ICB Fall 2009

G4120: Introduction to Computational Biology

Oliver Jovanovic, Ph.D.
Columbia University
Department of
Microbiology & Immunology

Lecture 4
Introduction to
DNA Analysis
October 15, 2009

Intrinsic and Extrinsic Sequence Analysis

Intrinsic Sequence Analysis

Evaluating sequence properties without explicitly referring to other sequences, using a derived formula, sequence or value.

With respect to DNA sequences, this includes calculating %G+C values, analyzing the information content of the sequence, or comparing the sequence against itself for the presence of direct or inverted repeats.

Extrinsic Sequence Analysis

Evaluating sequence properties by explicitly comparing them to other sequences or sets of sequences.

With respect to DNA sequences, this includes sequence comparison, either pairwise dot matrix, pairwise optimal (Needleman-Wunsch or Smith-Waterman), pairwise heuristic (BLAST, BLAT, FASTA) or multiple sequence alignment to detect homology, insertions or deletions, as well as statistical or phylogenetic analysis.

A common mistake in sequence analysis is to use only one of these approaches. Combining intrinsic and extrinsic computational approaches can give results neither alone can approach.

Information Content

Uncertainty

Uncertainty can be thought of as the number of yes/no questions required to identify the state something is in. It can be measured in bits.

- A coin toss, with only 2 possibilities, can be identified with a single question (i.e., "Is it heads?")
- A nucleotide, with 4 possibilities, can be identified with two questions (i.e. "Is it a purine? Is it adenine?")

Maximum Uncertainty

Maximum Entropy = $\log_2(n)$ where n is the number of possible states

Coin	$\log_2(2) = 1$ bit
DNA	$\log_2(4) = 2$ bits
Protein	$\log_2(20) = 4.32$ bits

Compression algorithms offer one approach to testing the randomicity of a DNA sequence.
A very random DNA sequence will require close to 2 bits per nucleotide to represent it, even when compressed. A sequence of DNA that has repeating patterns, or is otherwise highly structured, should be capable of being represented by less than 2 bits per nucleotide.

Compression of DNA

Compression Results with Standard Compression Algorithms

RK2 (60,099 bp)	62,555 bytes	Uncompressed (8 bit ASCII)
Arithmetic coding (.bin)	15,025 bytes	2.00 bits per nucleotide
Stuffit Deluxe (.sit)	15,195 bytes	2.02 bits per nucleotide
WinZip (.zip)	14,915 bytes	1.98 bits per nucleotide
UNIX Compress (.z)	17,667 bytes	2.35 bits per nucleotide

Compression Results with Specialized Biological Algorithms

Biocompress 2 (Loewenstern and Yianilos, DCC97)	1.62 -1.92 bits per nucleotide
Expectation Maximization (Allison, Edgoose and Dix, ISMB98)	1.61 - 1.91 bits per nucleotide
Approximate Repeat Model (Stern, Allison, Coppel and Dix, M&BP01)	0.61 - 1.59 bits per nucleotide

Standard compression algorithms are not particularly useful for such analysis, but specialized compression algorithms can be. The extent of the compression can yield information about how structured a sequence is, or identify the presence of even weak repeats. Eukaryotic sequences generally have more repeats than prokaryotic sequences, and are likely to demonstrate more compression.

Dot Matrix Analysis of DNA

Dot Matrix Analysis

A dot matrix analysis of a DNA sequence involves listing the sequence vertically and horizontally, either comparing it against itself or another DNA sequence, then placing a dot where a match occurs. With DNA sequences, identity results in a match. A stretch of matches appears as a long diagonal.

Filtering Results

A filter is used to filter out noise and focus on regions with extensive matches. The filter uses a sliding window of a given Window Size (**W**), placing a dot where a certain number of matches, called the Stringency (**S**), occur within that window.

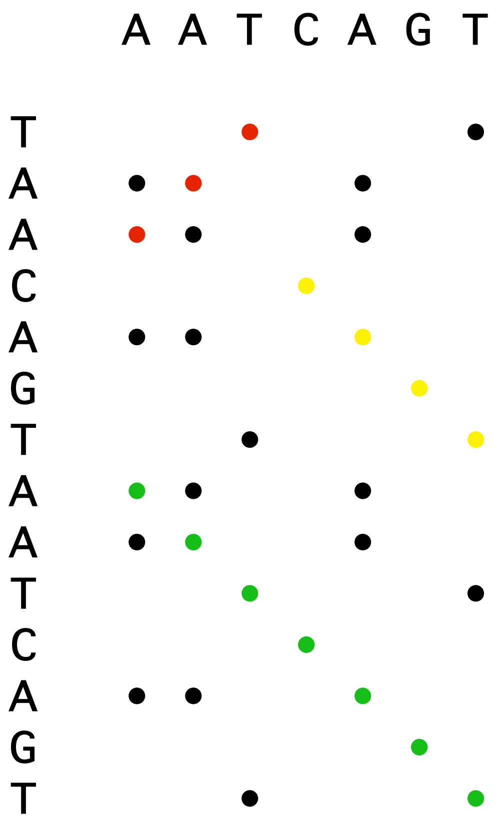
Direct and Inverted Repeats

Direct repeats appear as short parallel diagonals, while inverted repeats appear as short perpendicular diagonals.

Locating Repeats

If significant direct or inverted repeats are identified, they can be quickly located on the actual sequence using an appropriate sequence analysis program and function (i.e. the Seek Repeats and Seek Hairpins functions of DNA Strider).

Dot Matrix DNA Analysis Illustrated



A dot matrix shows all possible matches between two sequences with a dot placed at every match.

🟢 = Aligned sequence
🟡 = Direct repeat
🔴 = Inverted repeat

Dot Matrix Self DNA Analysis

Dot Matrix Self Analysis

Comparing a DNA sequence to itself in a dot matrix is known as a Dot Matrix Self DNA Analysis, and is a useful way to visualize the number of direct and inverted repeats present in the sequence. The sequence is listed both vertically and horizontally, and will appear as a diagonal line of dots.

Self Analysis Window Size and Stringency Values

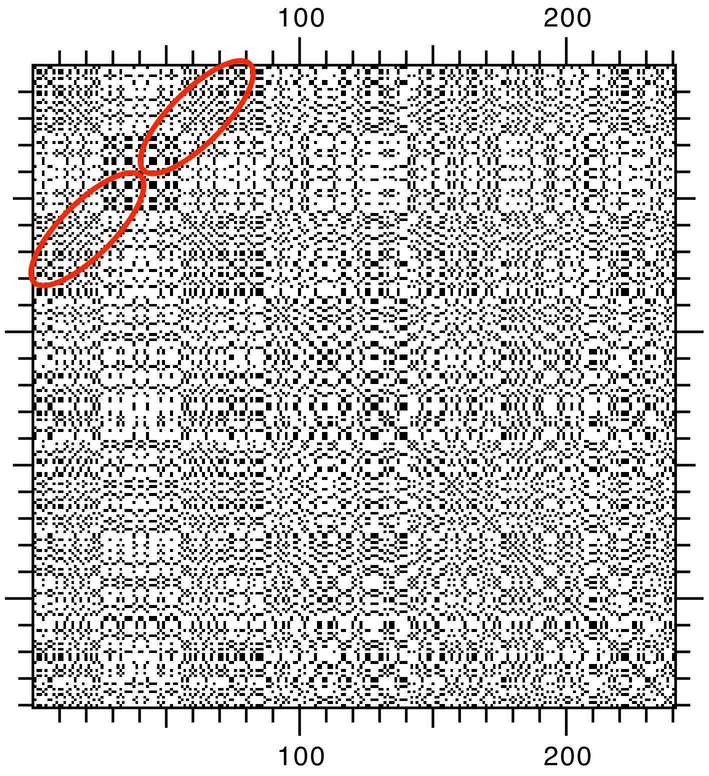
When comparing a sequence to itself, a good starting point might be comparing a Window Size of 1 and Stringency of 1 (**W1, S1**) to a Window Size of 23 with a Stringency of 7 (**W23, S7**). Trial and error should be used to compare windows of varying sizes and stringencies to the minimal values.

Direct and Inverted Repeats

Direct repeats appear as short diagonals parallel to the diagonal of the sequence itself, while inverted repeats appear as short diagonals perpendicular to the diagonal of the sequence itself.

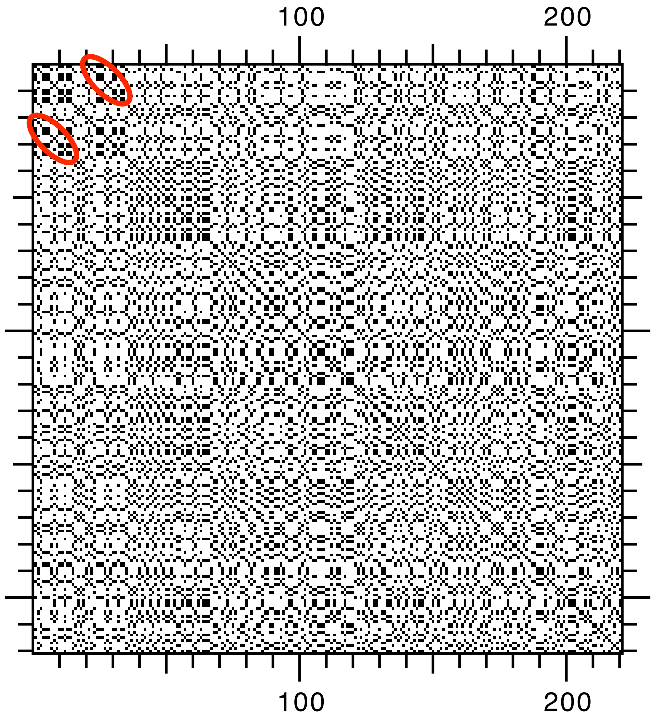
Dot Matrix DNA Self Analysis of a Synthetic Inverted Repeat

The 40 nt inverted repeat appears as short diagonals perpendicular to the diagonal of the sequence.

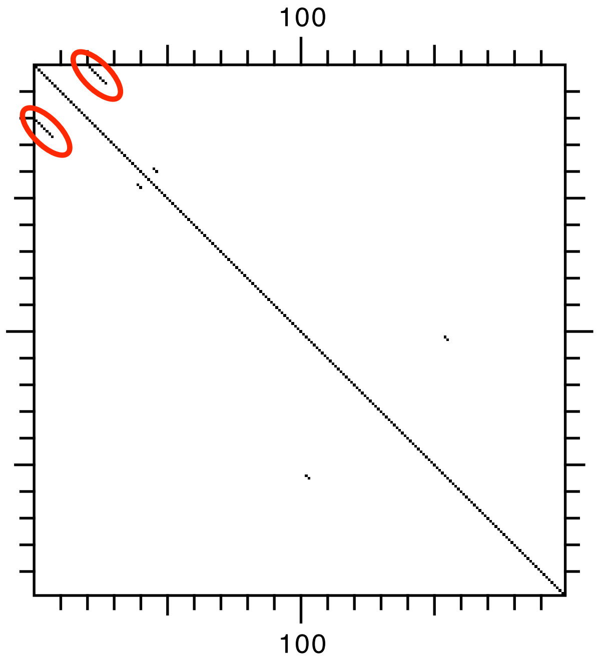


*This is a DNA Strider 1.3 DNA Self Matrix with Window Size 1 and Stringency 1 (**W1, S1**) of 200 nt of pBR322 sequence with a 40 nt inverted repeat added to the beginning of the sequence.*

Dot Matrix DNA Self Analysis of a Synthetic Direct Repeat



This is a DNA Strider 1.3 DNA Self Matrix with Window Size 1 and Stringency 1 (***W1, S1***) of 200 nt of pBR322 sequence with a 20 nt direct repeat added to the beginning of the sequence.



This is a DNA Strider 1.3 DNA Self Matrix with Window Size 23 and Stringency 15 (***W23, S15***) of 200 nt of pBR322 sequence with a 20 nt direct repeat added to the beginning of the sequence.

The 20 nt direct repeat appears as short diagonals parallel to the diagonal of the sequence.

Dot Matrix Pairwise DNA Analysis

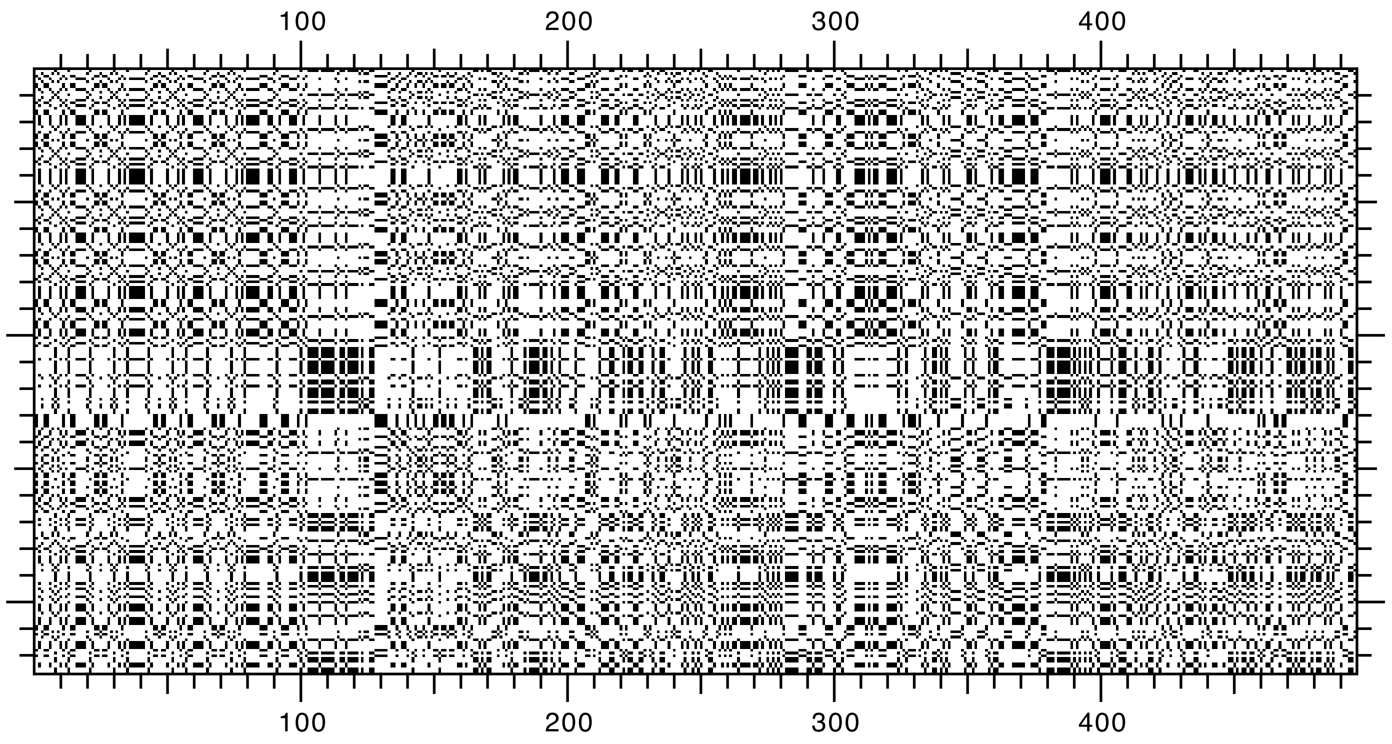
Dot Matrix Pairwise Analysis
A dot matrix analysis of two sequences functions by listing one sequence vertically and the other sequence horizontally, then placing a dot where a match occurs. A region of sequence identity will be revealed by a diagonal line of dots.

Pairwise Analysis Window Size and Stringency Values
For pairwise DNA comparisons, a good starting point might be a Window Size of 15 with a Stringency of 7 (**W15, S7**). In general, long windows with medium or high stringencies should be used (**W15, S7**; **W23, S15**; **W15, S10-11**; **W11, S7** or **W7, S4**).

Orientation
Unless you are certain of the orientation, make sure to run one sequence antiparallel or run a DNA Antiparallel Matrix as well.

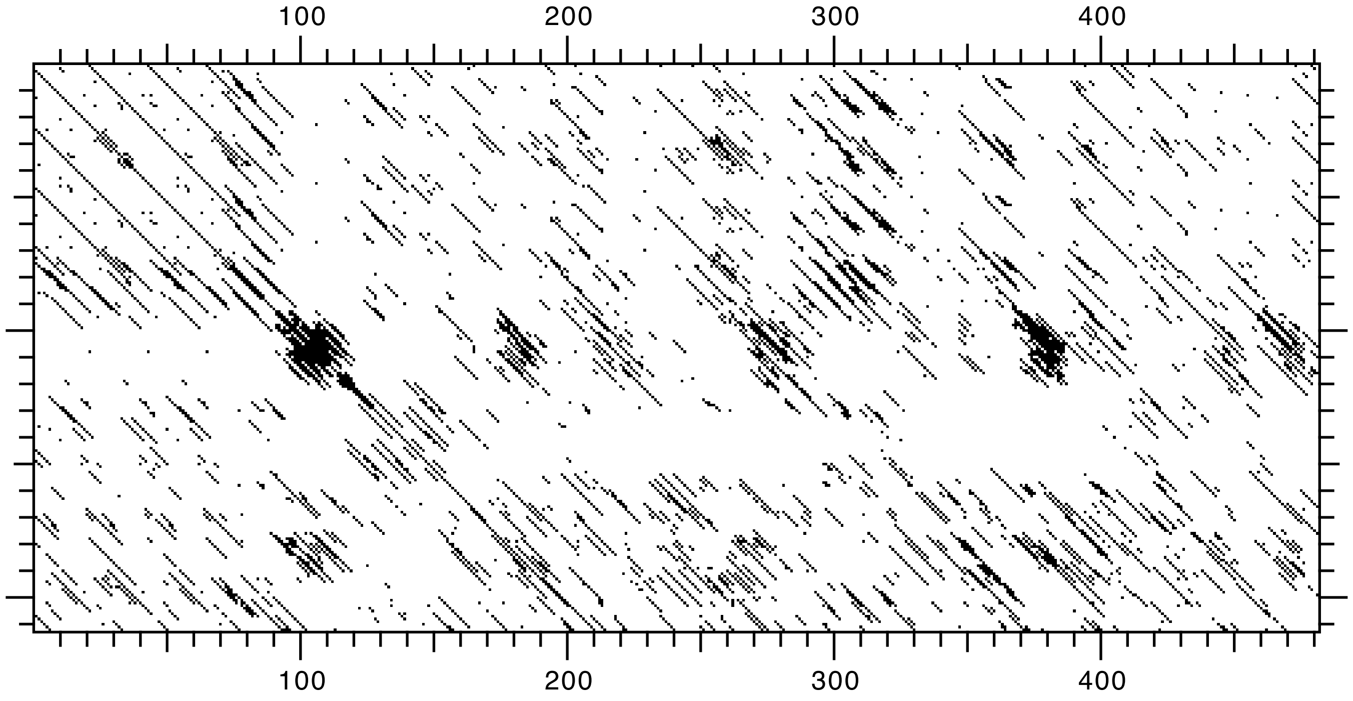
Dot Matrix Analysis and Alignment Algorithms
Since a dot matrix identifies all possible matches between two sequences, it is ideal for making a first pass, then comparing the matches to the results of a sequence alignment algorithm such as BLAST, Smith-Waterman or Needleman-Wunsch to make sure all the matches are accounted for.

Dot Matrix Pairwise Analysis



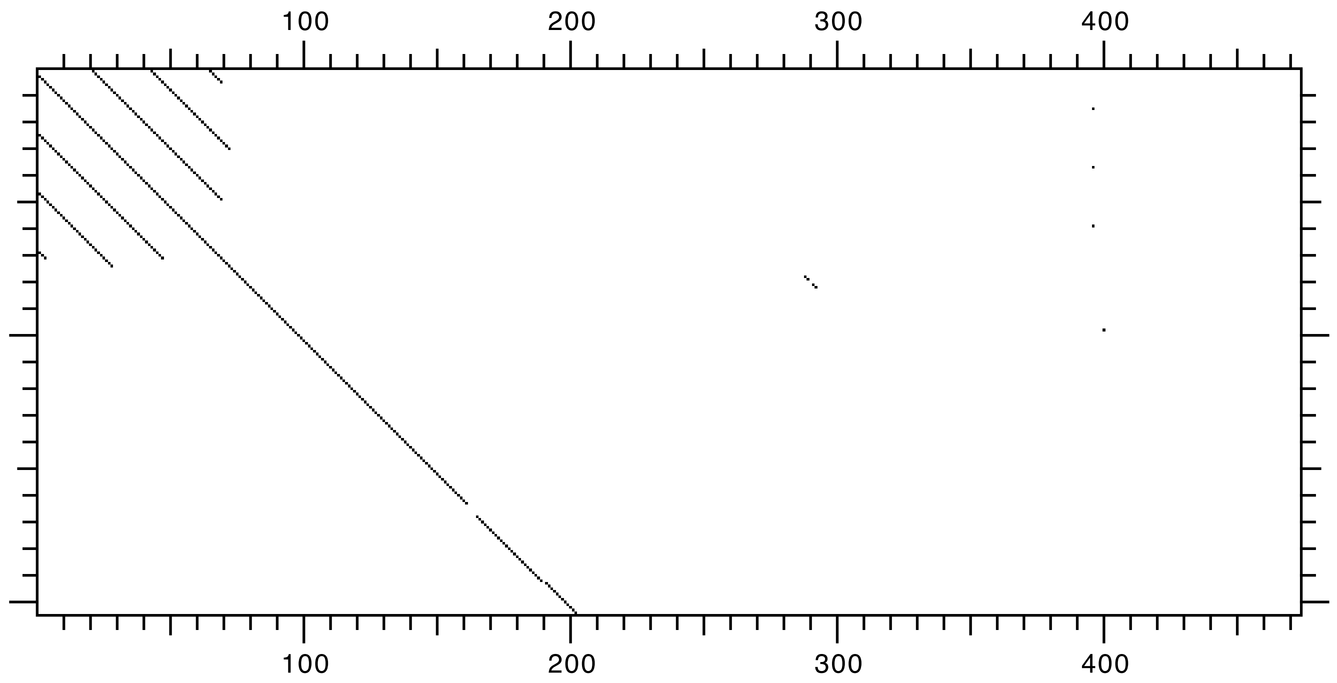
*This is a DNA Strider 1.3 DNA Matrix with Window Size 1 and Stringency 1 (**W1, S1**) of two plasmid oriV regions (RSF1010 and R1162 (antiparallel)).*

Dot Matrix Pairwise Analysis



*This is a DNA Strider 1.3 DNA Matrix with Window Size 15 and Stringency 7 (**W15, S7**) of two plasmid oriV regions (RSF1010 and R1162 (antiparallel)).*

Dot Matrix Pairwise Analysis



*This is a DNA Strider 1.3 DNA Matrix with Window Size 23 and Stringency 15 (**W23, S15**) of two plasmid oriV regions (RSF1010 and R1162 (antiparallel)).*

Dynamic Programming and Optimal DNA Sequence Alignment

Dynamic Programming
Solves a problem by breaking the problem into smaller subproblems, which are separately solved, then sequentially reassembled to solve the entire problem. The solution to each subproblem is stored in a table along with a score, and the final answer is arrived at by choosing the sequence of solutions that yields the highest score.

Dynamic programming was invented in 1950 by Richard Bellman at Princeton University, and works well when many solutions are possible but an optimal solution must be found.

Dynamic Programming in Sequence Alignment
This approach was first applied to solving biological sequence alignment problems by Saul Needleman and Christian Wunsch in 1970. It can be used to optimally solve either a global alignment problem (Needleman-Wunsch) or a local alignment problem (Smith-Waterman).

Dynamic Programming Steps in DNA Sequence Comparison

How Dynamic Programming Compares Sequences

1. Create matrix and fill with best scores
There are only three possible choices at each position of the matrix: (a) match the residues present; (b) insert a gap in the top sequence; or (c) insert a gap in the side sequence. The exact score depends on the substitution, gap creation and gap insertion values chosen. The best score of each choice is selected, and a pointer to the preceding position used to arrive at the score is stored with the score.

2. Find highest score
For global alignment (Needleman-Wunsch), the highest score in the final row and final column is used. For local alignment (Smith-Waterman), the highest score anywhere in the matrix is used.

3. Trace pointers back to start and generate alignment
The sequence alignment is created in reverse order, by tracing the pointer back from the highest score to the previous highest score, until either the very start (global) or when it reaches a starting score of 0 (local).

Dynamic Programming Illustrated

		G	C	T	G	G	A	A	G	G	C	A	T
	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	0	0	5	5	0	0	5	5	0	0	0
C	0	0	10	3	0	1	1	0	0	1	10	3	0
A	0	0	3	6	0	0	0	6	.	0	3	15	8
G	0	5	0	0	11	5	0	2	11	5	0	8	11
A	0	0	1	0	4	7	10	5	4	7	1	5	4
G	0	5	0	0	5	9	3	6	10	9	3	0	1
C	0	0	10	3	0	2	5	0	3	6	14	7	0
A	0	0	3	6	0	0	7	10	3	0	7	19	12
C	0	0	5	0	2	0	0	3	6	0	5	12	15
T	0	0	0	10	3	0	0	0	0	2	0	5	17

DYNAMIC PROGRAMMING STEPS

1. Create matrix and fill with best scores (keeping pointers)
2. Find highest score (global in final row and column; local anywhere)
3. Trace pointers back to start and generate sequence alignment

Scores are from a local dynamic programming example in Gibas & Jambeck

```
G C T G G A A G G C A . T
| |   |     |   | | |   |
G C A G . . A . G C A C T
```

GLOBAL

Needleman-Wunsch (GCG Gap)

Starts at final row and column in lower right (17), retraces path

```
G A A G . G C A
|   | |   | | |
G C A G A G C A
```

LOCAL

Smith-Waterman (GCG BestFit)

Starts at highest score in matrix (19), retraces path

Needleman-Wunsch Optimal Global Alignment

RSF 1010 oriV (495 bp) x R1162 oriV (226 bp)

Standard, Needleman-Wunsch ->, Mismatch Smaller (1), Gap Medium (2)

```
               •        20         •        40         •        60         •        80         •       100         •       120
  1 --actaactgtcacgaacccctgcaataactgtcacgcccccctgcaataactgtcacgaacccctgcaataactgtcacgcccccaaacctgcaaacccagcaggggcgggggctggcggggtg 123
      |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  1 gtactaactgtcacgaacccctgcaataactgtcacgcccccctgcaataactgtcacgaacccctgcaataactgtcacgcccccaaacctgcaaacccagcaggggcgggggctggcggggtg 125
             •        20         •        40         •        60         •        80         •       100         •       120
          •       140         •       160         •       180         •       200         •       220         •       240
124 ttggaaaaatccatccatgattatctaagaataatccactaggcgcggtta-tcacgccttgtggggcggtgctgcccttgcccaatatgcccggccagaggccggatagctggtctattcgctg 247
    ||||||||||||||||||||||||||||||||||||||||||||||||||| ||||||| ||||||||| ||||||||| |||||||||||||    |||||||||
126 ttggaaaaatccatccatgattatctaagaataatccactaggcgcggttagtcacgccctgtggggcgctgctgccct=gcccaatatgccc====agaggccgg=================== 226
        •       140         •       160         •       180         •       200          •           220
      •       260         •       280         •       300         •       320         •       340         •       360         •
248 cgctaggctacacaccgccccaccgctgcgcgcagggggaaaggcgggcaaagccgtaaaccccacaccaaaccccgcagaaatacgctggagcgcttttagccgctttagcggcctttccccct 372

    =============================================================================================================================

         380         •       400         •       420         •       440         •       460         •       480         •
373 acccgaagggtgggggcgcgtgtgcagccccgcagggcctgtctcggtcgatcattcagcccggctcatccttctggcgtggcggcagaccgaacaaggcgcggtcgtggtcgcgttcaaggt   495

    ===========================================================================================================================
```

```
% Identity =  44.4 (221/498)
```

Lecture 4
Introduction to
DNA Analysis
October 15, 2009

Heuristic Sequence Comparison

Heuristic Algorithms
Solve a problem by using rules of thumb to reach a solution. The solution is not guaranteed to be an optimal solution, but is generally arrived at far faster than using an optimal solution approach such as dynamic programming.

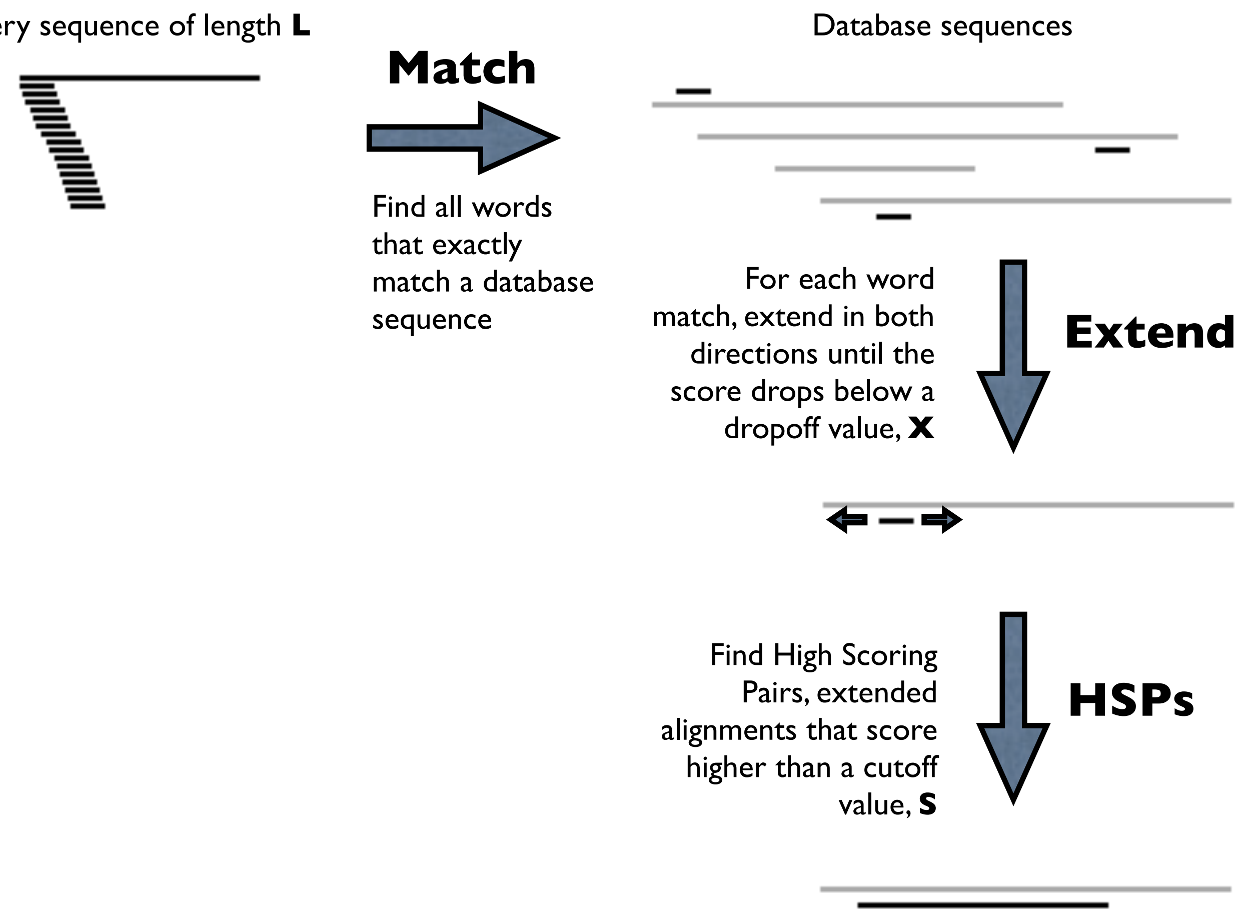
Heuristic Sequence Comparison Algorithms
In sequence comparison, commonly used heuristic approaches include the BLAT algorithm, developed by Jim Kent in 2002, the BLAST algorithm, developed by Stephen Altschul in 1990, and the FASTA algorithm, developed by William Pearson in 1988. The best known of these is the BLAST algorithm.

BLAST Heuristic Nucleotide Comparison Algorithm

How BLAST Works in Nucleotide Sequences

1. Make a list of 11 letter words (default DNA word length (W) = 11) in the DNA sequence. In a query sequence of length L, the maximum number of words will be L - W + 1 (L - 10 for the default word length of 11).
2. Search with each word for exact matches to words in a pre-indexed database of sequences.
3. For each match found, keep extending the alignment in each direction along the sequence, allowing small gaps, until the score drops below a dropoff value (X), to create high scoring segment pairs (HSPs).
4. Select HSPs that score above the HSP cutoff score (S).
5. Determine the statistical significance of each selected HSP score.
6. Use the Smith-Waterman algorithm to generate a local sequence alignment for each selected HSP.

Nucleotide BLAST Algorithm Illustrated



Generate a maximum of **L - W + 1** words of length **W**

Find all words that exactly match a database sequence

For each word match, extend in both directions until the score drops below a dropoff value, **X**

Find High Scoring Pairs, extended alignments that score higher than a cutoff value, **S**

BLAST Nucleotide Query Programs

blastn
Compares a nucleotide query sequence against a nucleotide sequence database. It is best used to find closely related DNA sequences (those with over 70% identity) and analyze highly conserved coding or non-coding DNA regions. It is not well suited to doing other kinds of comparisons.

blastx
Compares the six-frame translation of a nucleotide query sequence against a protein sequence database. It is best used to find potential translation products of an unknown nucleotide sequence. It works well if you are unsure of the quality of your sequence data, as it can identify a coding region despite sequencing errors such as frameshifts.

tblastx
Compares the six-frame translation of a nucleotide query sequence against the six-frame translation of a nucleotide sequence database. It is useful for discovering new proteins and ESTs, but is considerably more computationally intensive.

Search for short, nearly exact matches
This choice automatically sets the BLAST variables in blastn to better locate short, nearly exact DNA sequence matches (no filter, word size 7, expect 1000), and is best used when working with short query nucleotide sequences or looking for short, nearly exact matches.

megablast
This uses a variant of the BLAST algorithm called Mega BLAST which is optimized for quickly comparing nearly identical nucleotide sequences (over 80% identity). It is fastest with larger word sizes (16 and up). The discontiguous version of Mega BLAST is optimized to quickly compare more divergent nucleotide sequences (80% and below identity) and should give better results than Mega BLAST or BLAST for sequence comparisons involving distantly related organisms.

BLAST Variables

Database
You will generally be running searches against the nr (nonredundant) database. You may need to choose another database, or limit a search by an Entrez query (e.g. Escherichia coli [ORGN]).

Filter
Unless you are working with a query sequence that contains repetitive residues or has a biased composition, it is best to turn filtering off, particularly when working with shorter query sequences. Masking of segments in the query sequence is indicated by X's, resulting in "**XXXXX...**" marked regions.

The **Low complexity** filter uses the DUST program to filter low complexity regions from nucleic acid sequences.

The **Human repeats** filter masks LINE's and SINE's, and may be particularly useful when working with nucleotide sequences containing such elements.

The **Mask lower case** feature allows you to manually select which residues to mask by making them lower case, e.g. transmembrane or coiled-coil regions.

The **Mask for lookup table only** feature uses selected filters only for the initial match, but not the subsequent extension.

Expect (E Value)
The default expectation value is **10**, which means that you can expect up to 10 matches by chance alone. You can decrease the value to get only more significant results, or increase it to **1000 or more** when searching with short queries, which are more likely to be found by chance in a given database.

Word Size
The default value is **11**. Decreasing the word size to **7** may result in a more sensitive search and find shorter regions of homology, but will increase the search time. Decreasing the word size is particularly helpful with shorter query sequences. Increasing the word size to **15** will find longer regions of homology, and will decrease the search time.

BLAST Results Illustrated

```
                                                                       Score     E
Sequences producing significant alignments:                            (bits)   Value
gi|152576|gb|M21475.1|RSFORIVA Plasmid RSF1010 replication ...  981      0.0 gi|17864894|gb|AF403427.1|
AF403427 Cloning vector pRL1342, ...  906      0.0 gi|17864893|gb|AF403426.1|AF403426 Cloning vector
pRL1383a,...  906      0.0 gi|4323404|gb|AF100178.1|AF100178 Cloning vector pVZ361 com...  906      0.0
gi|4323393|gb|AF100177.1|AF100177 Cloning vector pVZ324 com...  906      0.0 gi|4323382|gb|AF100176.1|
AF100176 Cloning vector pVZ321 com...  906      0.0 gi|4323371|gb|AF100175.1|AF100175 Cloning vector
pVZ322 com...  906      0.0
.
.
gi|9967611|emb|AJ293027.1|UEU293027 Uncultured eubacterium ...  607      e-171 gi|151790|gb|M13380.1|
R11ORI Plasmid R1162 DNA between coor...  498      e-138
.
.
                               Alignments

>gi|152576|gb|M21475.1|RSFORIVA Plasmid RSF1010 replication origin (oriV) region
          Length = 495

 Score =  981 bits (495), Expect = 0.0
 Identities = 495/495 (100%)
 Strand = Plus / Plus

Query: 1   actaactgtcacgaacccctgcaataactgtcacgccccccctgcaataactgtcacgaac 60
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 1   actaactgtcacgaacccctgcaataactgtcacgccccccctgcaataactgtcacgaac 60
.
.
 >gi|9967611|emb|AJ293027.1|UEU293027 Uncultured eubacterium pIE1115 plasmid pIE1115, complete sequence
          Length = 10687

 Score =  607 bits (306), Expect = e-171
 Identities = 395/416 (94%), Gaps = 9/416 (2%)
 Strand = Plus / Plus

Query: 1    actaactgtcacgaacccctgcaataactgtcacgccccccctgcaataactgtcacgaac 60
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 7136 actaactgtcacgaacccctgcaataactgtcacgccccccctgcaataactgtcacgaac 7195
.
.
```

BLAST Results

Format
You may often want to increase the number of descriptions and alignments (up to 1,000 of each). You can also limit your results by a particular Entrez query, e.g. "Escherichia coli [ORGN]", or an expectation value range.

Score (bits)
A statistical measure of the significance of the alignment, measured in bits. The higher the score, the more likely for the alignment to be significant. ***A score of* 50 bits or above *is likely to be significant.***

E Value
The expectation value. An estimate of the number of times the match may have occurred by chance. The lower the value, the more likely for the alignment to be significant. ***An E value of* 0.0001 (1e-4) or below *is likely to be significant.***

Percent Identity
The fraction of identical residues in the final alignment. The higher the identity, the more likely for the alignment to be significant. ***A* 70% or greater *identity for nucleotide sequences is likely to be significant for longer regions of alignment* (100 nucleotides and up).**

Length
The length of the final alignment. The longer the length, the more likely for the alignment to be significant. Very short alignments (e.g. 10 nucleotides) may have a very high percent identity or very low E value, yet not be significant. ***Longer alignments (i.e.* 100 nucleotides and above) *with high scores, high identities or low E values are likely to be significant.***

*Remember that BLAST hits are not transitive, unless the alignments overlap, since BLAST alignment is fundamentally a local alignment. Thus, if query **A** results in significant hits to **B** and **C**, **B** and **C** are not necessarily similar to each other (**A** might contain domain **1** and **2**, **B** might contain only domain **1**, and **C** might contain only domain **2**). It is thus wise to carefully check the length and extent of any alignment.*

BLAT Heuristic Algorithm

BLAT is short for "**B**LAST-**l**ike **a**lignment **t**ool," and like BLAST, BLAT rapidly scans for relatively short matches (hits) in DNA or protein sequences, and extends them into high-scoring pairs (HSPs). However, BLAT differs from BLAST in some significant ways:

- Where BLAST builds an index of the query sequence and then scans linearly through the database, BLAT builds an index of non-overlapping words of length K in the database, stores the index in RAM, then scans linearly through the query sequence.
- Storing the word index in RAM is memory intensive, typically requiring 1 to 2 gigabytes of RAM for a single eukaryotic genome, but results in far more rapid searches. BLAT searches can be hundreds of times faster than BLAST searches (which are already fast).
- BLAT can trigger extensions on any number of perfect or near-perfect hits, unlike BLAST, and is better at stitching areas of homology into a larger alignment.
- BLAT has special code to handle introns in DNA/RNA alignments and aligns splice sites more accurately than BLAST.
- BLAT is optimized for finding matches with a high degree of identity. When run with its default values it will generally find anything with 90% or greater identity at the DNA level, and 70% or greater identity at the protein level. For more sensitive searches, PSI-BLAST (for proteins) or optimal dynamic programming algorithms such as Smith-Waterman (local) or Needleman-Wunsch (global) are better options.

References

BLAST

http://blast.ncbi.nlm.nih.gov/

http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=FAQ

BLAST: An Essential Guide to the BASIC Local Alignment Search Tool by Ian Korf, Mark Yandell & Joseph Bedell

BLAT

http://hgwdev.cse.ucsc.edu/cgi-bin/hgBlat?command=start
http://www.kentinformatics.com/

BLAT Documentation

http://genome.ucsc.edu/goldenPath/help/blatSpec.html