Bioinformatics (MVE360)

Course Organiser: Graham Kemp

http://www.cse.chalmers.se/edu/year/2012/course/MVE360/

Assessment

- Grades will be determined by a written exam at the end of the course.
- But in order to pass the course you must also submit solutions to specified exercises.

Content

The course covers basic methods used in sequence analysis such as pairwise and multiple alignment, searching databases for sequence similarity, profiles, pattern matching, hidden Markov models, RNA bioinformatics, gene prediction methods and principles for molecular phylogeny.

The course includes modern high-throughput sequencing techniques and their applications, as well as molecular biology databases and different systems to query such databases.

The course considers theoretical principles as well as how existing programs are being used by bioinformaticians.

Learning outcome/goals

- understand the use of bioinformatics in addressing a range of biological questions
- describe how bioinformatics methods can be used to relate sequence, structure and function
- discuss the technologies for modern high-throughput DNA sequencing and their applications
- use and understand some central bioinformatics data and information resources
- know principles and algorithms of pairwise and multiple alignments, and sequence database searching
- perform pattern matching in biomolecular sequences
- describe how evolutionary relationships can be inferred from sequences (phylogenetics)
- understand the most important principles in gene prediction methods
- know basic principles of hidden Markov models and their application in sequence analysis
- understand and implement solutions to basic bioinformatics problems

Personnel

Graham Kemp
- General interests in bioinformatics
- Structural bioinformatics (another course, TDA506)

Marina Axelson-Fisk
- Hidden Markov Models (HMMs)
- Biological sequence analysis
- Cross-species gene finding

Erik Kristiansson
- Microarray gene expression
- Large-scale sequence analysis
- Metagenomics

Sequence alignment

Comparison of macromolecular sequences.

Nucleic acids (DNA, RNA) or proteins.

Assignment of nucleotide-nucleotide or residue-residue correspondences.

Suggest evolutionary, structural and functional relationships.

Rigorous algorithms for global and local alignment.

Heuristic algorithms for practical database searching.

Dotplots

A pictorial representation of the similarity between two sequences.

Compare a sequence with itself:
- Repeats
- Palindromic sequences

Compare two sequences:
- Any path from upper left to lower right represents an alignment.
- Horizontal or vertical moves correspond to gaps in one of the sequences.
- Path with highest score corresponds to an optimal alignment.

Measures of sequence similarity

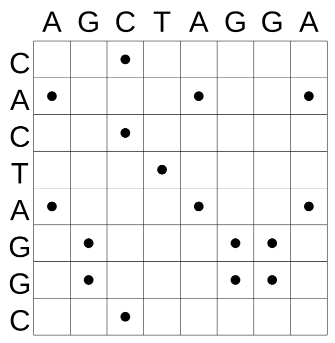
Hamming distance:
- Number of positions with mismatching characters.
- Defined for two strings of equal length.

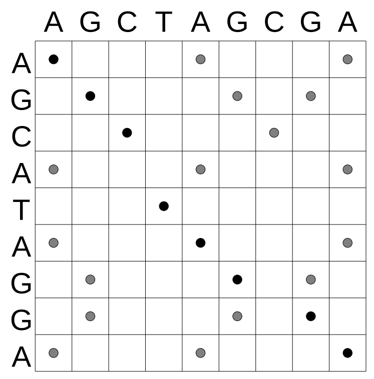
```
agtc
cgta
```

Levenshtein distance:
- Minimum number of edit operations (delete, insert, change a single character) needed to change one sequence into another.

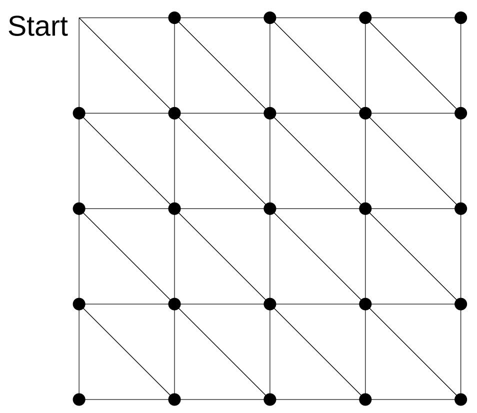
```
agtcc
cgctca
```

Dotplots





How many paths?



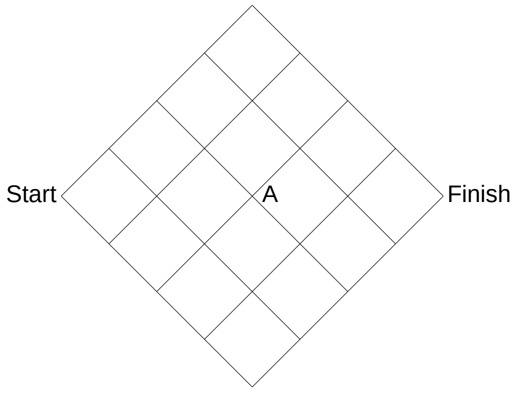
Pairwise global alignment (Needleman-Wunsch algorithm)

Rigorous algorithms use dynamic programming to find an optimal alignment.

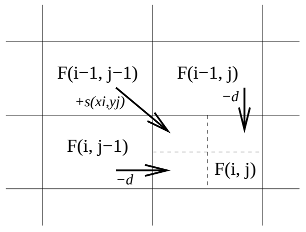
- match score
- mismatch score
- gap penalty

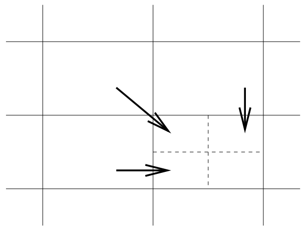
$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

Do we have to enumerate all paths?

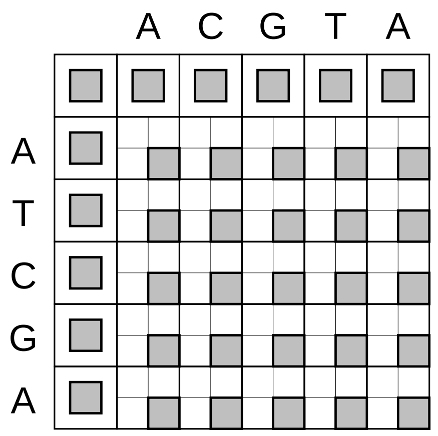


Dynamic programming





Score matrix



Percent identity

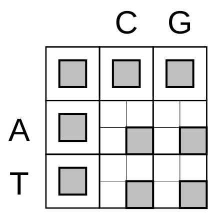
Having obtained an alignment, it is common to quantify the similarity between a pair of sequences by stating the percent identity.

Count the number of alignment positions with matching characters and divide by ... *what*?

- the length of the shortest sequence?
- the length of the alignment?
- the average length of the sequences?
- the number of non-gap positions?
- the number of equivalenced positions excluding overhangs?

Sequences are either homologous (i.e. they share a common evolutionary ancestor) or they are not.
The phrase “percent homology” is meaningless!

How many ways can “AT” be aligned with “CG”?



Two diagonal moves:

```
AT
CG
```

One diagonal move:

```
AT−   A−T
C−G   CG−

AT−   −AT
−CG   CG−

A−T   −AT
−CG   C−G
```

No diagonal moves:

```
AT−−   A−T−   A−−T
−−CG   −C−G   −CG−

−−AT   −A−T   −AT−
CG−−   C−G−   C−−G
```

Pairwise local alignment (Smith-Waterman algorithm)

Local similarities may be masked by long unrelated regions.

A minor modification to the global alignment algorithm.

- If the score for a subalignment becomes negative, set the score to zero.

$$F(i,\, j) = \max \begin{cases} 0 \\ F(i-1,\, j-1) + s(x_i, y_j) \\ F(i-1,\, j) - d \\ F(i,\, j-1) - d \end{cases}$$

- Trace back from the position in the score matrix with the highest value.
- Stop at cell where score is zero.

“Scripting: Higher Level Programming for the 21st Century” (John Ousterhout)

http://www.tcl.tk/doc/scripting.html

For the last fifteen years a fundamental change has been occurring in the way people write computer programs. The change is a transition from system programming languages such as C or C++ to scripting languages such as Perl or Tcl. Although many people are participating in the change, few people realize that it is occurring and even fewer people know why it is happening. This article is an opinion piece that explains why scripting languages will handle many of the programming tasks of the next century better than system programming languages.

Scripting languages are designed for different tasks than system programming languages, and this leads to fundamental differences in the languages.

“Scripting: Higher Level Programming for the 21st Century” (John Ousterhout)

In deciding whether to use a scripting language or a system programming language for a particular task, consider the following questions:

- Is the application’s main task to connect together pre-existing components?
- Will the application manipulate a variety of different kinds of things?
- Does the application include a graphical user interface?
- Does the application do a lot of string manipulation?
- Will the application’s functions evolve rapidly over time?
- Does the application need to be extensible?

"Yes" answers to these questions suggest that a scripting language will work well for the application.

“Scripting: Higher Level Programming for the 21st Century” (John Ousterhout)

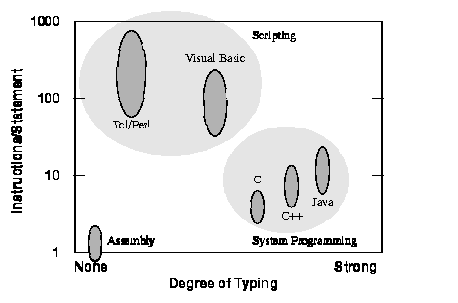


Figure 1. A comparison of various programming languages based on their level (higher level languages execute more machine instructions for each language statement) and their degree of typing. System programming languages like C tend to be strongly typed and medium level (5-10 instructions/statement). Scripting languages like Tcl tend to be weakly typed and very high level (100-1000 instructions/statement).

“Scripting: Higher Level Programming for the 21st Century” (John Ousterhout)

"Yes" answers to the following questions suggest that an application is better suited to a system programming language:

- Does the application implement complex algorithms or data structures?
- Does the application manipulate large datasets (e.g. all the pixels in an image) so that execution speed is critical?
- Are the application’s functions well-defined and changing slowly?

Scripting and system programming are symbiotic. Used together, they produce programming environments of exceptional power: system programming languages are used to create exciting components which can then be assembled using scripting languages.