log(N))$ algorithms, even for very large genomes!!

- ∃ other, older $O(N)$ methods (‘suffix trees’), but these are
  - much less space efficient,
  - harder to program, and
  - (probably) slower in practice
• HW #1 (to be posted soon) asks you to apply this algorithm to find
  – longest perfectly matching subsequences in 2 genomic sequences & their reverse complements.
• much faster than an $O(N^2)$ algorithm (e.g. Smith-Waterman, or even BLAST), but
• limited to finding exact matches