

Genome 540

Introduction to Computational Molecular Biology:

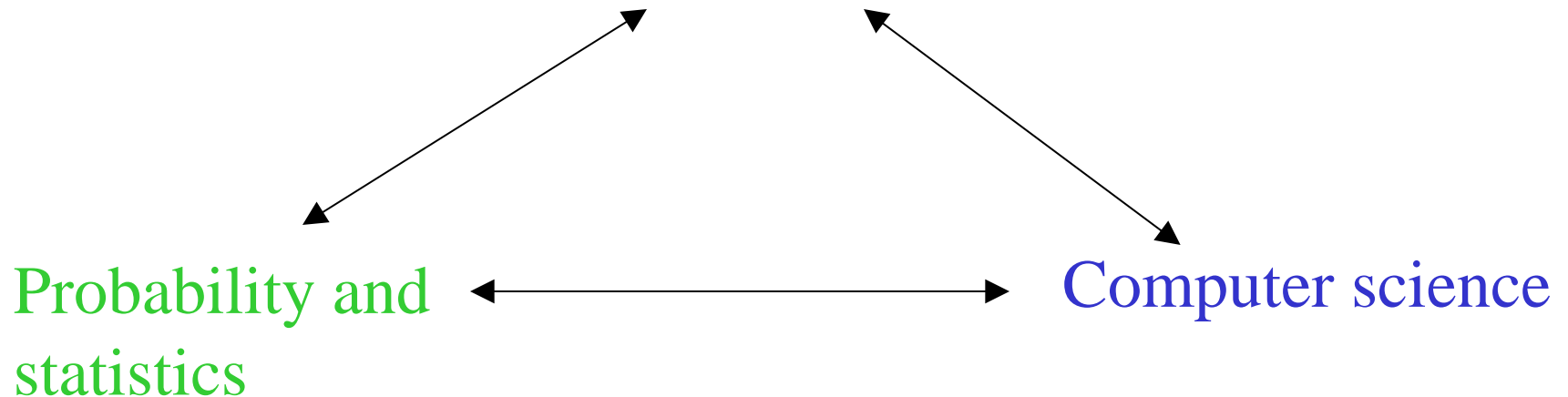
Genome and protein sequence analysis

Today's Lecture

- Course overview
- Administrative details
- Finding exact matches in sequences using suffix arrays

Computational Molecular Biology

Molecular biology



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 programsHW assignment # 1 will be a good test!
- Some previous familiarity with
 - probability and statistics
 - molecular biologyis 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: Eliah Overbey
- Office hours by appointment (send Eliah 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
- Web site: <http://www.phrap.org/compbio/mbt599>
 - will post HW assignments, copies of slides here
 - has link to last year's site – for approx syllabus & slides

Texts (*will follow only loosely*):

- *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* by Durbin, Eddy, Krogh & Mitchison. Paperback, ~\$60.
- *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

ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC

p_1	ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_2	CCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_3	CTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_4	TGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_5	GCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_6	CACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_7	ACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_8	CTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_9	TAAACCGTACACTGGGTTCAAGAGATTTCCC
p_{10}	AAACCGTACACTGGGTTCAAGAGATTTCCC
p_{11}	AACCGTACACTGGGTTCAAGAGATTTCCC
p_{12}	ACCGTACACTGGGTTCAAGAGATTTCCC

⋮

Suffix array step 2:

View as Strings to be Compared

ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC

p_1	ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_2	CCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_3	CTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_4	TGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_5	GCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_6	CACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_7	ACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_8	CTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_9	TAAACCGTACACTGGGTTCAAGAGATTTCCC
p_{10}	AAACCGTACACTGGGTTCAAGAGATTTCCC
p_{11}	AACCGTACACTGGGTTCAAGAGATTTCCC
p_{12}	ACCGTACACTGGGTTCAAGAGATTTCCC

⋮

Suffix array step 3:

Sort the Pointers Lexicographically

ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC

p_{10}	AAACCGTACACTGGGTTCAAGAGATTTCCC
p_{11}	AACCGTACACTGGGTTCAAGAGATTTCCC
p_{28}	AAGAGATTTCCC
p_{17}	ACACTGGGTTCAAGAGATTTCCC
p_{12}	ACCGTACACTGGGTTCAAGAGATTTCCC
p_1	ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_7	ACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_{19}	ACTGGGTTCAAGAGATTTCCC
p_{29}	AGAGATTTCCC
p_{31}	AGATTTCCC
p_{33}	ATTTCCC
p_{27}	CAAGAGATTTCCC
	⋮

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!!
- \exists 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