

Lecture 1: Introduction to bioinformatics

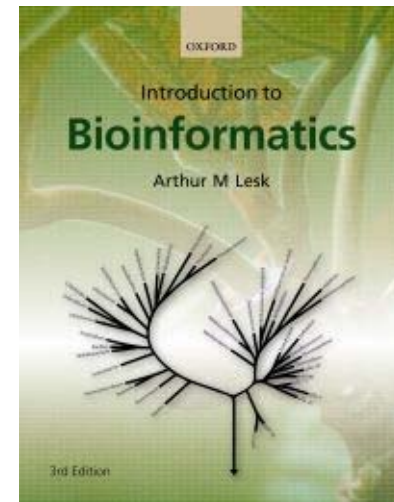
Torgeir R. Hvidsten

Professor
Norwegian University of Life Sciences

Guest lecturer
Umeå Plant Science Centre
Computational Life Science Centre (CLiC)

Course information (I)

- 10 Lectures
- 7 computer labs
- Book: Introduction to Bioinformatics.
Arthur M. Lesk, Oxford University Press.
- Credit points: 4 ECTS
- To pass:
 - attend lectures and labs
 - send lab to: david.sundell@plantphys.umu.se



Computer labs, MA316



David Sundell



Course information (II)

- Course webpage:
 - <http://www.trhvidsten.com/complife/2012>
- Here you can find the
 - course program
 - online resources
- and download
 - lecture slides
 - labs and suggestions for solutions to labs
 - additional material/examples research articles

Computational Life Science - Opera

File Edit View Bookmarks Feeds Tools Help

Downloads Google Reader (75) Computational Life Sci...

Web www.trhvidsten.com/complife/2012/index.html Search with Google

Aftenposten Dagbladet YR Yr Wikipedia Birro Football-italia Milano Siamo Noi TRH CiteULike IntraUMB SpruceWiki UPSC NetAlign

2012.09.20 Thursday	Lecture 1: 9-11 KBF30 Self-Study	Introduction to bioinformatics, computational biology and systems biology Introduction, Scientific publication and archives, Archives and information retrieval	Slides 1 Chapter 1, 3 and 4	WEEK 38
2012.09.21 Friday	Lecture 2: 9-11 KB4C10 Self-study	Programming in Perl: Introduction 1 Introduction, Scientific publication and archives, Archives and information retrieval	Slides 2 Chapter 1, 3 and 4	
2012.09.24 Monday	Computer LAB 1: 9-11 MA316 Lecture 3: 13-15 KBF30	Basic expressions, scalars, arrays, loops, conditions, file handling Programming in Perl: Introduction 2	Lab 1 (solutions) numbers.txt Perl tutorials Slides 3	WEEK 39
2012.09.25 Tuesday	Computer LAB 2: 9-11 MA316 Self-study	Hashes, data structures, references, subroutines, modules LAB 2 cont.	Lab 2 (solutions) microarray.txt network.txt	
2012.09.26 Wednesday	Lecture 4: 9-11 KBF30 Computer LAB 3: 13-15 MA316	Programming in Perl: Introduction 3 Bioperl	Slides 4 Lab 3 (solutions) hvidsten.com.html genpept.txt	WEEK 40
2012.09.27 Thursday	Computer LAB 3: 9-11 MA316 Lecture 5: 13-15 KBF30	LAB 3 cont. Algorithm design and time/space complexity analysis	Slides 5	
2012.09.28 Friday	Computer LAB 4: 13-15 MA316 Self-study	Pseudo-code, algorithm design and time/space complexity analysis LAB 3/ Lab 4	Lab 4 (solutions) promoters.txt MEME results (pdf) MEME results shuffled (pdf)	
2012.10.01 Monday	Lecture 6: 9-11 KBF30 Computer LAB 5: 13-15 MA316	Sequence alignment Sequence alignment	Slides 6 Chapter 5 Lab 5 (solutions)	WEEK 41
2012.10.02 Tuesday	Lecture 7: 9-11 KBF30 Computer LAB 6: 13-15 MA316	Evolutionary analysis, phylogenetic analysis Phylogenetic analysis	Slides 7 Chapter 2 and 5 Lab 6 get_sequences.pl change_names.pl hiv_ids.txt	
2012.10.03 Wednesday	Lecture 8: 9-11 KBF30 Self-study	Protein structure analysis Read the book	Slides 8 Chapter 6 Chapter 2, 5 and 6	WEEK 42
2012.10.04 Thursday	Lecture 9: 9-11 KBF30 Computer LAB 7: 13-15 MA316	Machine learning Machine learning	Slides 9 Lab 7 adenoca_markers.xls Dennis-ClinCancerRes2005.pdf	
2012.10.05 Friday	Lecture 10: 9-11 KB4C10	Systems biology	Slides 10 Course evaluation: eval If not present at the last lecture, mail the evaluation to: Thea	

http://www.trhvidsten.com/complife/2012/index.html

Course goals

At the end of this course you will :

- have basic knowledge of online bioinformatics resources (databases, ontologies, etc)
- know how to write and debug basic Perl programs and use online Perl libraries
- recognize the different algorithm design techniques and be able to do basic time/space complexity analysis
- know how to apply, and interpret results from, classical bioinformatics approaches such as pairwise and multiple alignment, phylogenetic analysis, and machine learning
- have a basic understanding of approaches applied in structural bioinformatics and systems biology

This lecture

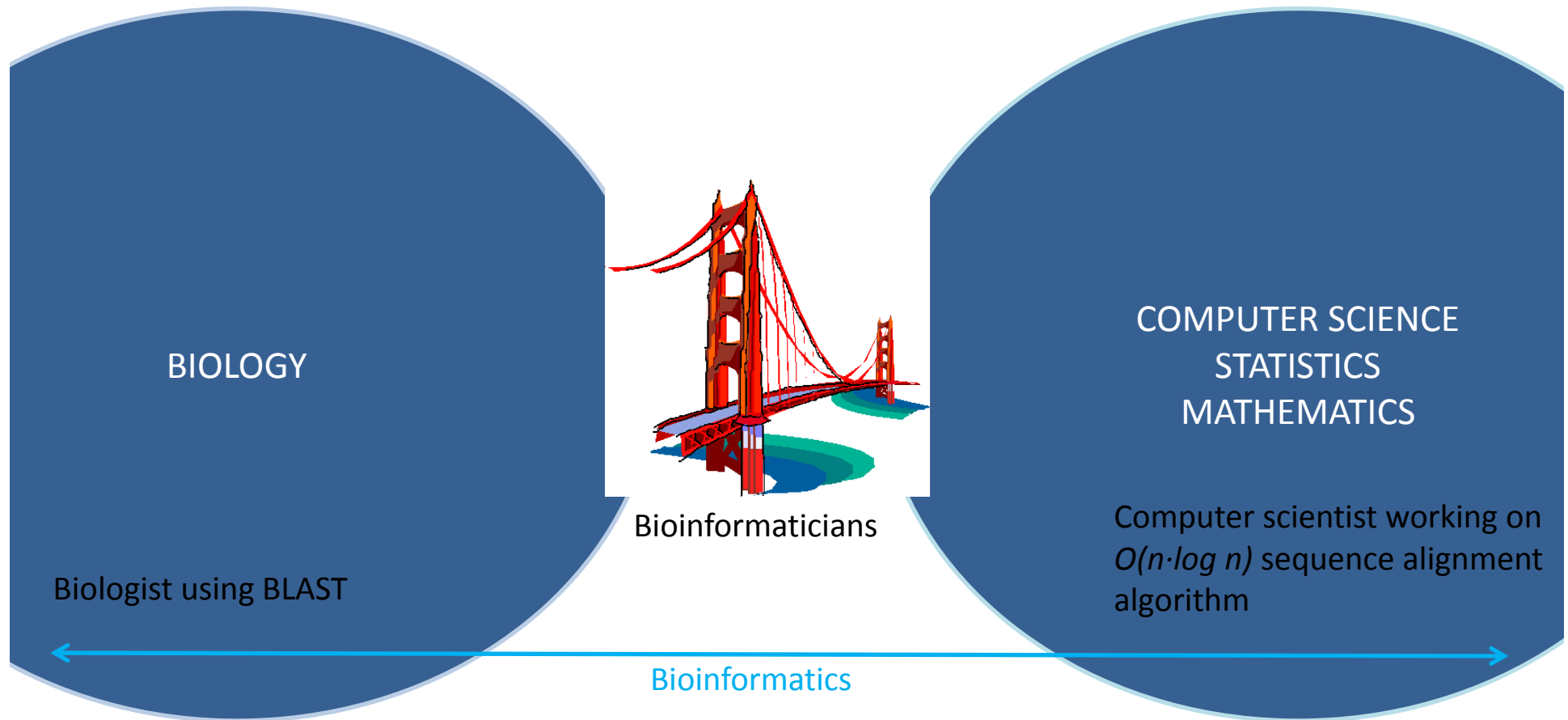
- Introduction to bioinformatics
- Introduction to the topics of this course
- Introduction to programming:
 - programming languages
 - pseudo-code



DNA sequence comparison: First success story of bioinformatics

- In 1984 Russell Doolittle and colleagues found similarities between a cancer-causing gene and a normal growth factor (PDGF) gene using a database search
- Finding sequence similarities with genes of known function is a common approach to predict the function of a newly sequenced gene
- “Bioinformatics will disappear and become an integrated part of biology” (Doolittle, 2002)

Bioinformatics



Definitions

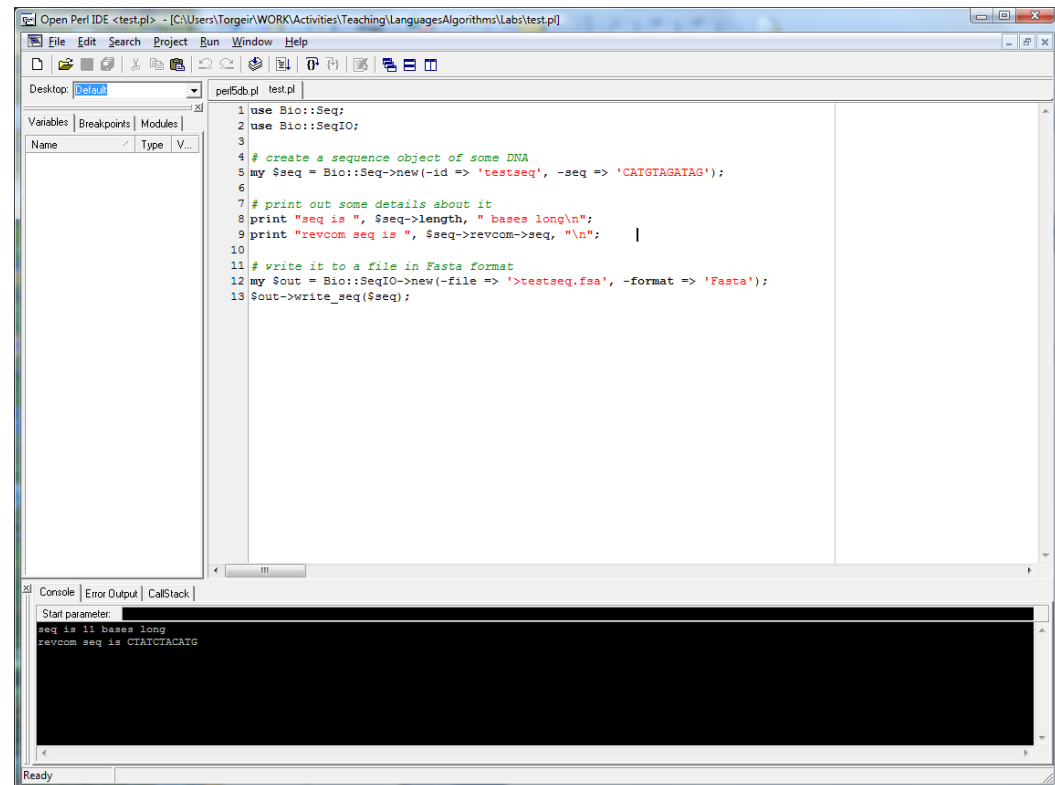
- *Bioinformatics*: Research, development, or application of computational tools and approaches for expanding the use of biological data, including those to acquire, store, organize, archive, analyze, or visualize such data.
- *Computational biology*: The development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological systems.
- *System biology*: the systematic study of complex interactions in biological systems (integration/holism instead of reduction)

The book

- Introduction to Bioinformatics. Arthur M. Lesk, Oxford University Press (2nd edition is online)
- Covers mostly basic bioinformatics (databases, online resources)
- Some of the lectures will be more advanced and towards computational biology/systems biology

Programming

- Lectures 2-4, Labs 1-3
- Do bioinformaticians need to know how to program?
- We will use the programming language Perl



The screenshot shows a Perl IDE window titled "Open Perl IDE <test.pl> - [C:\Users\Torgeir\WORK\Activities\Teaching\Languages\Algorithms\Labs\test.pl]". The main editor displays a Perl script with the following code:

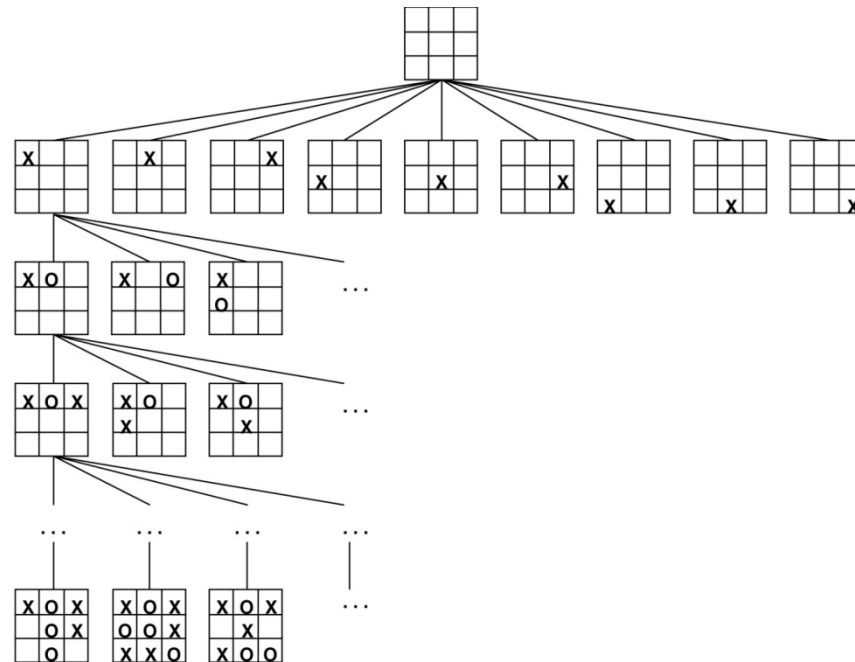
```
1 use Bio::Seq;
2 use Bio::SeqIO;
3
4 # create a sequence object of some DNA
5 my $seq = Bio::Seq->new(-id => 'testseq', -seq => 'CATGTAGATAG');
6
7 # print out some details about it
8 print "seq is ", $seq->length, " bases long\n";
9 print "revcom seq is ", $seq->revcom->seq, "\n";
10
11 # write it to a file in Fasta format
12 my $out = Bio::SeqIO->new(-file => '>testseq.fsa', -format => 'Fasta');
13 $out->write_seq($seq);
```

The console window at the bottom shows the output of the script:

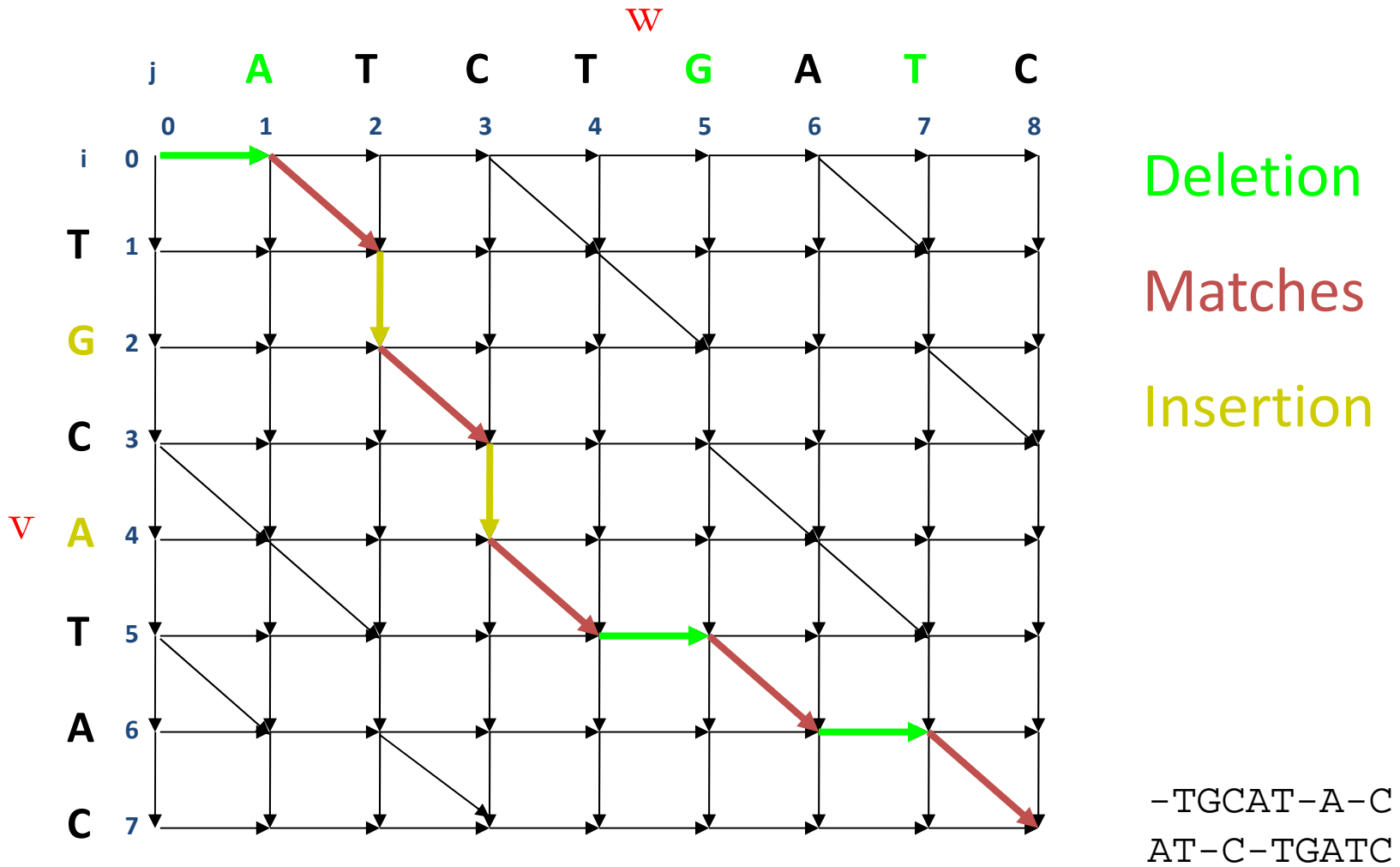
```
Start parameter:
seq is 11 bases long
revcom seq is CTATCTACATC
```

Algorithm designs

- Lecture 5, Lab 4
- Types of search algorithms
- Time/space complexity



Sequence alignment as a search problem



Algorithm design (I)

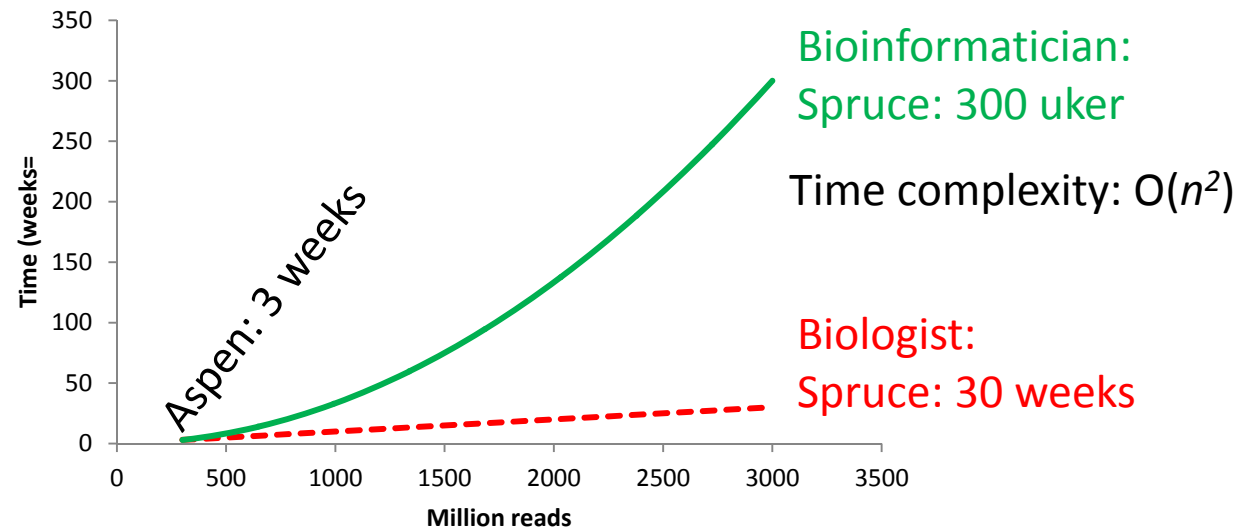
- Exhaustive algorithms (brute force): examine every possible alternative to find the solution
- Branch-and-bound algorithms: omit searching through a large number of alternatives by branch-and-bound or pruning
- Greedy algorithms: find the solution by always choosing the currently "best" alternative
- Dynamic programming: use the solution of the subproblems of the original problem to construct the solution

Algorithm design (II)

- Divide-and-conquer algorithms: splits the problem into subproblems and solve the problems independently
- Randomized algorithms: finds the solution based on randomized choices
- Machine learning: induce models based on previously labeled observations (examples)

Time complexity

- Genome assembly: pice together a genome from short reads ($\sim 200\text{bp}$)
 - Aspen: 300M reads
 - Spruce: 3000M reads
- Pair-wise all-against-all alignment for Aspen takes 3 weeks on 16 processors
- What about spruce?



Tractable versus intractable problems

- Some problems requires polynomial time
 - e.g. sorting a list of integers
 - called **tractable** problems
- Some problems require exponential time
 - e.g. listing every subset in a list
 - called **intractable** problems
- Some problems lie in between
 - e.g. the traveling salesman problem
 - called **NP-complete** problems
 - nobody have proved whether a polynomial time algorithm exists for these problems

Sequence alignment

- Lecture 6, Labs 3 and 5
- Pair-wise alignment (BLAST)

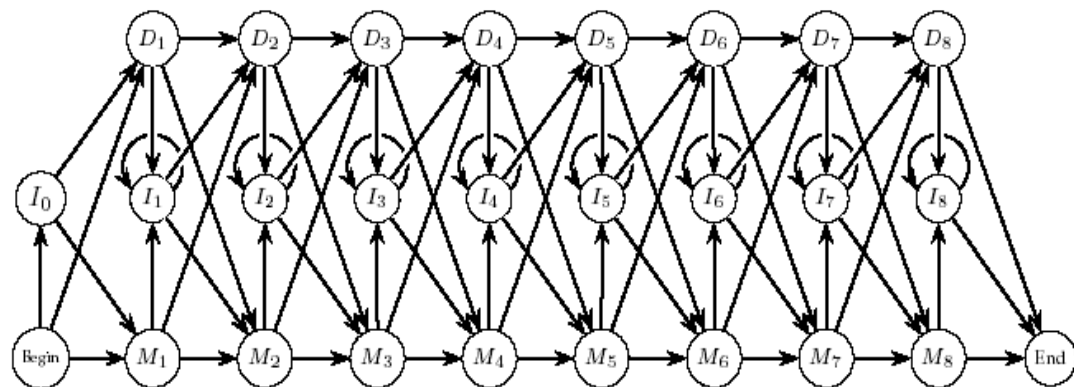
```

--T--CC-C-AGT--TATGT-CAGGGGACACG-A-GCATGCAGA-GAC
|   |   |   |   |   |   |   |   |   |   |   |   |
AATTGCCGCC-GTCGT-T-TTCAG-----CA-GTTATG-T-CAGAT--C
    
```

- Multiple alignment (PSI-BLAST, HMMs, pFAM, etc.)

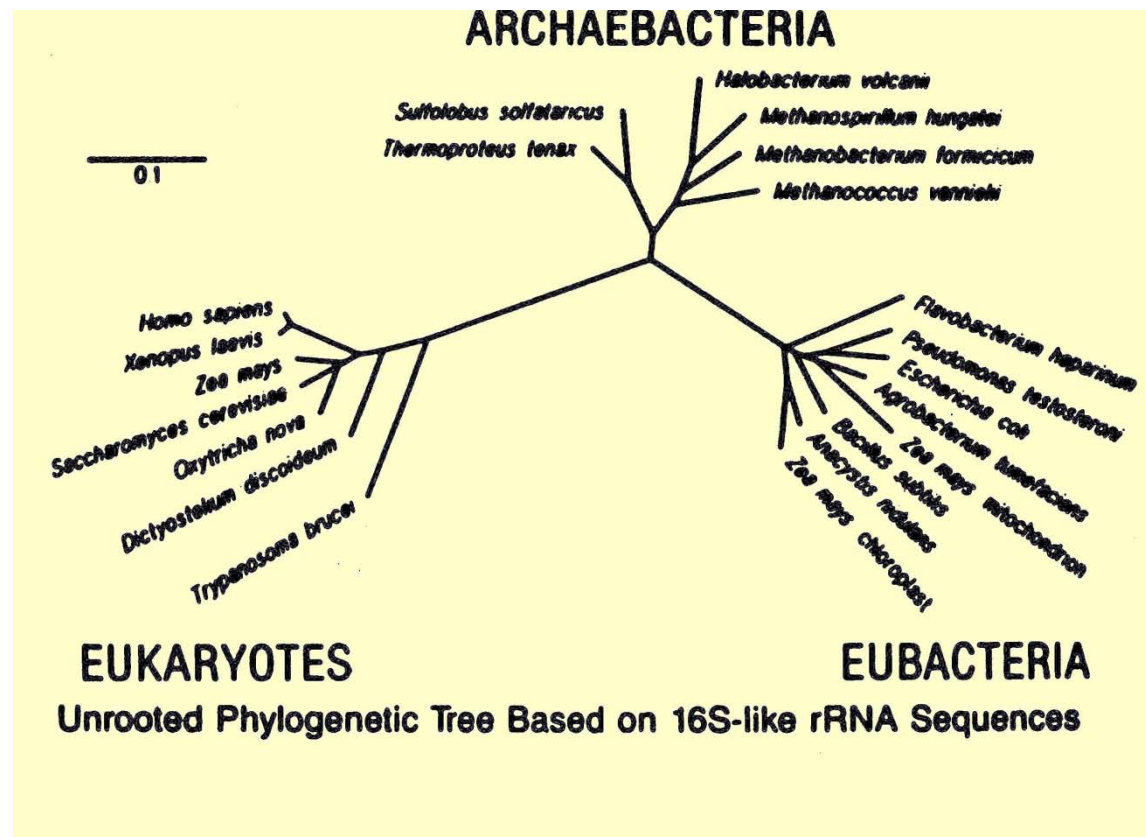
```

VTISCTGSSSNIGAG-NHVKQYQQLPG
VTISCTGTSSNIGS--ITVNWYQQLPG
LRLSCSSSGFIFSS--YAMYWVRQA--
LSLTCTVSG-SFDD--YYSTWVRQP--
PEVTCVVVD-SHEDPQVKFNWYVDG--
ATLVCLISDFYPGA--VTVAWKADS--
AALGCLVKD-FPEP--VTVSWNSG---
VSLTCLVKGFYPSD--IAVEWESNG--
    
```



Evolutionary analysis, phylogenetic analysis

- Lecture 7, Lab 6

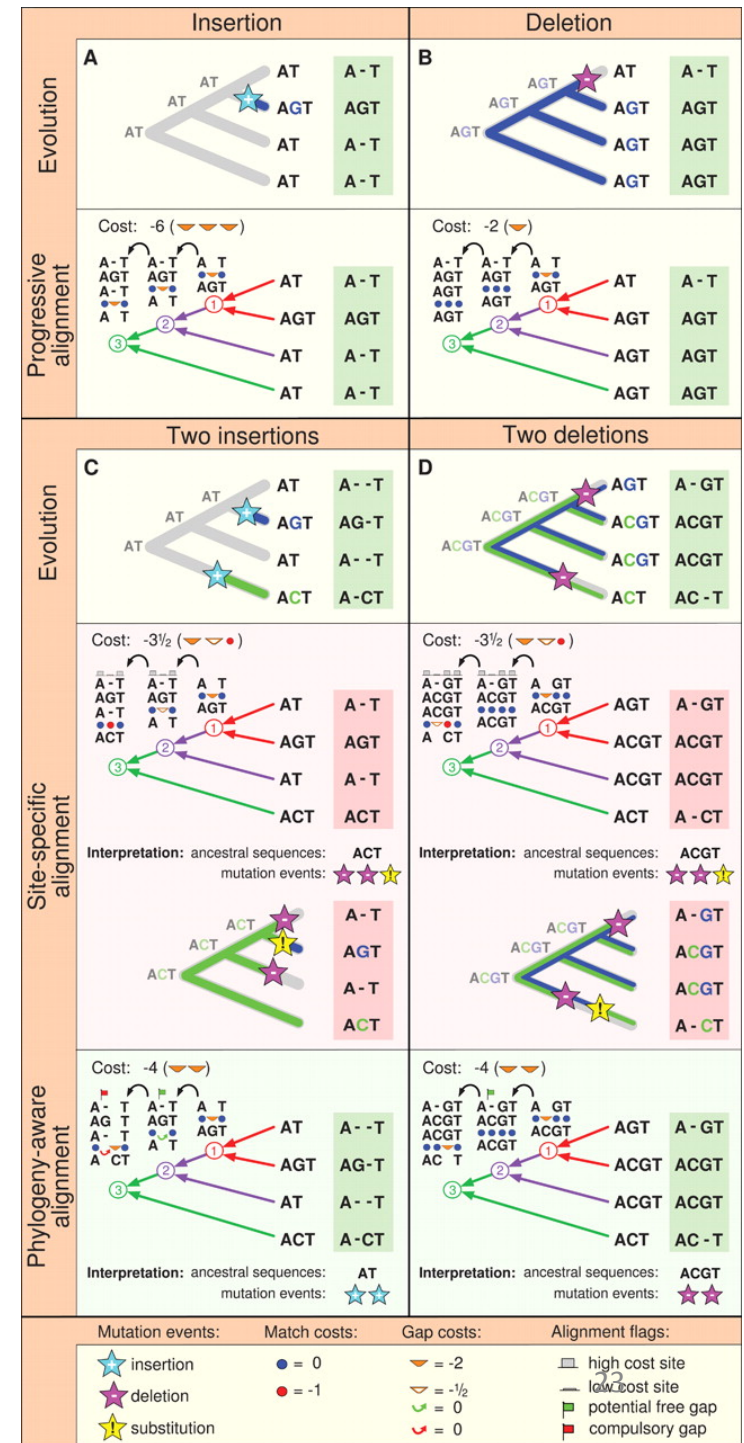


Phylogeny-aware gap placement

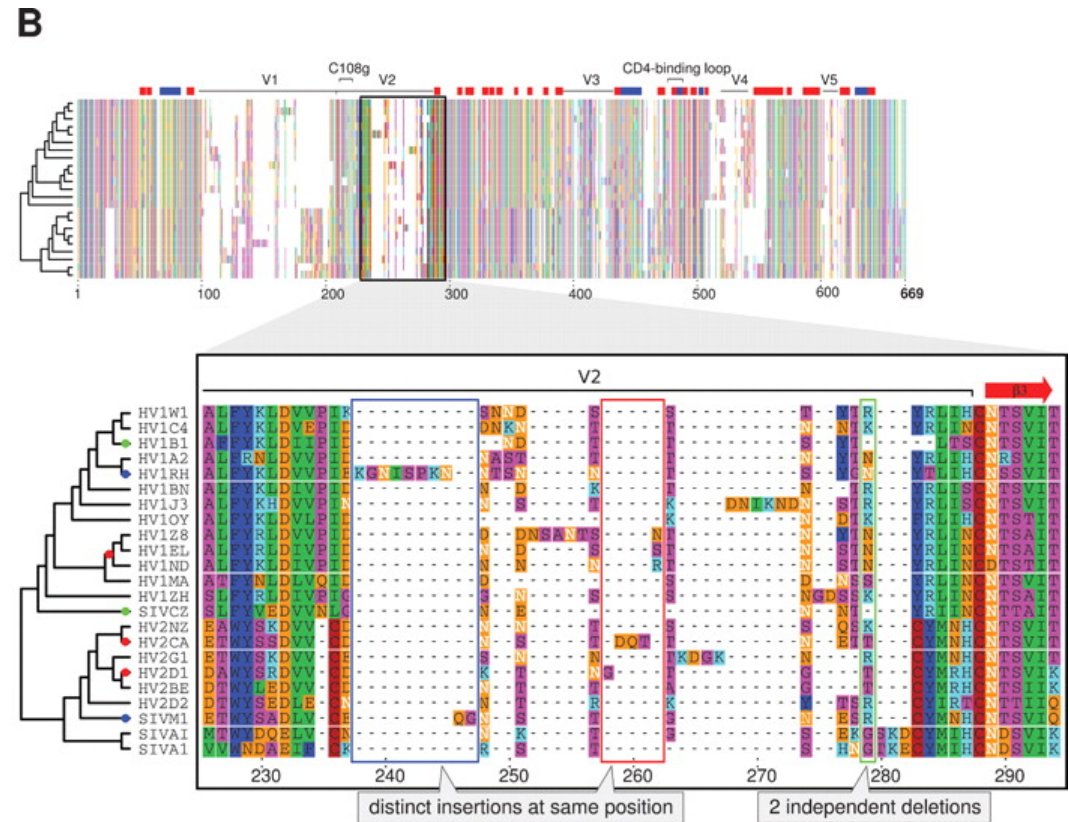
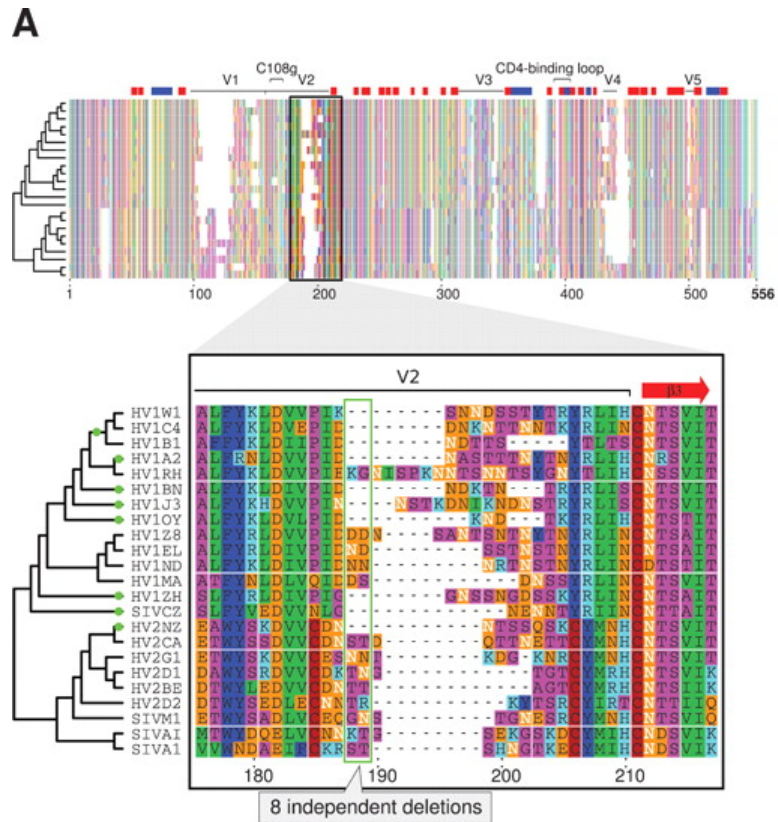
Conclusion:

“The resulting alignments may be fragmented by many gaps and may not be as visually beautiful as the traditional alignments, but if they represent correct homology, we have to get used to them.”

A. Löytynoja and N. Goldman. Phylogeny-Aware Gap Placement Prevents Errors in Sequence Alignment and Evolutionary Analysis. *Science* 320: 1632-35, 2008.

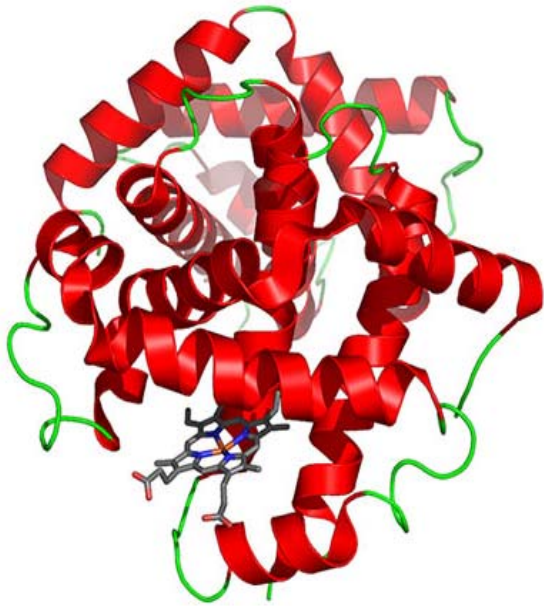


Phylogeny-aware gap placement



Protein structure analysis

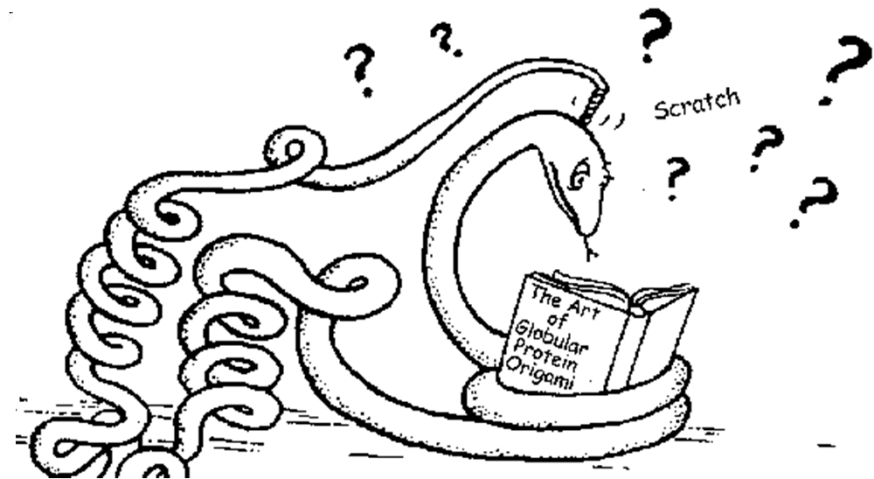
- Lecture 8
- The sequence, structure, function relationship
- The protein folding problem



Hemoglobin



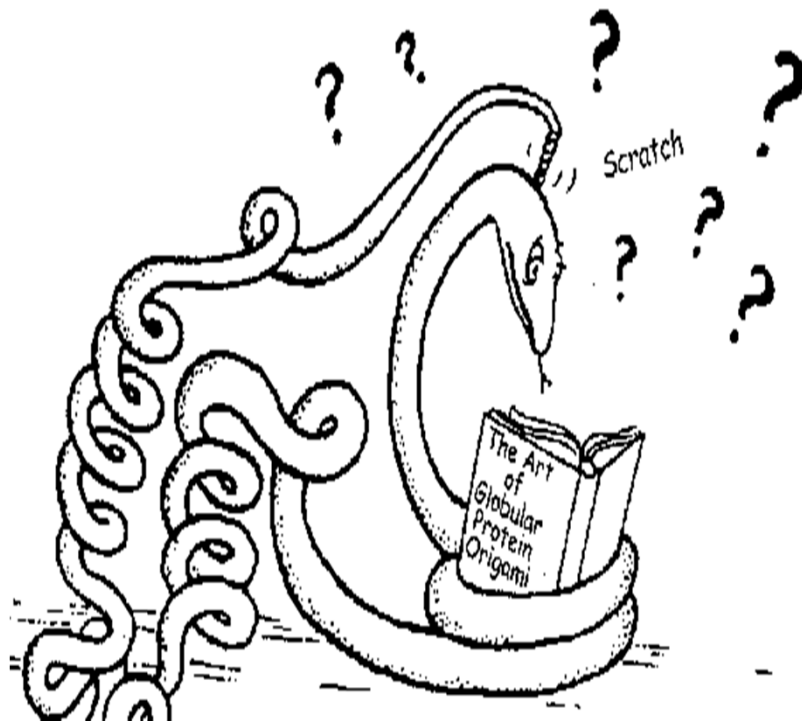
Hammer



The protein folding problem



Anfinsen's thermodynamic hypothesis (1973):
Protein folding is a **strictly physical** process, which
solely depends on the **protein sequence**

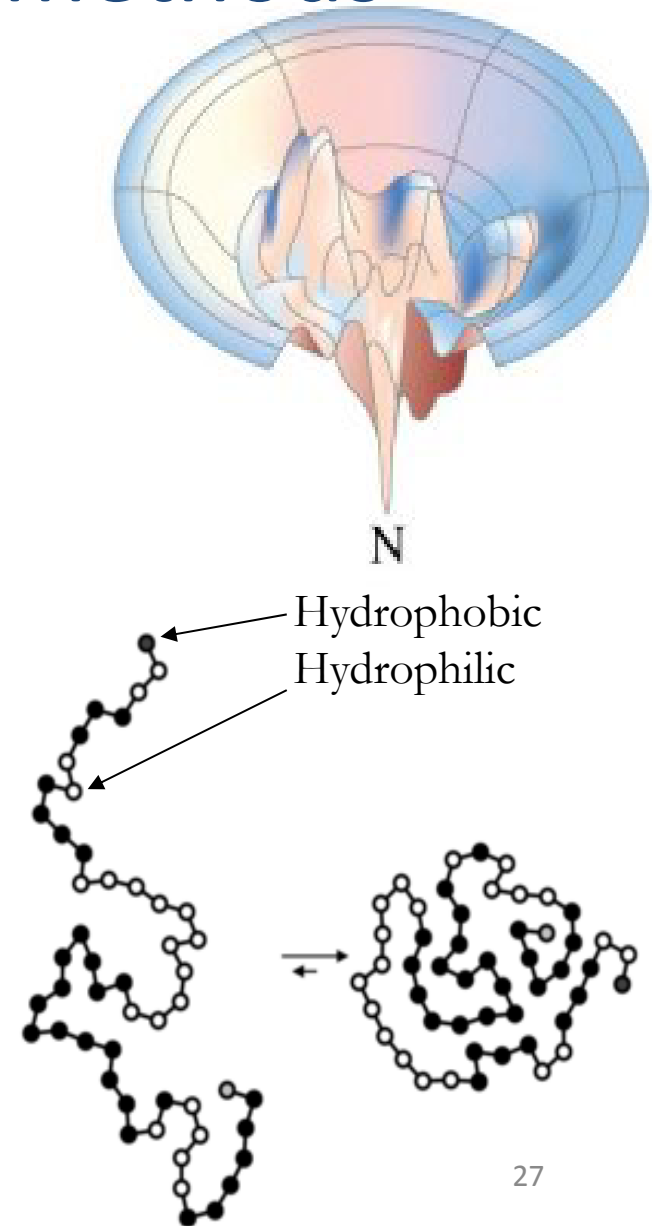


The folding problem:

discover nature's algorithm for
specifying 3D structure of proteins
from their amino acid sequences

Computational folding methods

- No effective folding machine exists that is based on physical principles and energy minimization alone
- Current computational methods rely on known protein structures –
machine learning approach:
 - Template-based modeling
 - Template-free modeling



Ab initio prediction

AVGIFRAAVCTRGVAKAVDFVP...

AVGIFR

AAVCTR

GVAKAVDF



Score and select model

Machine learning

- Lecture 9, Lab 7
- Induction of general models from data
 - Bayes decision rule
 - Decision trees
 - Nearest neighbour approach
 - Artificial neural networks (linear versus non-linear methods)
 - Genetic algorithms and genetic programming
 - Model evaluation

What is AI?

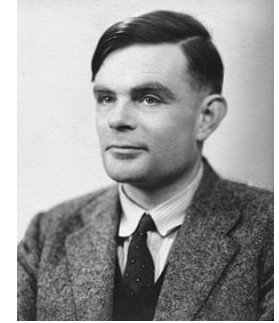
“...making a machine behave in ways that would be called intelligent if a human were so behaving”

- John McCarthy, August 31, 1955

“The subfield of computer science concerned with the concepts and methods of symbolic inference by computer and symbolic knowledge representation for use in making inferences.”

- The Free On-line Dictionary of Computing (September 27, 2003)

Acting humanly: Turing test



1912-1954

- Turing proposed that a computer program show intelligent behavior if it is able to fool a human interrogator:
- **The Turing test:** the computer is interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end
 - natural language processing
 - knowledge representation
 - automated reasoning
 - machine learning

Definitions of AI

Four categories of definitions:

Empirical science

Engineering

Human-centered

Rationality-centered

Reasoning

Systems that think like
humans

**Systems that think
rationally**

Behavior

Systems that act like
humans

**Systems that act
rationally**

Reasoning	Systems that think like humans	Systems that think rationally
Behavior	Systems that act like humans	Systems that act rationally

AI techniques

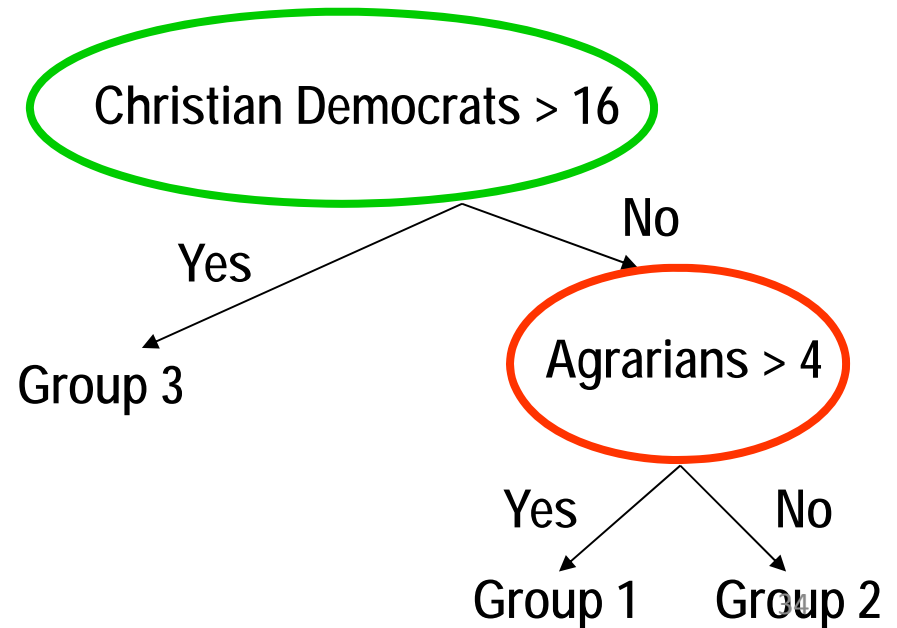
- Logics
- Knowledge representation
- Search
- Machine learning
- Pattern recognition
- Automatic theorem proving
- Planning
- Machine vision
- Natural language processing

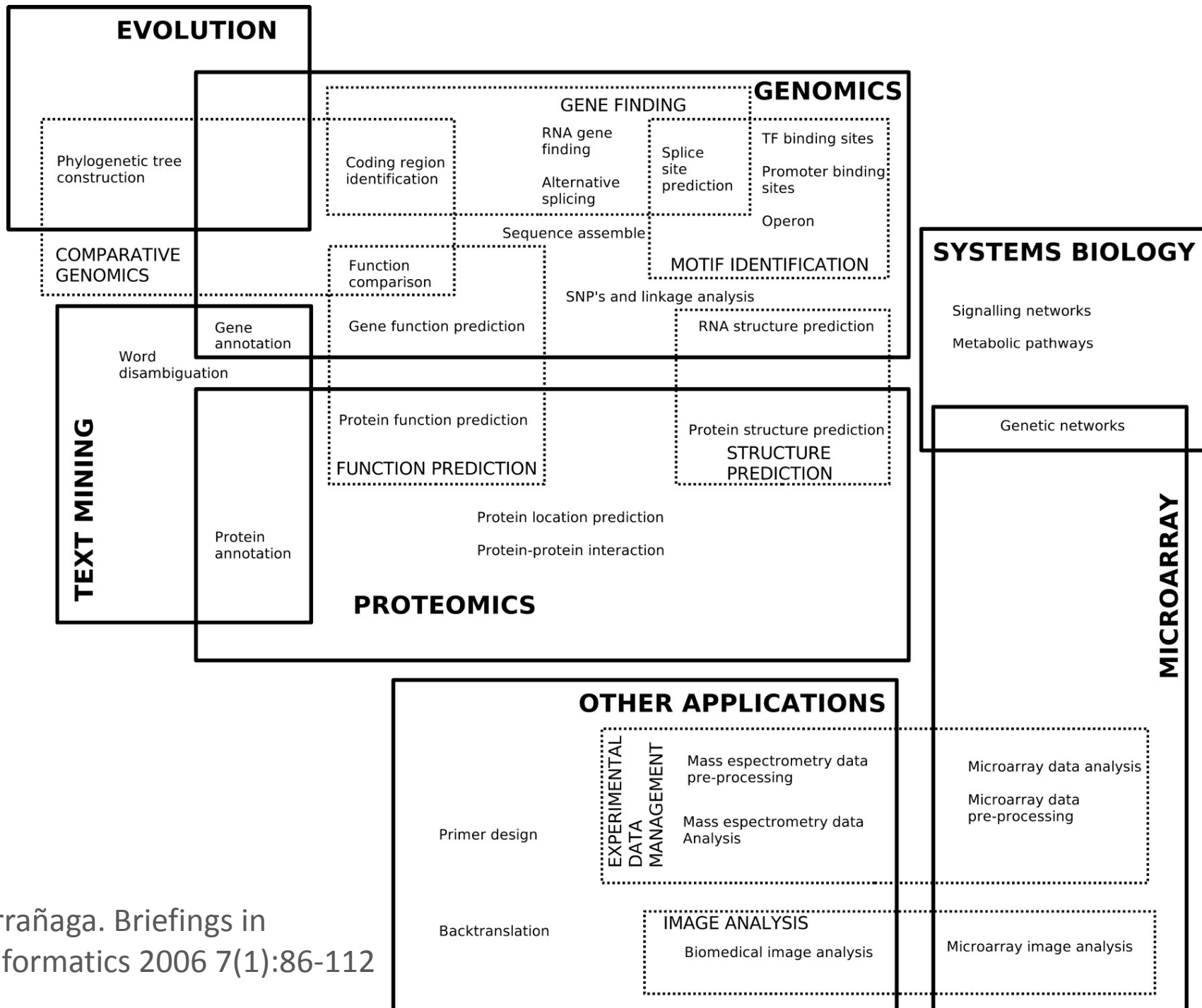
Example: Decision tree learning

Country	Communists	Socialists	Greens	Social Democrats	Liberals	Agrarians	Subnational, regional and ethnic parties	Christian Democrats	Conservatives	Extreme Right
Norway	0	7	0	38	4	8	0	9	24	6
Sweden	6	0	2	43	10	17	0	2	18	1
Denmark	4	9	0	33	13	14	0	3	15	9
Finland	15	0	2	24	3	25	5	3	21	0
Iceland	0	18	3	16	4	22	0	0	36	0
UK	0	0	9	39	15	0	4	0	42	0
Netherlands	2	5	0	30	23	0	0	37	0	0
Belgium	2	0	4	27	19	0	14	31	0	2
Luxembourg	6	1	3	31	21	0	0	34	0	1
Switzerland	2	2	7	22	23	11	0	22	3	5
Austria	1	0	2	48	0	0	0	41	0	8
Germany	1	0	3	40	9	0	0	46	0	1
France	15	2	2	28	20	0	0	0	25	5
Italy	29	0	3	15	4	0	3	35	2	6
Greece	10	0	0	39	6	0	0	0	44	0
Spain	8	0	0	39	16	0	10	0	21	0
Portugal	15	0	1	31	38	0	0	1	11	0

Class knowledge:

- Group 1: Nordic countries
- Group 2: UK, France, Greece, Spain, Portugal
- Group 3: Benelux countries, Switzerland, Austria, Italy, Germany



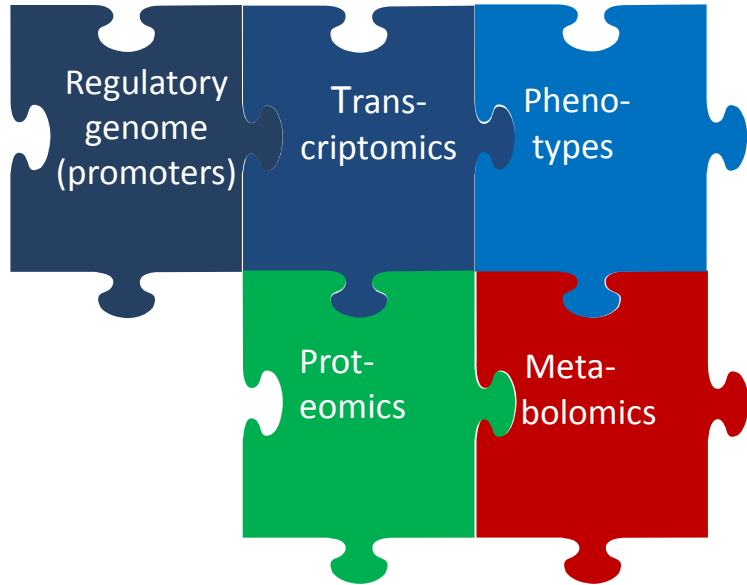


P. Larrañaga. Briefings in Bioinformatics 2006 7(1):86-112

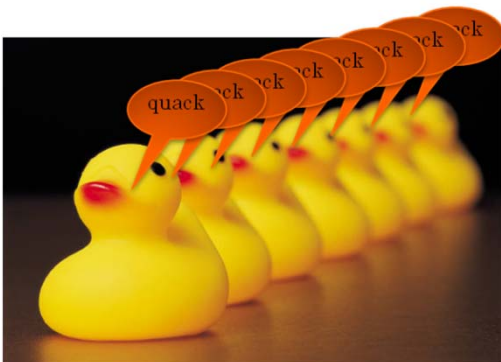
Systems biology

- Lecture 10
- The systematic study of complex interactions in biological systems (integration and holism instead of reduction)

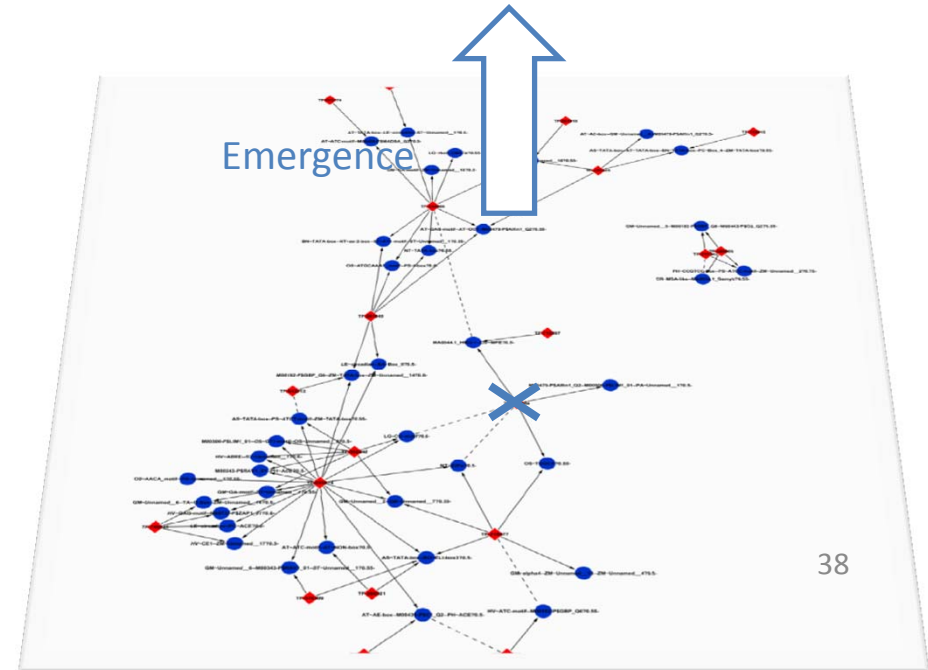
Systems biology



Synergy from integration

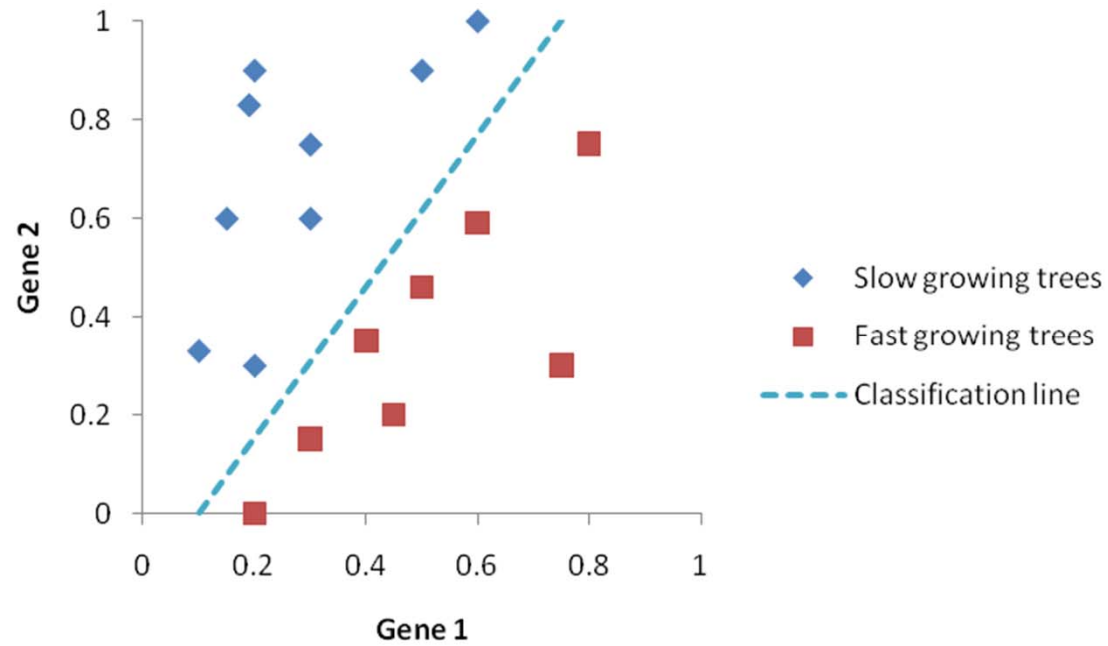


Phenotypes

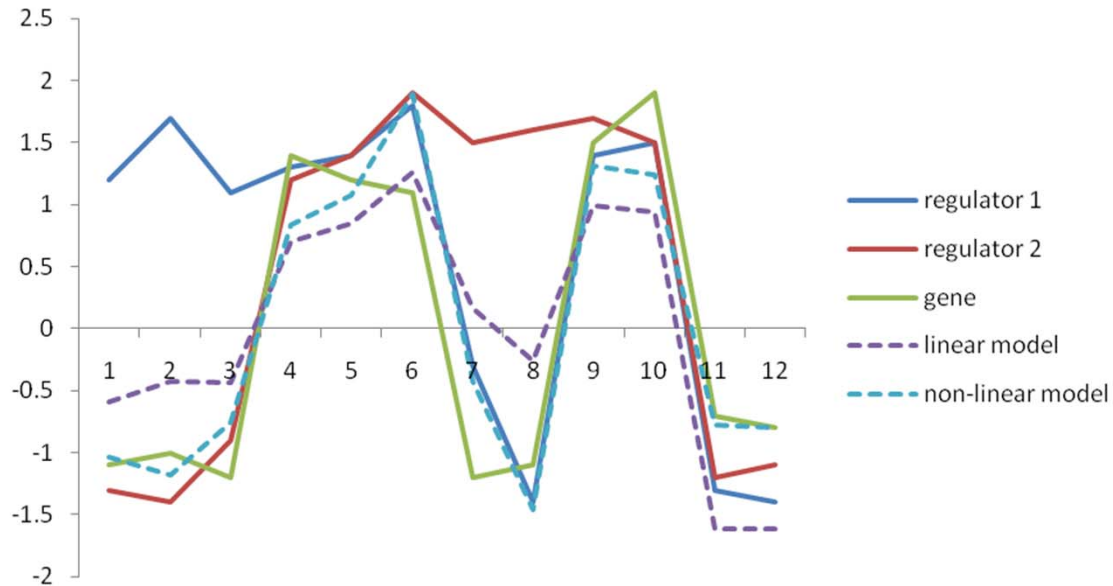


Interacting genes/protein/metabolites

Emergent properties: differential expression



Emergent properties: AND logics in regulation



Correlation between the gene and

regulator 1:

0.55 (P < 0.06)

regulator 2:

0.65 (P < 0.02)

Correlation between the gene and

linear model:

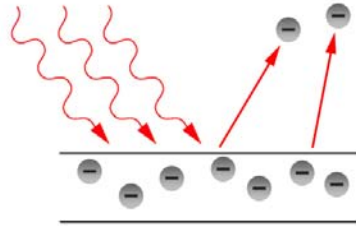
0.77 (P < 0.003)

non-linear model:

0.91 (P < 1.2E-05)

Physics

Photoelectric effect



$$E = hv$$

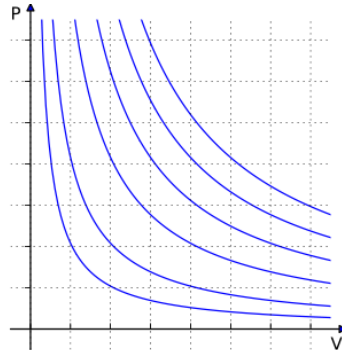
E = energy

H = Planck's constant

v = frequency of light radiation

Chemistry

Ideal gas law



$$PV = nRT$$

P = absolute pressure

V = volume of the vessel

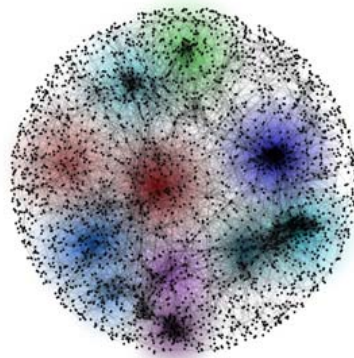
n = number of moles of gas

R = ideal gas constant

T = absolute temperature

~~Biology~~

~~Gene interactions~~



$$y_i = \alpha_i + \sum_{j=0}^n \beta_{ij} y_j$$

~~y_i = gene expression of gene i~~

~~n = number of genes~~

~~α = transcription rate~~

~~β_{ij} = effect of gene j on gene i~~



Systems biology

Scale free networks

Reductionism: «one gene at a time»

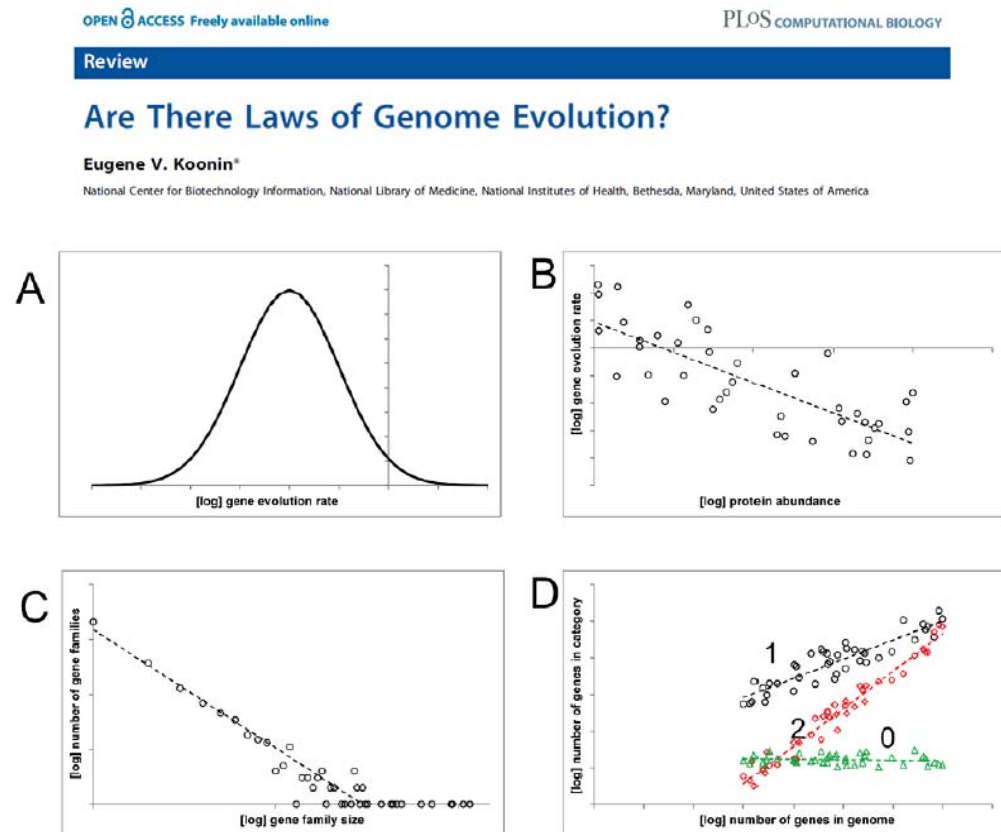
$$P(k) \sim k^{-\gamma}$$

k = node degree

γ = degree exponent

Laws of genome evolution

- A. Log-normal distribution of the evolutionary rates between orthologous genes
- B. Negative correlation between gene sequence evolution rate and expression level (or protein abundance)
- C. Power law–like distributions of membership in paralogous gene families and node degree in biological networks
- D. Distinct scaling of functional classes of genes with genome size



- 0. No dependence: translation
- 1. Linear dependence: enzymes
- 2. Quadratic dependence: regulation/signaling

Computer programming

Algorithm

- **Algorithm**: a sequence of instructions that one must perform in order to solve a well-formulated problem
- **Correct algorithm**: translate every input instance into the correct output
- **Incorrect algorithm**: there is at least one input instance for which the algorithm does not produce the correct output
- Many successful algorithms in bioinformatics are incorrect algorithms

Programs

- Algorithms are implemented in a programming language to form **programs**
- Programs consists of:
 - Variables: names with values (float, integer, string) or arrays/tables/ashes of values
 - Conditional statements: IF-THEN-ELSE
 - Loops: while, for, until, etc.
 - Modularity: procedures/functions/sub-routines/objects/methods
- Pseudo-code: programming language-independent, often used to sketch a program using pen and paper

Pseudo-code

Sorting problem: Sort a list of n integers:

$\mathbf{a} = (a_1, a_2, \dots, a_n)$ e.g. $\mathbf{a} = (7, 92, 87, 1, 4, 3, 2, 6)$

SelectionSort(\mathbf{a}, n)

- 1 **for** $i \leftarrow 1$ **to** $n-1$
- 2 $j \leftarrow$ Index of the smallest element
 among a_i, a_{i+1}, \dots, a_n
- 3 Swap elements a_i and a_j
- 4 **return** \mathbf{a}

Example run

$i = 1:$ (7,92,87,1,4,3,2,6)
 $i = 2:$ (1,92,87,7,4,3,2,6)
 $i = 3:$ (1,2,87,7,4,3,92,6)
 $i = 4:$ (1,2,3,7,4,87,92,6)
 $i = 5:$ (1,2,3,4,7,87,92,6)
 $i = 6:$ (1,2,3,4,6,87,92,7)
 $i = 7:$ (1,2,3,4,6,7,92,87)
(1,2,3,4,6,7,87,92)

Syntax versus semantics

- **Syntax**: the rules for constructing valid statements in a programming language
- **Semantics**: the meaning of a program
- A specific algorithm implemented in different programming languages would use different syntax, but have the same semantics
- Syntax is easy and can be checked before execution (the interpreter will tell you when you make syntax mistakes)
- Semantics is hard and "bugs" typically only reveal themselves at execution time

Programming languages

- **Imperative programming**: describes computation as statements that change a program state (e.g. Perl, Fortran, C, and Java)
- **Functional programming**: treats computation as the evaluation of (mathematical) functions, and often avoids state (e.g. LISP)
- **Declarative programming**: while imperative programs explicitly specify an algorithm to achieve a goal, declarative programs explicitly specify the goal and leave the implementation of the algorithm to the support software (e.g. PROLOG)

Sorting: imperative

Sorting problem: Sort a list of n integers:

$$\mathbf{a} = (a_1, a_2, \dots, a_n)$$

SelectionSort(\mathbf{a}, n)

- 1 **for** $i \leftarrow 1$ **to** $n-1$
- 2 $j \leftarrow$ Index of the smallest element
 among a_i, a_{i+1}, \dots, a_n
- 3 Swap elements a_i and a_j
- 4 **return** \mathbf{a}

Pseudo-code hides ugly details such as

“Swap elements a_i and a_j .”

1 $tmp \leftarrow a_j$

2 $a_j \leftarrow a_i$

3 $a_i \leftarrow tmp$

or

“ $j \leftarrow$ Index of the smallest element among a_i, a_{i+1}, \dots, a_n ”

IndexOfMin(**array**, *first*, *last*)

```
1  index  $\leftarrow$  first
2  for  $k \leftarrow$  first + 1 to last
3      if  $array_k < array_{index}$ 
4          index  $\leftarrow$   $k$ 
5  return index
```

Remember, though, that **the devil is in the details!**

Recursion

RecursiveSelectionSort($\mathbf{a}, first, last$)

- 1 **if** ($first < last$)
- 2 $index \leftarrow$ Index of the smallest element
 among $a_{first}, a_{first+1}, \dots, a_{last}$
- 3 Swap elements a_{first} and a_{index}
- 4 $\mathbf{a} \leftarrow$ RecursiveSelectionSort($\mathbf{a}, first+1, last$)
- 5 **return a**

Example I

Write pseudo-code for a program that solves a quadratic equation $ax^2 + bx + c = 0$:

QuadraticEquationSolver (a, b, c)

Remember that: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

QuadraticEquationSolver(a, b, c)

1 $\text{root} \leftarrow b^2 - 4ac$;

2 **if** $\text{root} < 0$

3 **return** “No solution”

4 $x_1 \leftarrow \frac{-b + \sqrt{\text{root}}}{2a}$

5 $x_2 \leftarrow \frac{-b - \sqrt{\text{root}}}{2a}$

6 **if** $x_1 = x_2$

7 **output** “Solution: $x = x_1$ ”

8 **else**

9 **output** “Solutions: $x = x_1$ or $x = x_2$ ”

Example II

Write pseudo-code for a program that adds a constant to every number in an array $\mathbf{a} = (a_1, a_2, \dots, a_n)$:

AddConstant (\mathbf{a}, n, c)

E.g. $\mathbf{a} = (1, 2, 4)$ and $c = 3$ outputs $(4, 5, 7)$

AddConstant(**list**, n , c)

1 **for** $i \leftarrow 1$ **to** n

2 $list_i \leftarrow list_i + c$

3 **return** **list**

Example III

Write pseudo-code for a program that remove duplicates in an array $\mathbf{a} = (a_1, a_2, \dots, a_n)$

RemoveDuplicates (\mathbf{a}, n)

E.g. $\mathbf{a} = (1, 2, 2, 4, 4)$ outputs $(1, 2, 4)$

RemoveDuplicates(**list**, n)

1 **newlist** $\leftarrow ()$

2 **for** $i \leftarrow 1$ **to** n

3 $m \leftarrow$ length of **newlist**

4 $foundDuplicate \leftarrow false$

5 **for** $j \leftarrow 1$ **to** m

6 **if** $list_i = newList_j$

7 $foundDuplicate = true$

8 **break**

9 **if** $foundDuplicate = false$

10 add $list_i$ to **newlist**

11 return **newlist**

Example IV

Write pseudo-code for a program that counts from 0 to \mathbf{n}
= (n_1, n_2, \dots, n_m) :

Count (\mathbf{n}, m)

E.g. $\mathbf{n} = (1, 2)$ outputs:

00
01
02
10
11
12

Count(\mathbf{n} , m)

1 $\mathbf{c} \leftarrow (0, 0, \dots, 0)$

2 **while forever**

3 **for** $i \leftarrow m$ **to** 1

4 **if** $c_i = n_i$

5 $c_i \leftarrow 0$

6 **else**

7 $c_i \leftarrow c_i + 1$

8 **break**

9 **output** \mathbf{c}

10 **if** $\mathbf{c} = (0, 0, \dots, 0)$

11 **break**