BLAST

J. Mol. Biol. (1990) 215, 403–410

Basic Local Alignment Search Tool

Stephen F. Altschul[1], Warren Gish[1], Webb Miller[2]
Eugene W. Myers[3] and David J. Lipman[1]

[1]*National Center for Biotechnology Information*
National Library of Medicine, National Institutes of Health
Bethesda, MD 20894, U.S.A.

[2]*Department of Computer Science*
The Pennsylvania State University, University Park, PA 16802, U.S.A.

[3]*Department of Computer Science*
University of Arizona, Tucson, AZ 85721, U.S.A.

(Received 26 February 1990; accepted 15 May 1990)

A new approach to rapid sequence comparison, basic local alignment search tool (BLAST), directly approximates alignments that optimize a measure of local similarity, the maximal segment pair (MSP) score. Recent mathematical results on the stochastic properties of MSP scores allow an analysis of the performance of this method as well as the statistical significance of alignments it generates. The basic algorithm is simple and robust; it can be implemented in a number of ways and applied in a variety of contexts including straightforward DNA and protein sequence database searches, motif searches, gene identification searches, and in the analysis of multiple regions of similarity in long DNA sequences. In addition to its flexibility and tractability to mathematical analysis, BLAST is an order of magnitude faster than existing sequence comparison tools of comparable sensitivity.

1. Introduction

The discovery of sequence homology to a known protein or family of proteins often provides the first clues about the function of a newly sequenced gene. As the DNA and amino acid sequence databases continue to grow in size they become increasingly useful in the analysis of newly sequenced genes and proteins because of the greater chance of finding such homologies. There are a number of software tools for searching sequence databases but all use some measure of similarity between sequences to distinguish biologically significant relationships from chance similarities. Perhaps the best studied measures are those used in conjunction with variations of the dynamic programming algorithm (Needleman & Wunsch, 1970; Sellers, 1974; Sankoff & Kruskal, 1983; Waterman, 1984). These methods assign scores to insertions, deletions and replacements, and compute an alignment of two sequences that corresponds to the least costly set of such mutations. Such an alignment may be thought of as minimizing the evolutionary distance or maximizing the similarity between the two sequences compared. In either case, the cost of this alignment is a measure of similarity; the algorithm guarantees it is optimal, based on the given scores. Because of their computational requirements, dynamic programming algorithms are impractical for searching large databases without the use of a supercomputer (Gotoh & Tagashira, 1986) or other special purpose hardware (Coulson *et al.*, 1987).

Rapid heuristic algorithms that attempt to approximate the above methods have been developed (Waterman, 1984), allowing large databases to be searched on commonly available computers. In many heuristic methods the measure of similarity is not explicitly defined as a minimal cost set of mutations, but instead is implicit in the algorithm itself. For example, the FASTP program (Lipman & Pearson, 1985; Pearson & Lipman, 1988) first finds locally similar regions between two sequences based on identities but not gaps, and then rescores these regions using a measure of similarity between residues, such as a PAM matrix (Dayhoff *et al.*, 1978) which allows conservative replacements as well as identities to increment the similarity score. Despite their rather indirect approximation of minimal evolution measures, heuristic tools such as FASTP have been quite popular and have identified many distant but biologically significant relationships.

0022-2836/90/190403-08 $03.00/0

Genomics and Bioinformatics

Chapter 9

One of the most highly cited papers in the history of science

Rank 1 (most cited paper) in the 1990s

It is a paper on a program for searching in molecular databases

J. Mol. Biol. (1990) 215, 403–410

Basic Local Alignment Search Tool

Stephen F. Altschul[1], Warren Gish[1], Webb Miller[2]
Eugene W. Myers[3] and David J. Lipman[1]

[1]*National Center for Biotechnology Information*
National Library of Medicine, National Institutes of Health
Bethesda, MD 20894, U.S.A.

[2]*Department of Computer Science*
The Pennsylvania State University, University Park, PA 16802, U.S.A.

[3]*Department of Computer Science*
University of Arizona, Tucson, AZ 85721, U.S.A.

(Received 26 February 1990; accepted 15 May 1990)

A new approach to rapid sequence comparison, basic local alignment search tool (BLAST), directly approximates alignments that optimize a measure of local similarity, the maximal segment pair (MSP) score. Recent mathematical results on the stochastic properties of MSP scores allow an analysis of the performance of this method as well as the statistical significance of alignments it generates. The basic algorithm is simple and robust; it can be implemented in a number of ways and applied in a variety of contexts including straightforward DNA and protein sequence database searches, motif searches, gene identification searches, and in the analysis of multiple regions of similarity in long DNA sequences. In addition to its flexibility and tractability to mathematical analysis, BLAST is an order of magnitude faster than existing sequence comparison tools of comparable sensitivity.

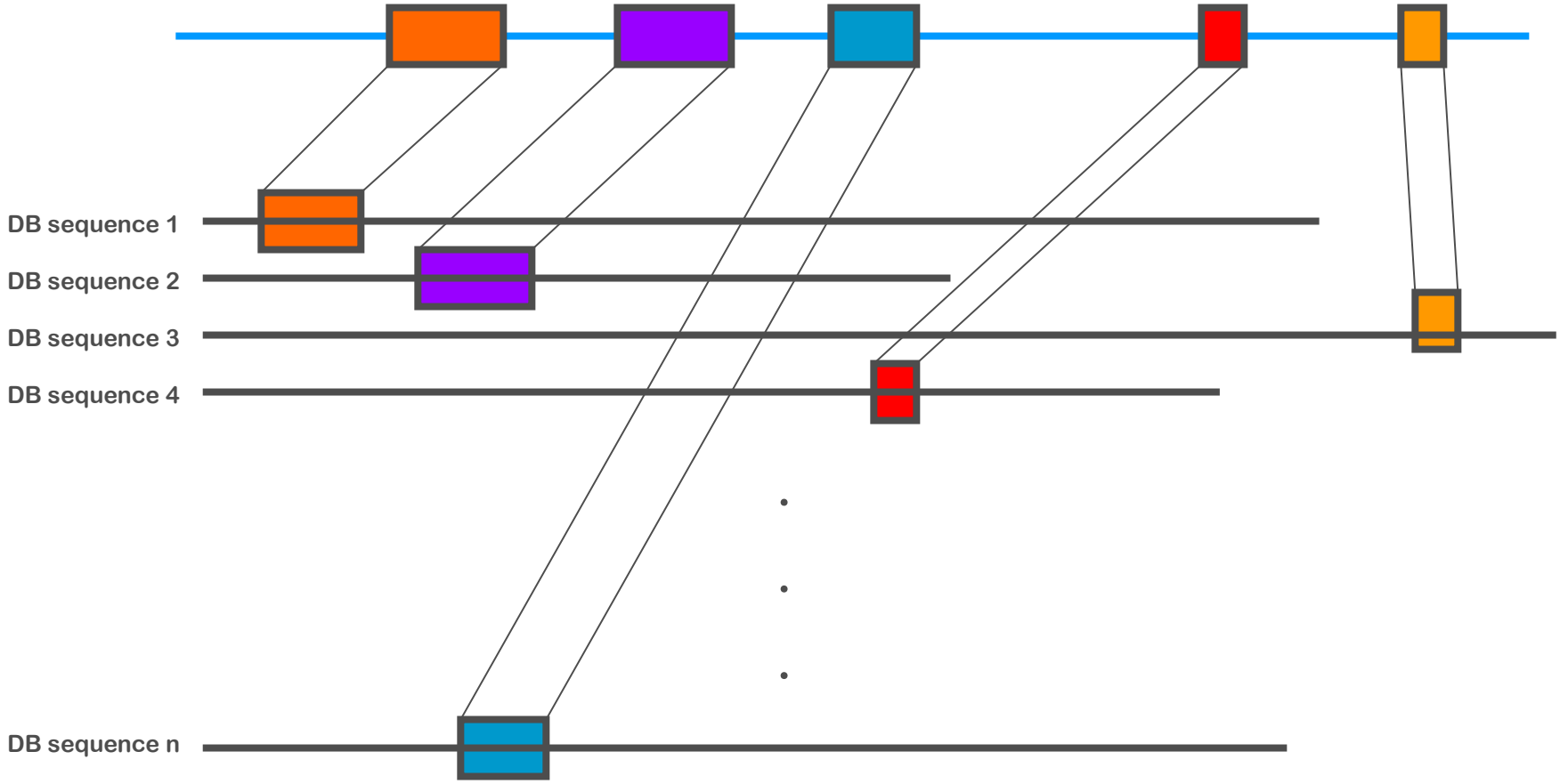
1. Introduction

The discovery of sequence homology to a known protein or family of proteins often provides the first clues about the function of a newly sequenced gene. As the DNA and amino acid sequence databases continue to grow in size they become increasingly useful in the analysis of newly sequenced genes and proteins because of the greater chance of finding such homologies. There are a number of software tools for searching sequence databases but all use some measure of similarity between sequences to distinguish biologically significant relationships from chance similarities. Perhaps the best studied measures are those used in conjunction with variations of the dynamic programming algorithm (Needleman & Wunsch, 1970; Sellers, 1974; Sankoff & Kruskal, 1983; Waterman, 1984). These methods assign scores to insertions, deletions and replacements, and compute an alignment of two sequences that corresponds to the least costly set of such mutations. Such an alignment may be thought of as minimizing the evolutionary distance or maximizing the similarity between the two sequences compared. In either case, the cost of this alignment is a measure of similarity; the algorithm guarantees it is optimal, based on the given scores. Because of their computational requirements, dynamic programming algorithms are impractical for searching large databases without the use of a supercomputer (Gotoh & Tagashira, 1986) or other special purpose hardware (Coulson *et al.*, 1987).

Rapid heuristic algorithms that attempt to approximate the above methods have been developed (Waterman, 1984), allowing large databases to be searched on commonly available computers. In many heuristic methods the measure of similarity is not explicitly defined as a minimal cost set of mutations, but instead is implicit in the algorithm itself. For example, the FASTP program (Lipman & Pearson, 1985; Pearson & Lipman, 1988) first finds locally similar regions between two sequences based on identities but not gaps, and then rescores these regions using a measure of similarity between residues, such as a PAM matrix (Dayhoff *et al.*, 1978) which allows conservative replacements as well as identities to increment the similarity score. Despite their rather indirect approximation of minimal evolution measures, heuristic tools such as FASTP have been quite popular and have identified many distant but biologically significant relationships.

0022-2836/90/190403-08 $03.00/0

BLAST is a database search program that detects local sequence similarity in database sequences to a query sequence

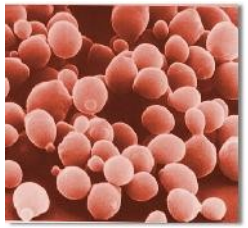


BLAST can be used to generate a hypothesis on the function of a newly sequenced gene

your sequence

81% sequence identity

yeast DNA ligase

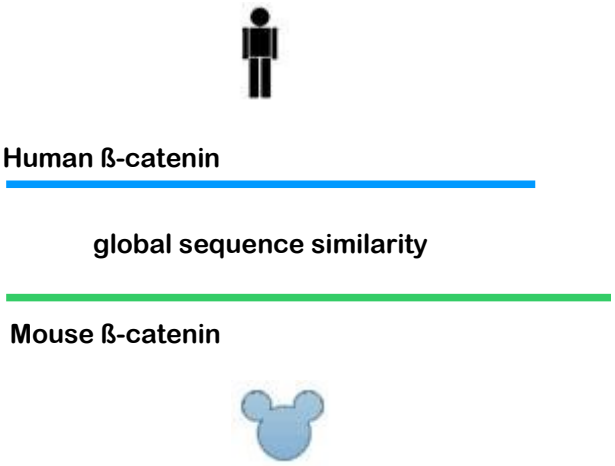


Your sequence has a good chance to be a DNA ligase too

Translate the gene into a protein sequence and search against a protein database

If you find a protein from a different species with a strong global similarity to your gene whose function is known in this species, chances are good that your gene has the same or a similar function in your species

BLAST can be used to transfer experimental analysis of gene function from humans to model organisms



From the medical perspective we are most interested in deciphering the function of human genes

Functional analysis typically requires experiments that interfere with an organism

These are not ethical to do in humans

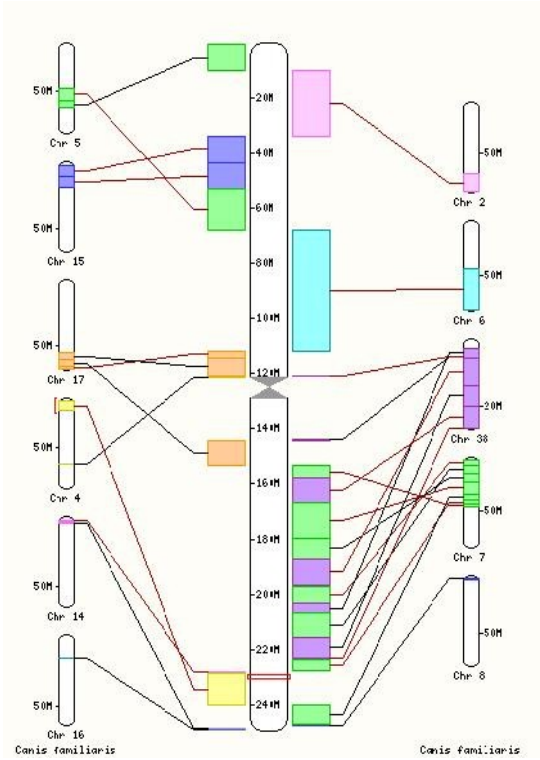
Study the orthologous gene in a model organism (mouse, fly, worm)

BLAST can be used to annotate a newly sequenced genome

BLAST all predicted genes in the newly sequenced genome to the genome of a related species

Transfer functional annotations between the genomes

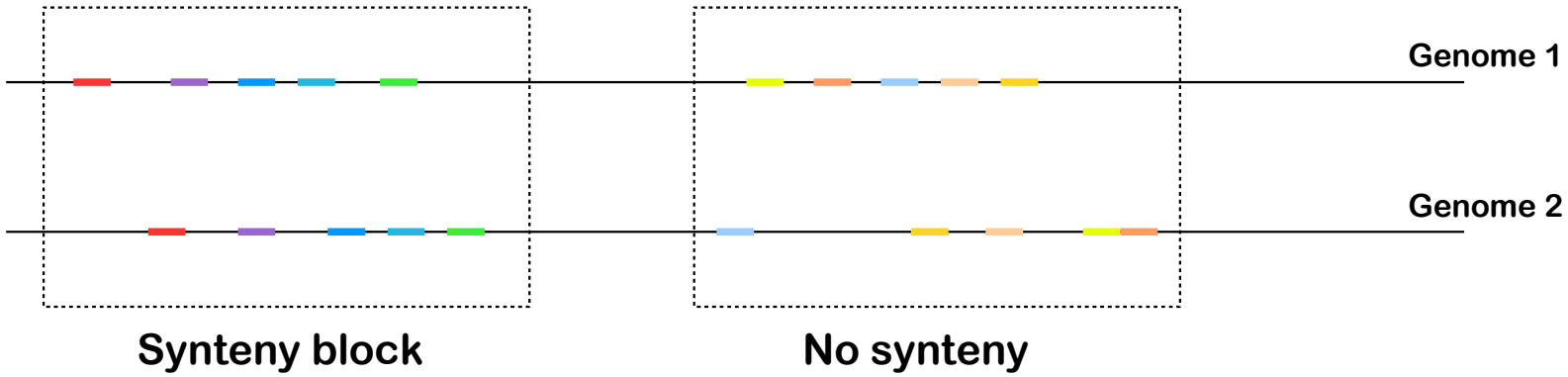
BLAST can be used to study the evolution of genome organization



Related species store the same genes but on different places on their genome

Compare all genes of genome A to all genes of genome B and match up

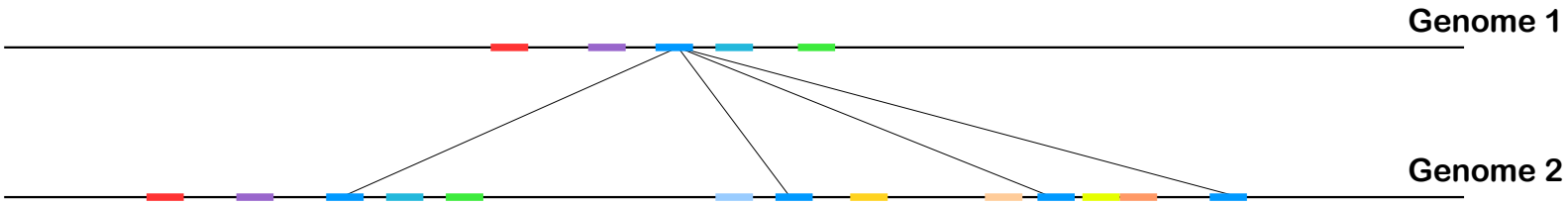
BLAST can be used to study synteny



Stretches of genes follow in the same order in both genomes (synteny)

BLAST can match up the genes, the actual detection of the synteny block requires a different algorithm

Synteny helps finding pairs of orthologous sequences



We search genome 2 for sequences that are homologous to the blue gene from genome 1 and find 4 clear hits

Only one is in a conserved synteny block. That's the orthologous one

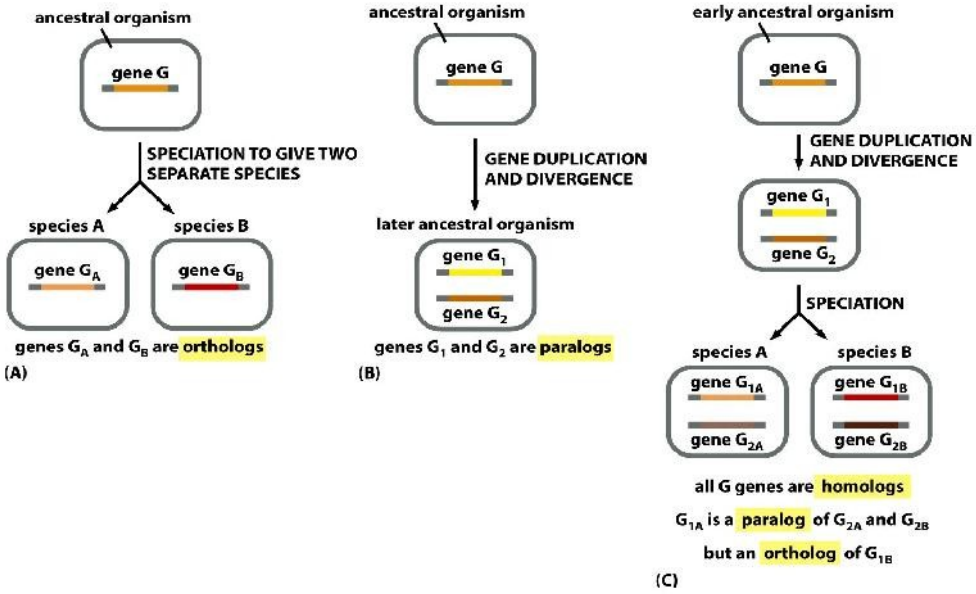
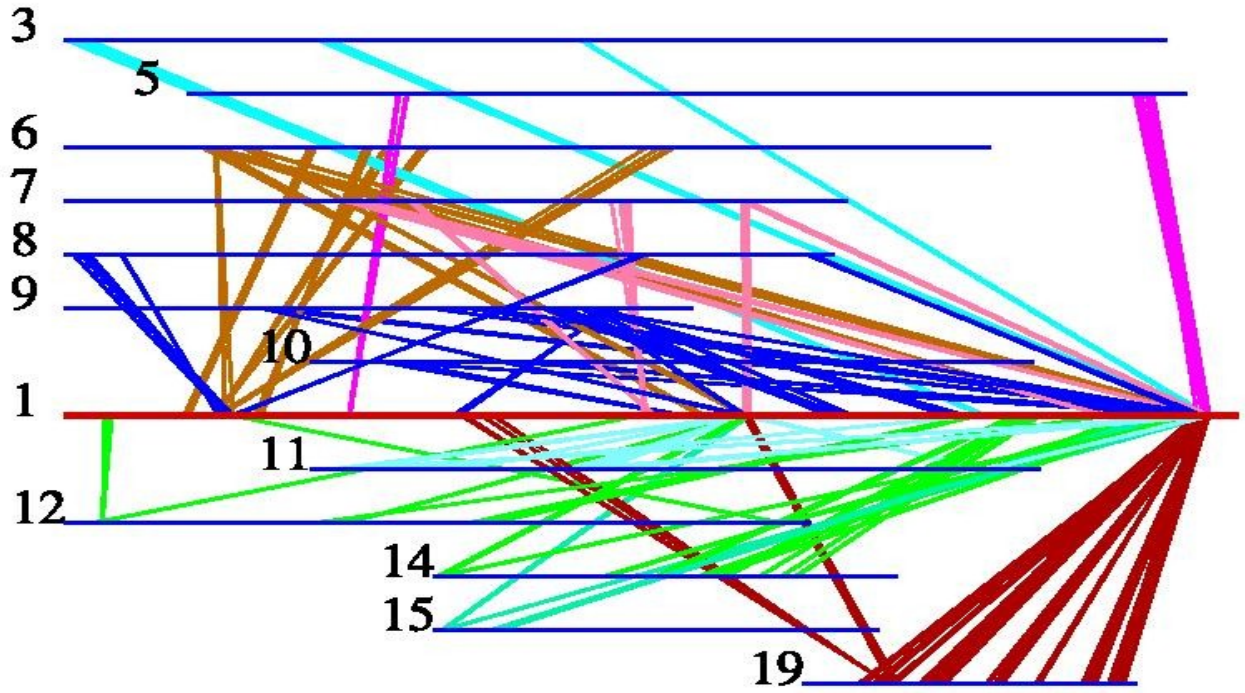


Figure 1-25 *Molecular Biology of the Cell*, Fifth Edition (© Garland Science 2008)

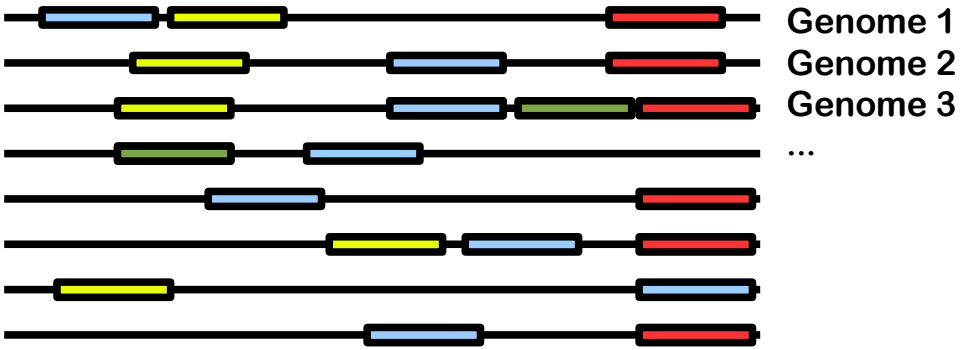
BLAST can be used to study duplication events within a genome



Code that is used multiple times

Search all genes in a genome against that genome. Ignore the first hit in the hit list (that's the query gene itself)

BLAST can be used to identify genes that are essential for life

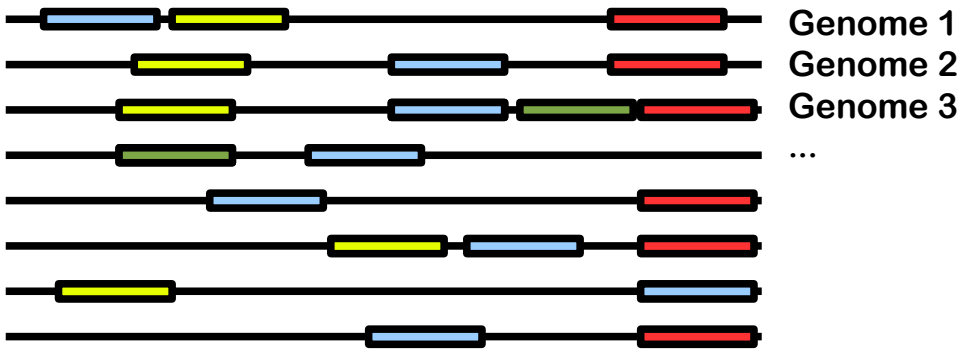


Only the blue gene is essential

Compare many genomes and search for genes that have a homologue in all of them

polymerases, ribosomes, ligases, …

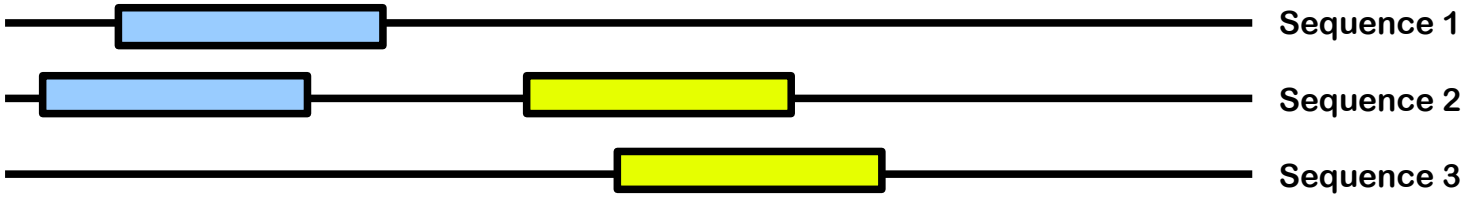
BLAST can be used to detect genes involved in a certain physiological function



Assume that all aerobic organisms have a version of the red gene and anaerobic bacteria do not

What kind of function will the red gene have?

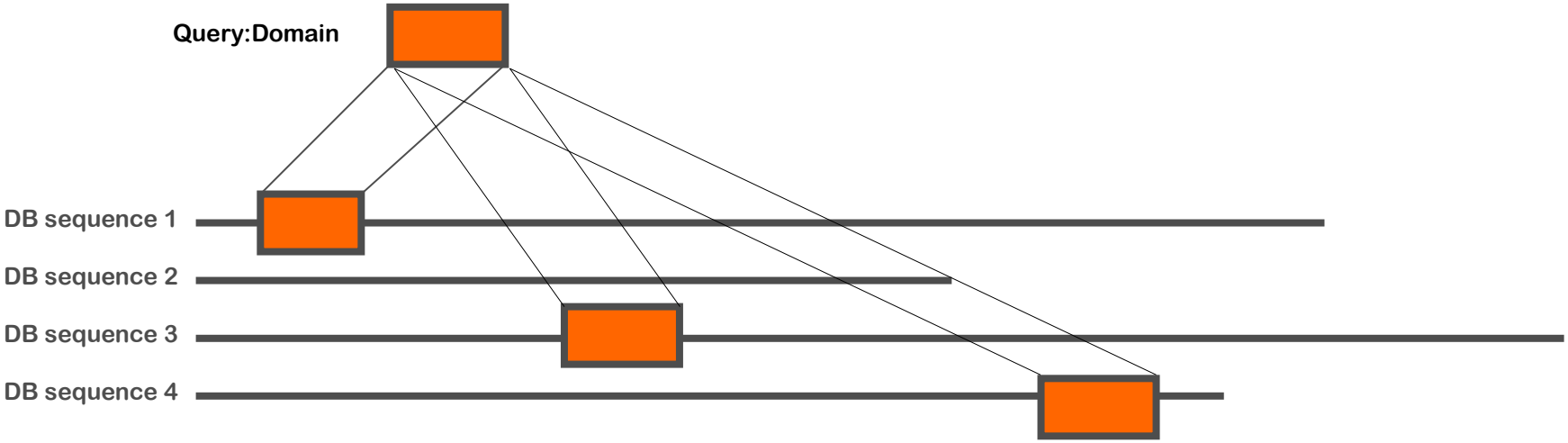
Caution: Local sequence similarity is not transitive



Sequences 1 and 2 are locally homologous
Sequences 2 and 3 are locally homologous

Sequences 1 and 3 are not

BLAST can be used to find proteins that share domains



Search a protein database for the sequence of a protein domain

BLAST will detect local sequence similarities in proteins that contain this domain

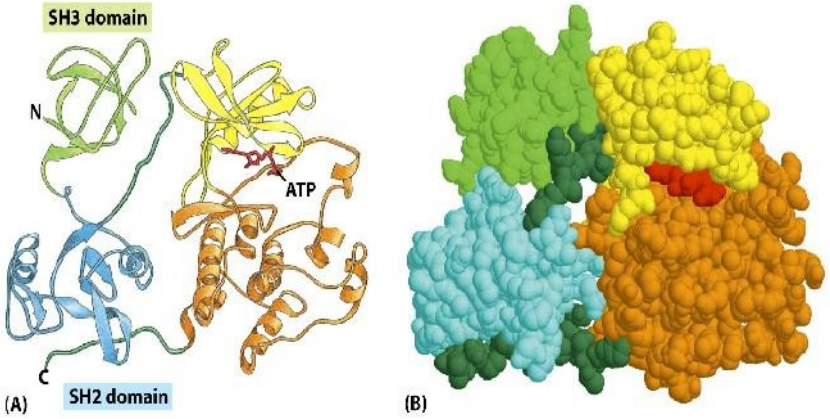


Figure 3-10 *Molecular Biology of the Cell* (© Garland Science 2008)

BLAST can be used to find out what is a protein domain

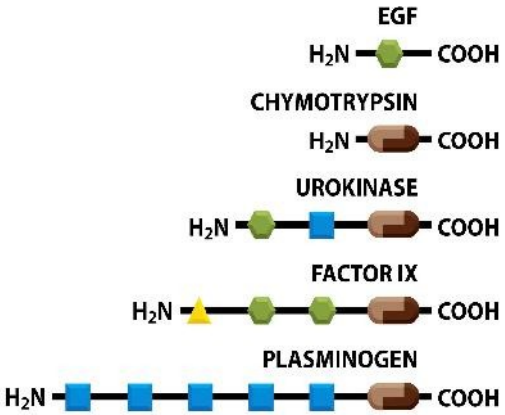


Figure 3-15 *Molecular Biology of the Cell* (© Garland Science 2008)

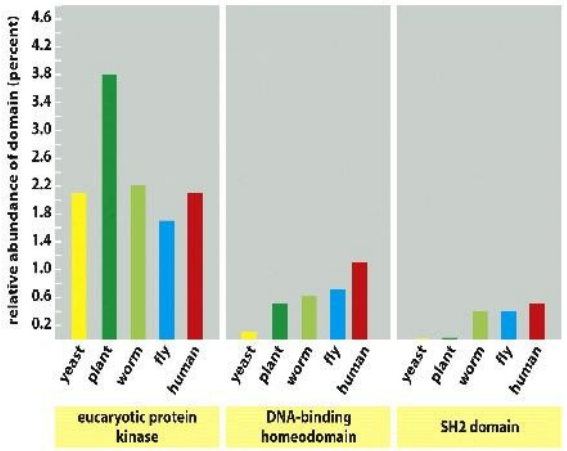


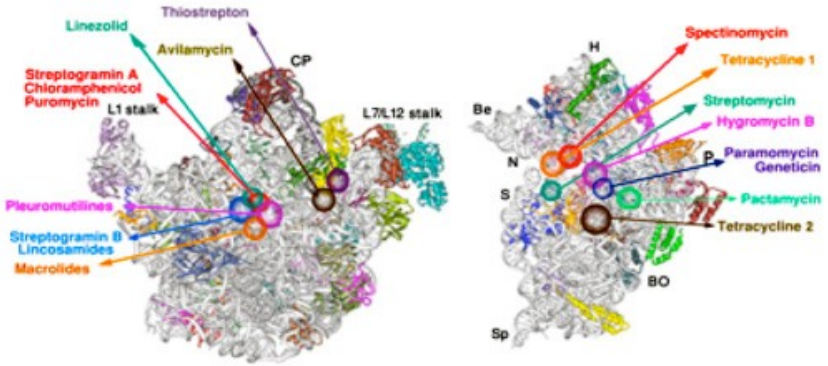
Figure 3-18 *Molecular Biology of the Cell* (© Garland Science 2008)

- Use all proteins in a protein database as a query and search the database again
- Have a close look at proteins that generate many hits

→Frequently reused code → Domain

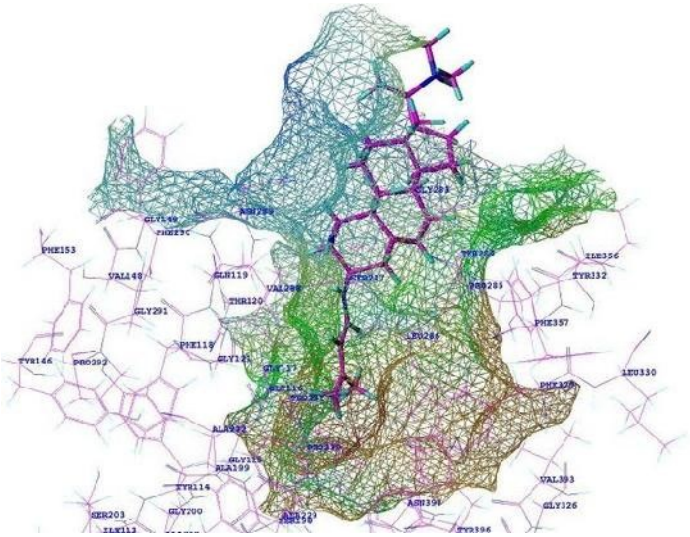
BLAST can be used to design antibiotics

You want to design an antibiotic that kills bacteria but does not hurt us



Compare many bacterial genomes and choose a target that
- kills bacteria
- is found in all targeted bacteria
- has no homologues in humans

BLAST can be used to predict drug side effects



You have a candidate drug that inhibits a target protein

What side effects might it have?

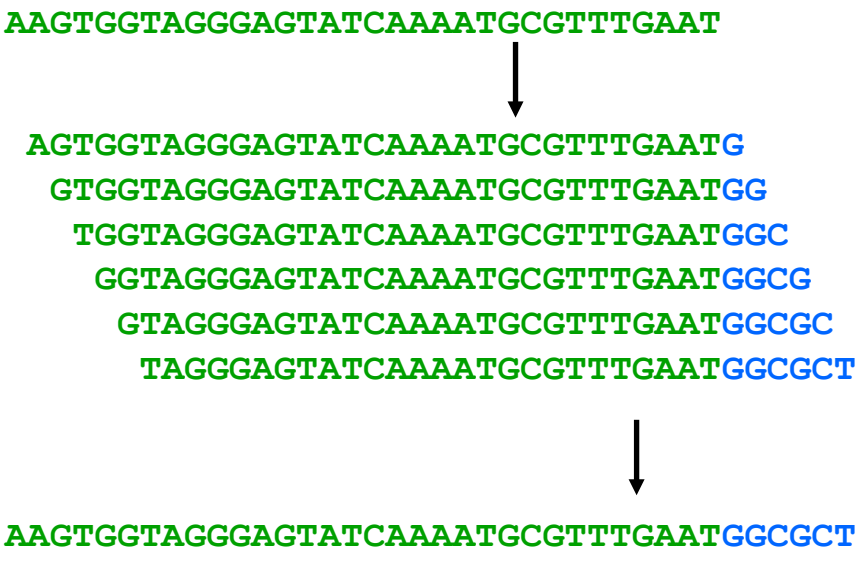
It might interfere with other proteins too

Which ones?

Those with strong similarities in the docking region

Search the human proteome for possible side targets

BLAST can be used in genome assembly



	Key	Value
1	AC	GTTA AATA
2	CT	TTTA
3	GC	TTTA
4	TT	TTAA CGTT

In chapter 3 we assembled reads via hashing. This does not work if there are sequencing errors. Then we need to allow for some mismatches.

BLAST can do this

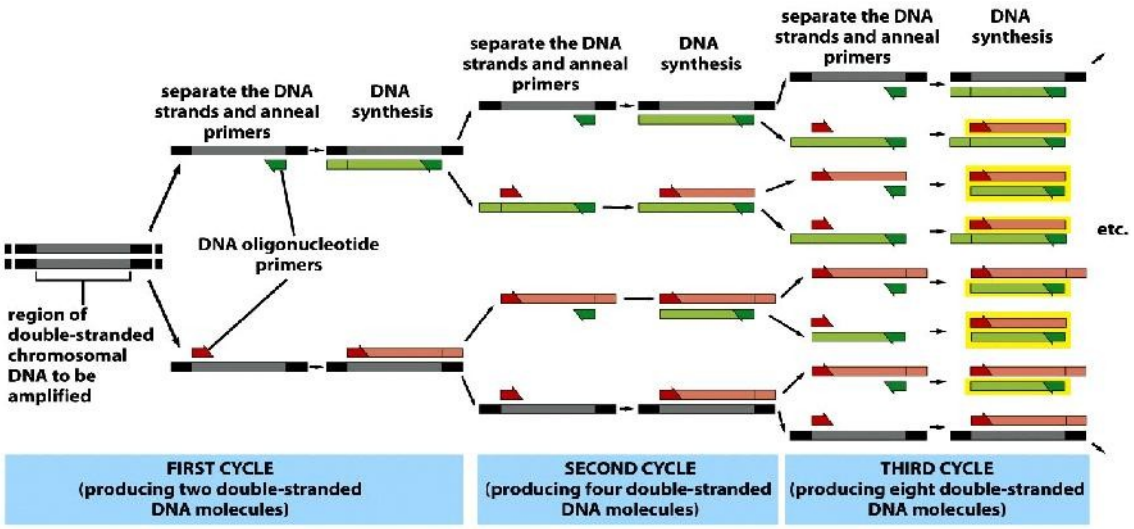
BLAST can be used to design primers for amplifying specific DNA sequences

You do not want to sequence a complete genome but just a specific gene of a patient to look for mutations

You can amplify a specific gene by PCR and a specific primer

A good primer is unique … it occurs only once in the genome

You can use BLAST to find unique primers



All these applications have in common that ...

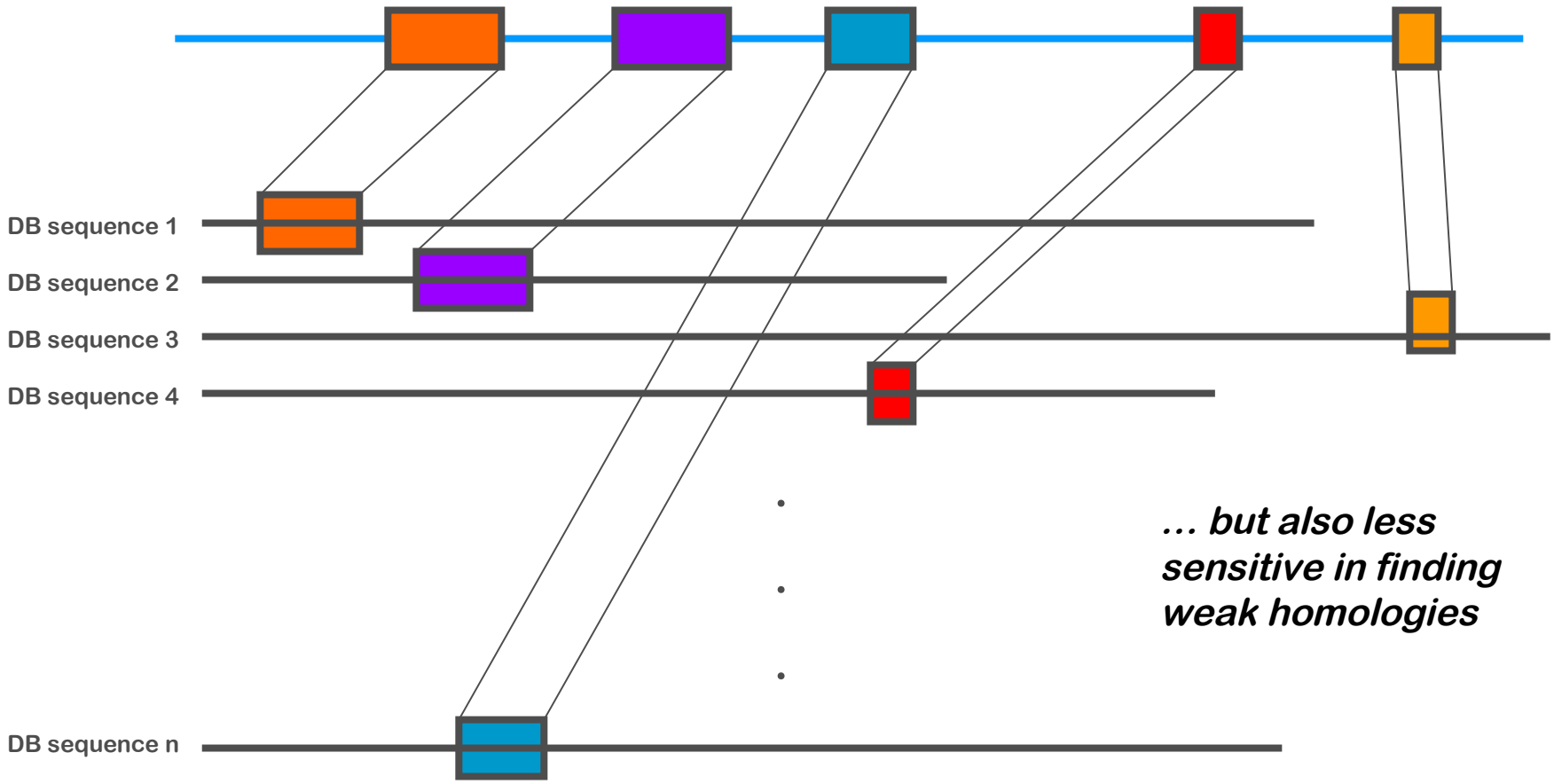
... my description greatly oversimplifies the problem

... you screen massive amounts of data for sequence similarity

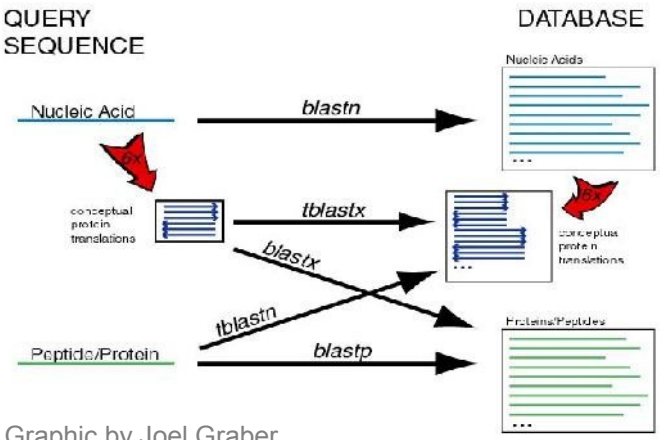
... there are better and more specialized programs to approach them

However BLAST can be used too. It's an all purpose program. Many biologists know how to use it and they use what they know

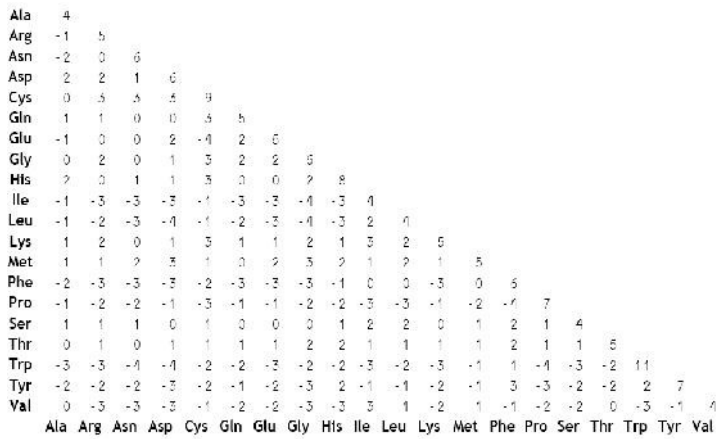
BLAST is faster than the Smith-Waterman algorithm in finding local sequence similarities



BLAST exploits the power of protein alignment with amino acid specific substitution scores whenever possible



Graphic by Joel Graber



Remember that DNA can be translated to protein in 6 different reading frames

```
5′ CAT CAA
 5′ ATC AAC
  5′ TCA ACT

        5′ CATCAACTACAACTCCAAAGACACCCTTACACATCAACAAACCTACCCAC 3′
        3′ GTAGTTGATGTTGAGGTTTCTGTGGGAATGTGTAGTTGTTTGGATGGGTG 5′

                                                  5′ GTG GGT
                                                   5′ TGG GTA
                                                    5′ GGG TAG
```

BLAST generates Smith-Waterman alignments of the top ranking sequences

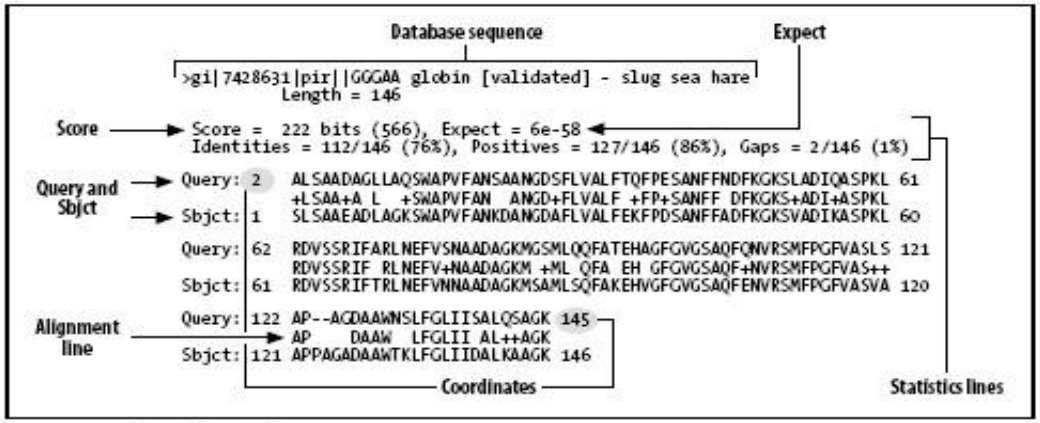


Figure 6-2. A BLASTP alignment

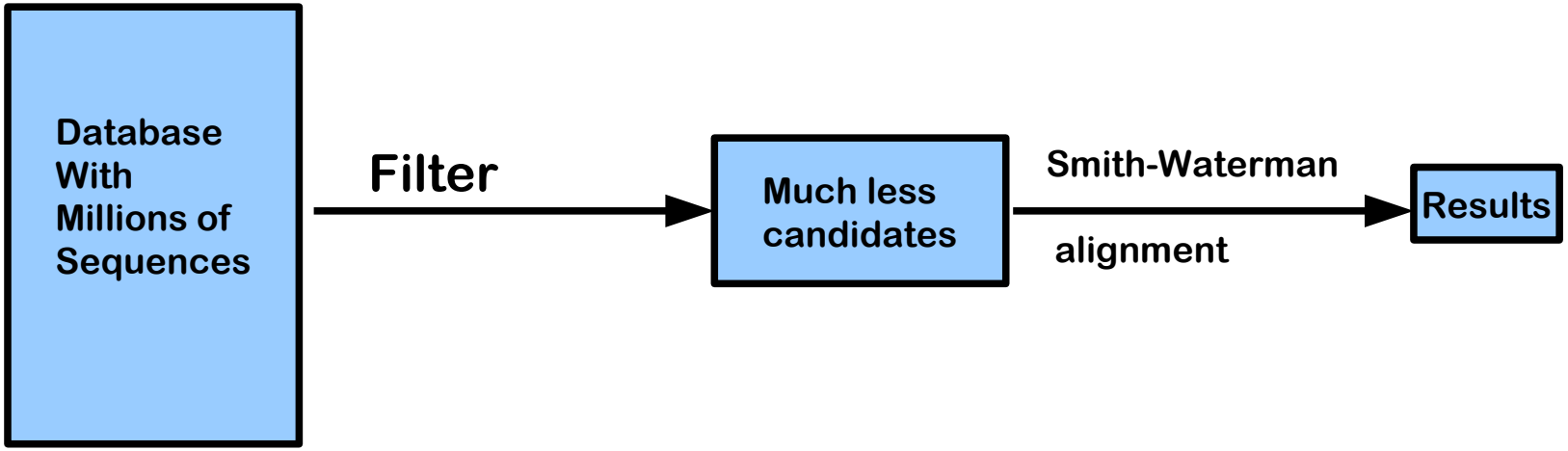
But it finds these sequences without calculating optimal alignments for all comparisons

BLAST is a sequence filter

BLAST is a sequence filter, or as the authors put it …

"The central idea of the BLAST algorithm is to confine attention to segment pairs that contain a word pair of length w with a score of at least T."

Altschul et al. (1990)



Score all pairs of w-words using a BLOSUM matrix

Word 1

CQE

Word 2

CEC

Scores

C -> C = +9

Q -> E = +2

E -> C = - 4

Word score: (=7)

CQE
CEC
9+2-4

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7

Section of BLOSUM62 substitution matrix

Compile a list of all w-letter words of the query sequence

Take the query (e.g. `LVNRKPVVP`)

Chop it into overlapping w-words

(protein sequence: w = 3, DNA sequence: w = 11)

```
Query:
LVNRKPVVP
Word1:     LVN
Word2:      VNR
Word3:       NRK
...                    ...
```

Generate a list of high scoring words for every k-word of the query

Query: . . . VPSRREMARATAGPALRDFRHVVLTAT . . .

EMA	=	14
AAA	=	2
AAD	=	- 4
....		
DMC	=	7
DMA	=	11
...		
YYW	=	- 6
YYY	=	- 5
...		

High Scoring Words (T ≥ 6)

EMA

DMC

DMA

. . .

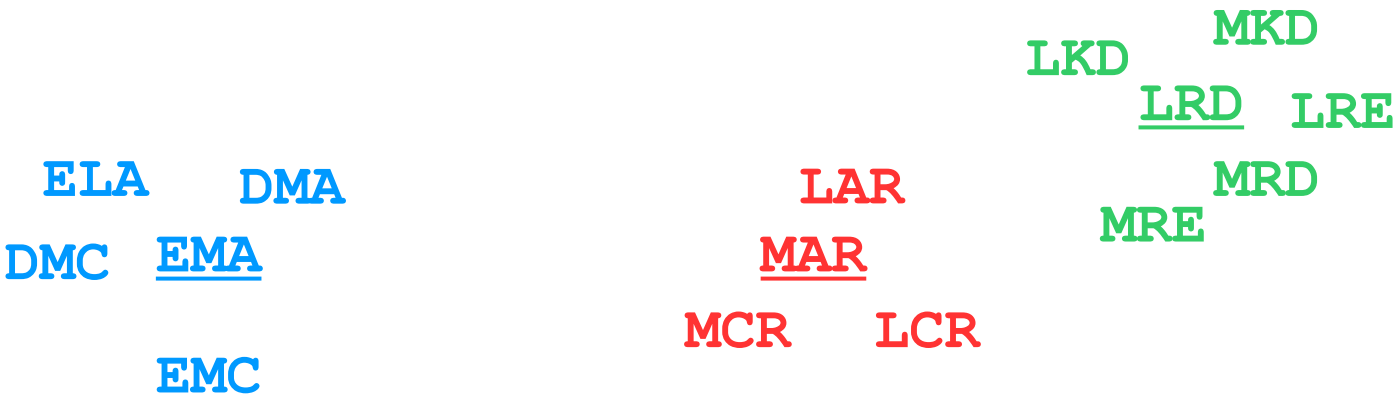
Protein: Number of 3-words = 20^3= 8000

DNA: Number of 11-words = 4^{11}= 4,194,304

Select words that have a score higher than some threshold *T*

Put all these lists together

Query: . . .VPSRREMARATAGPALRDFRHVVLTAT. . .



Tune the threshold T such that this list does not become too large

Search all occurrences of any of these words in every database sequence

This is the core computational problem:
This search needs to be fast

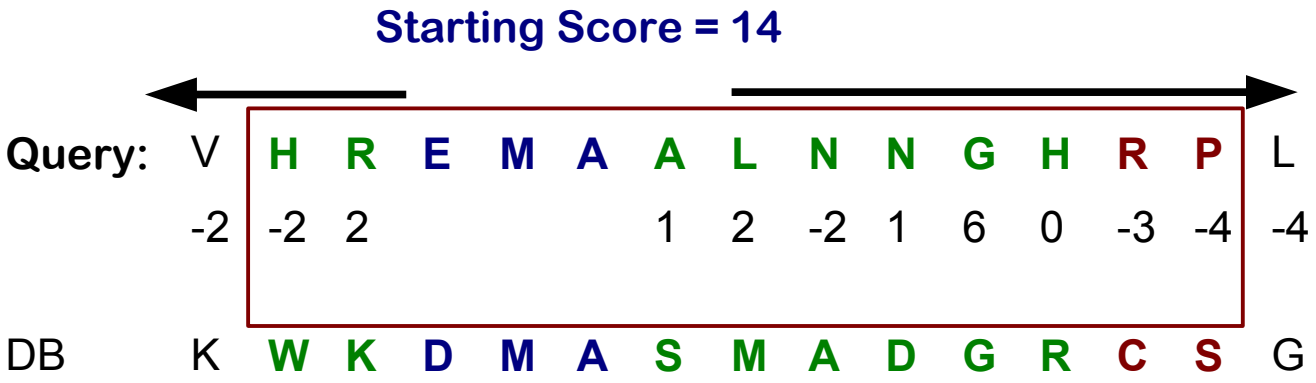
Alternative 1:
Preprocessing the database using suffix trees

Problem: whenever the database changes (every couple of minutes) you need to update the suffix tree

Alternative 2:
Preprocessing the word list using key word trees

We will discuss key word trees in detail later

Every hit is used as a seed to extend it to a longer gap less alignment

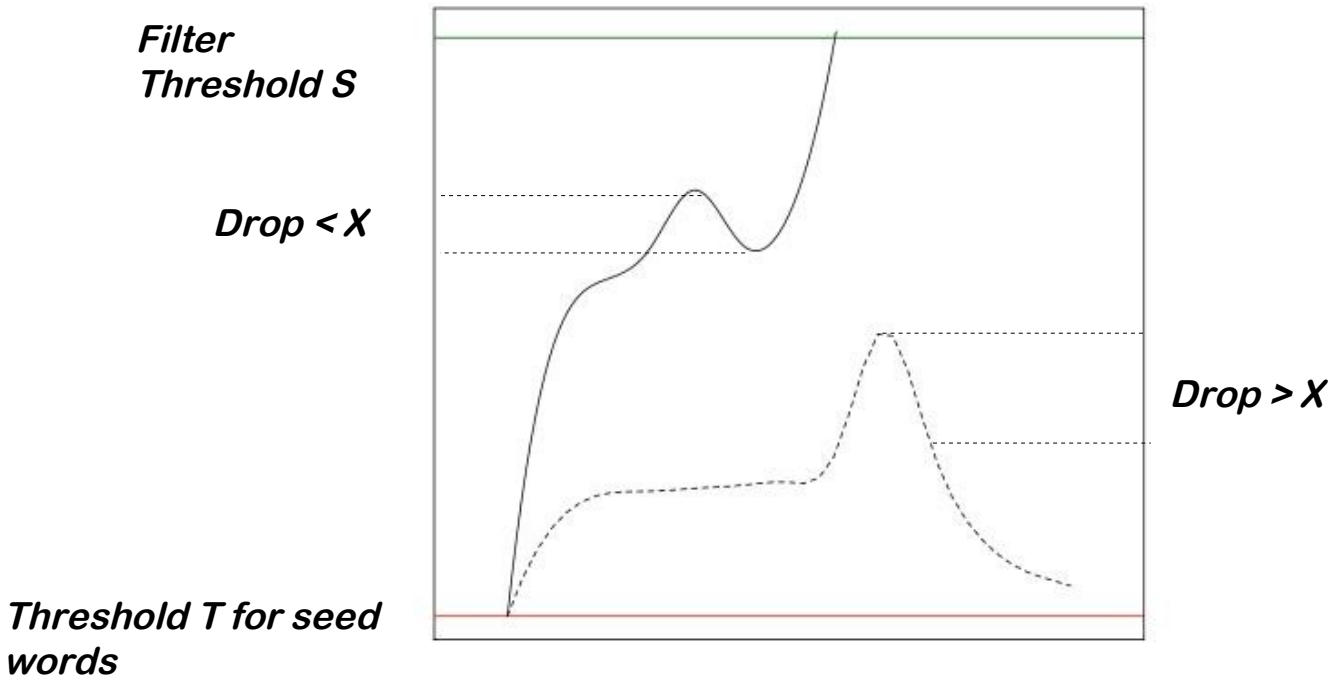


The alignment is extended in both directions until the sum of scores passes a threshold S or drops below some level X from the best known

If S is reached the database sequence passed the BLAST filter

If the extension terminated before S was reached other seeds in the database sequence will be checked. If no extension passes S the database sequence does not pass the BLAST filter

The margin X allows the extension to look a couple of positions ahead



Alignments are extended if the next positions yield positive scores. The margin X allows to bridge short stretches of negative scores

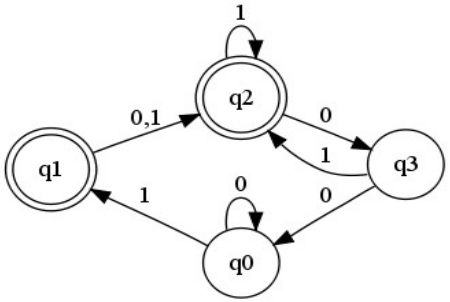
The filter works because the exact occurrences of words in long texts can be calculated efficiently

From the algorithmic perspective:

Homology search is a problem of approximate string matching

The BLAST framework translates the problem into multiple exact string matching problems, that can be tackled by suffix trees or …

Deterministic Finite Automata (DFA)



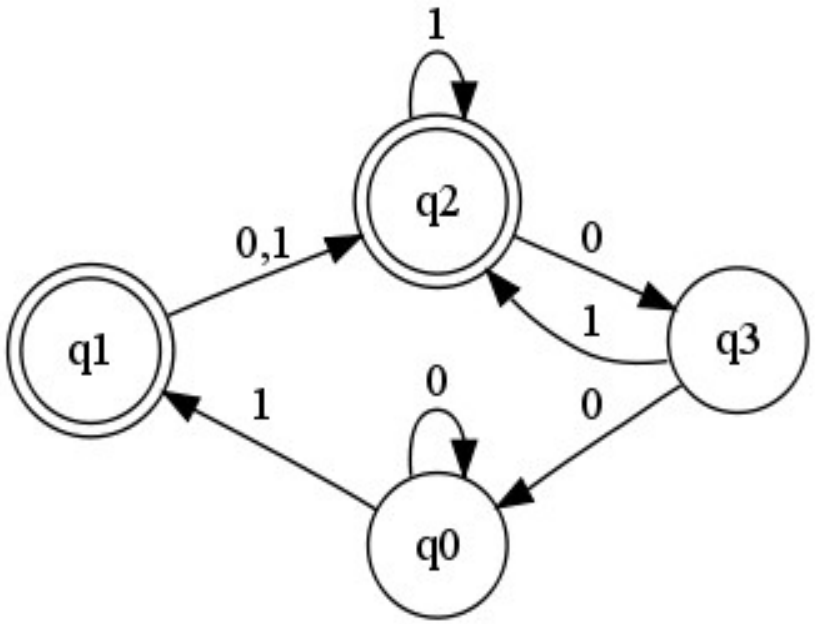
A DFA screens strings

Binary input: s='1011'

Read string s from left to right!

(i)	start with q0.
(ii)	read first character: 1
(iii)	go from q0 to q1.
(iv)	read next character: 0
(v)	go from q1 to q2.
(vi)	read next character: 1
(vii)	remain in q2.
(viii)	read last character: 1
(ix)	remain in q2

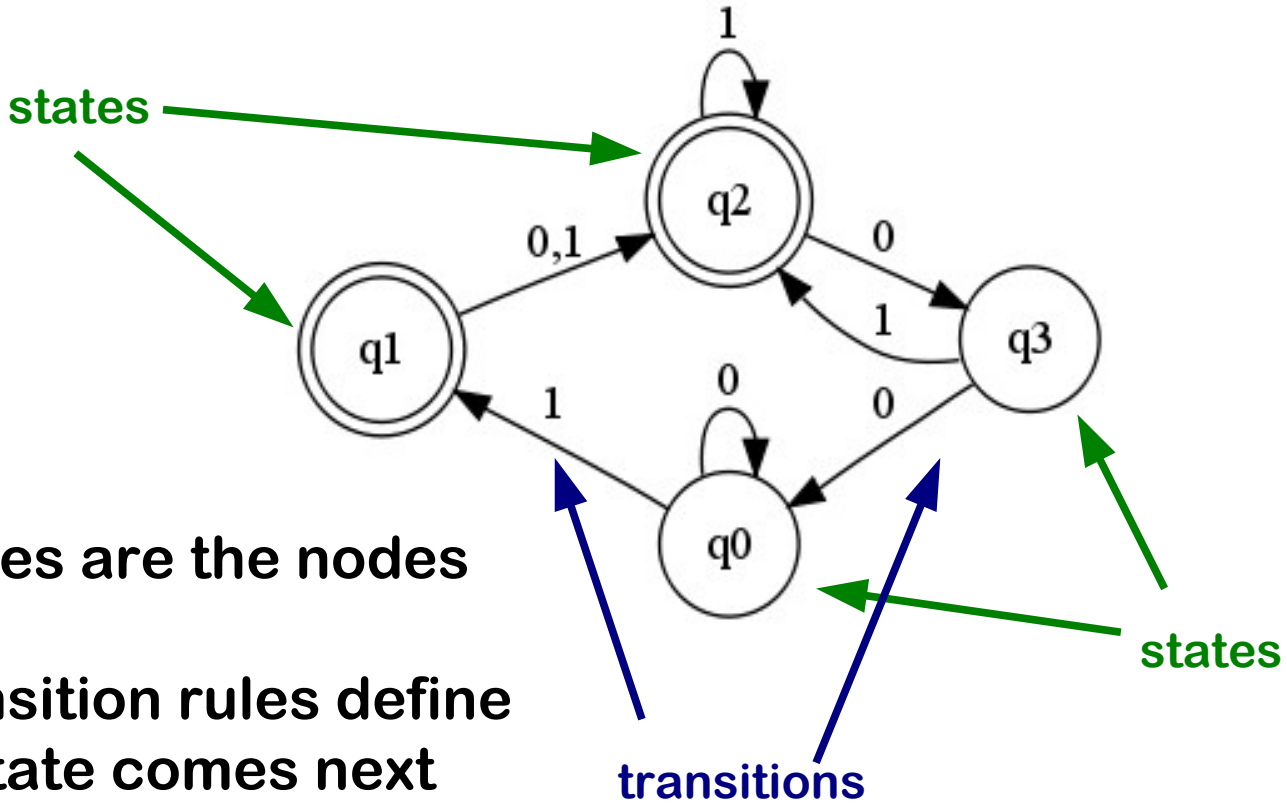
The DFA has terminated in q2



This DFA can screen any binary string

Depending on the string it will terminate in a different node

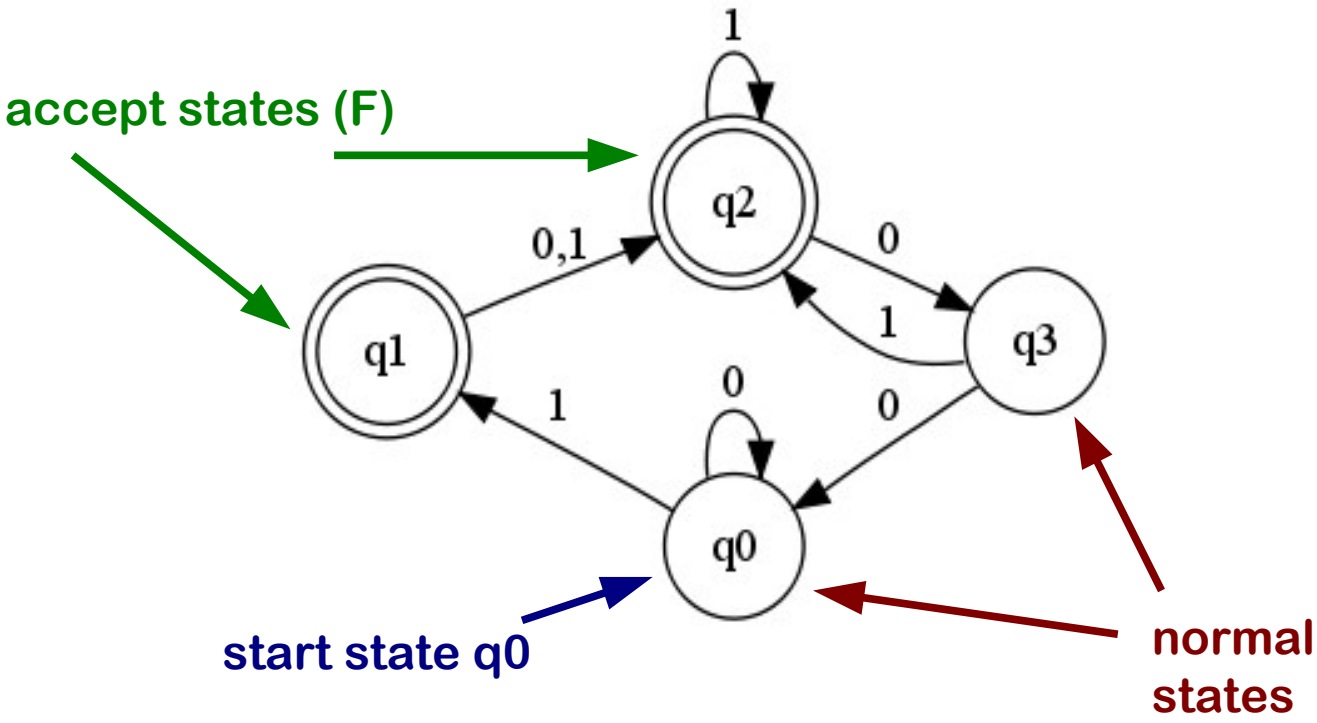
The DFA has states and transition rules



The states are the nodes

The transition rules define which state comes next depending on the next read input character

A DFA has three types of states



It starts in the start state

It accepts the input string only if it ends in an accept state
(Accept states have double circles)

The DFA accepts the input string only if it terminates in one of its accept states

Binary input: s='1011'

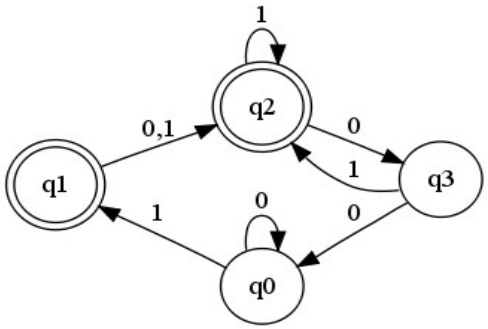
Read string s from left to right!

(i)	start with q0.
(ii)	read first character: 1
(iii)	go from q0 to q1.
(iv)	read next character: 0
(v)	go from q1 to q2.
(vi)	read next character: 1
(vii)	remain in q2.
(viii)	read last character: 1
(ix)	remain in q2

The DFA has terminated in q2

q2 is an accept state

The DFA accepts 1011



Binary input: s='1100'

Read string s from left to right!

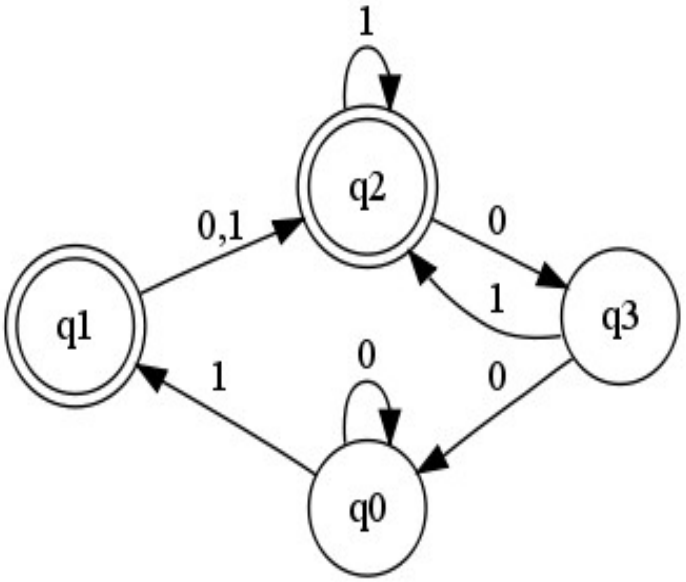
(i)	start with q0.
(ii)	read first character: 1
(iii)	go from q0 to q1.
(iv)	read next character: 1
(v)	go from q1 to q2.
(vi)	read next character: 0
(vii)	go from q2 to q3
(viii)	read last character: 0
(ix)	go from q3 to q0

The DFA has terminated in q0

q0 is no accept state

The DFA rejects 1100

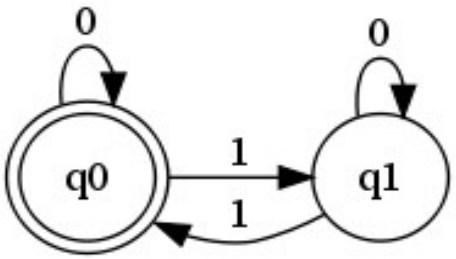
The transition rules can be stored in the goto table δ



input	q0	q1	q2	q3
0	q0	q2	q3	q0
1	q1	q2	q2	q2

If automaton is in state q0 (column index) and reads '1' (row index) as next character, then its state changes to δ(1,q0)=q1 (table entry)

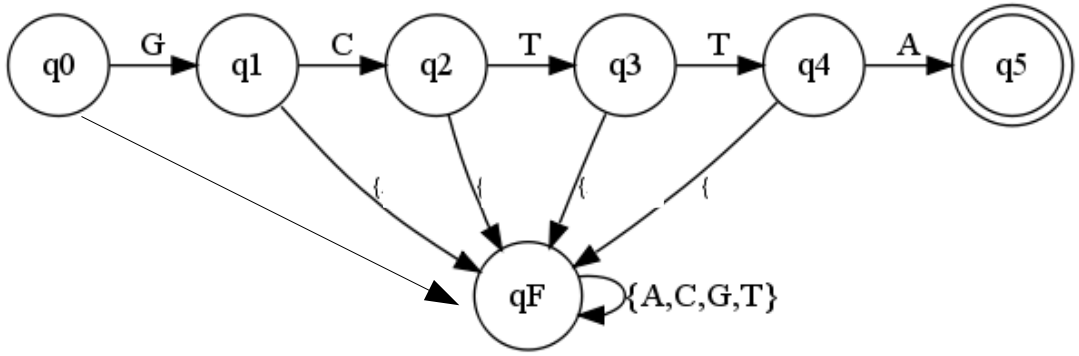
A DFA that accepts binary strings with an even number of "1"s



input	q0	q1
0	q0	q1
1	q1	q0

An automaton that only accepts the DNA sequence 'GCTTA'

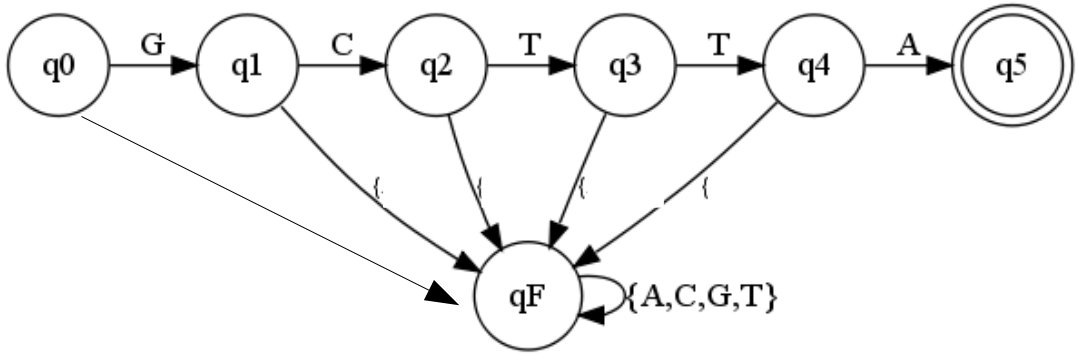
An automaton that only accepts the DNA sequence 'GCTTA'

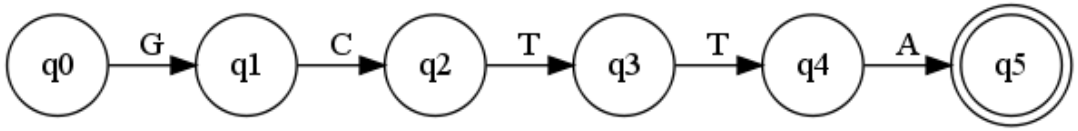


The fail state qF is a non-acceptance state with δ(qF, .) = qF

The program can be stopped once we reached qF

The graph representation of a DFA becomes easier to read if we omit the fail state qF and all transitions to it

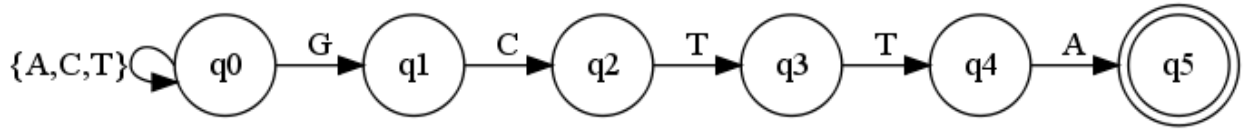




If no transition is specified for your current state and input character, terminate and reject the input

A program that returns all occurrences of "GCTTA" in the input string

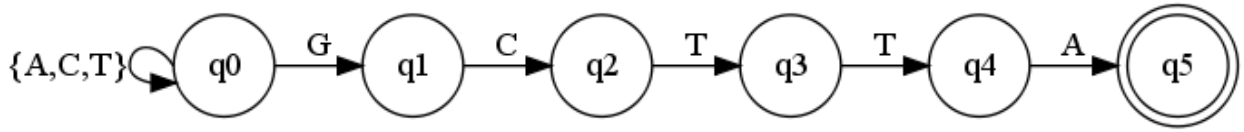
A program that returns all occurrences of "GCTTA" in the input string



If there is no transition specified go back to q0 (not qF) without proceeding to the next character

Whenever the automaton is in q5 it is at the end of an occurrence of GCTTA and generates an output without terminating

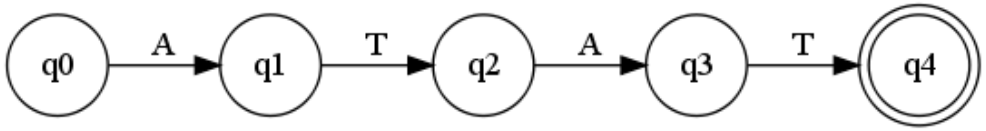
This program is an extension of the DFA concept



There is a new type of transition (auxiliary transition) that does not consume a character

The role of the state q5 has changed from an accept to an output state

A program that returns all (?) start points of "ATAT" in the input string



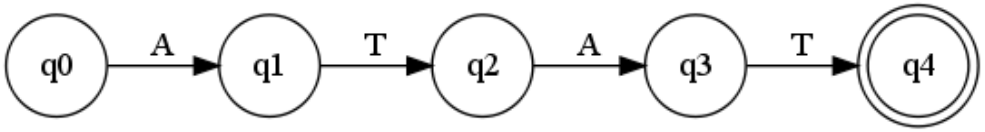
If there is no transition specified go back to q0 (not qF) without proceeding to the next character

What happens to the input string:

`GATTCATATATTTC ?`

The algorithm misses occurrences

GATTCATATATTTC



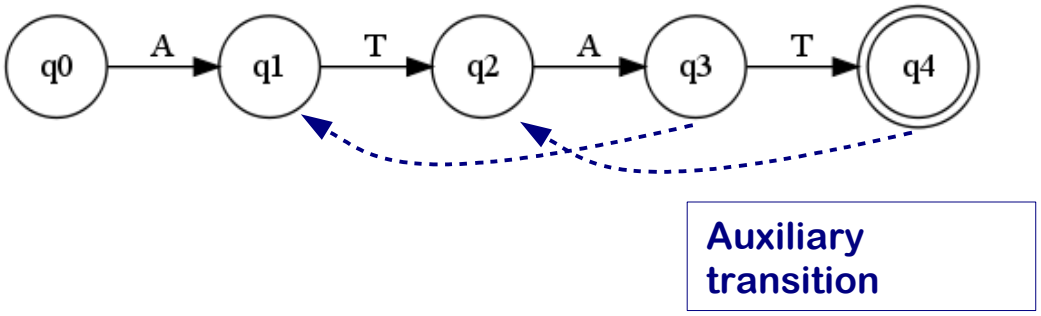
If there is no transition specified go back to q0 (not qF) without proceeding to the next character

After having read the green T the algorithm is in q4 and jumps back to q0 with remaining characters `ATTTC`. It misses the second `ATAT`

What type of search words cause this problem?

For search words where a suffix of a prefix matches a prefix of the word we need a special auxiliary link for output notes

AT is suffix but also prefix of ATAT and so is A a suffix of the prefix ATA

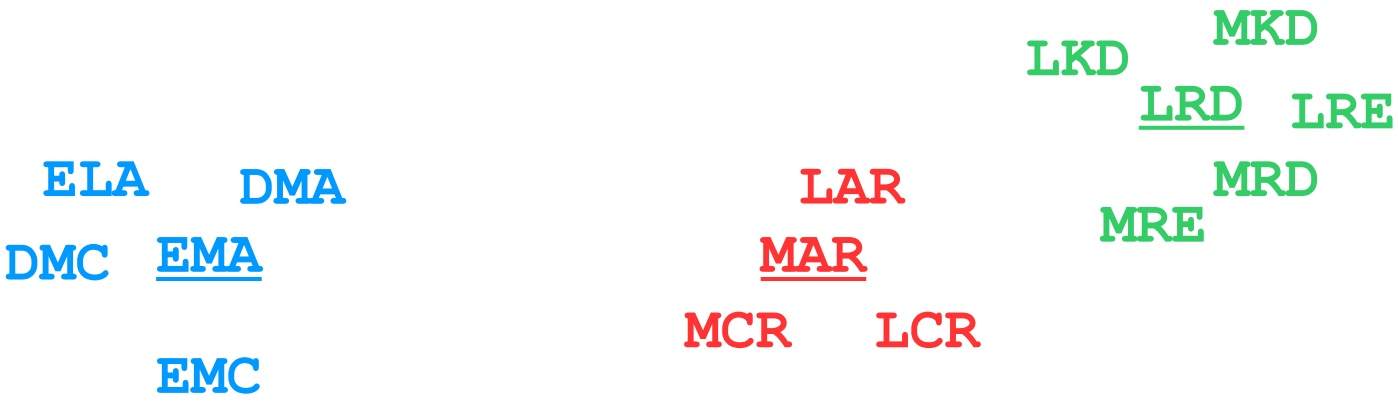


There is one auxiliary transition destination for each node

auxiliary(state) = longest proper prefix that is also a suffix

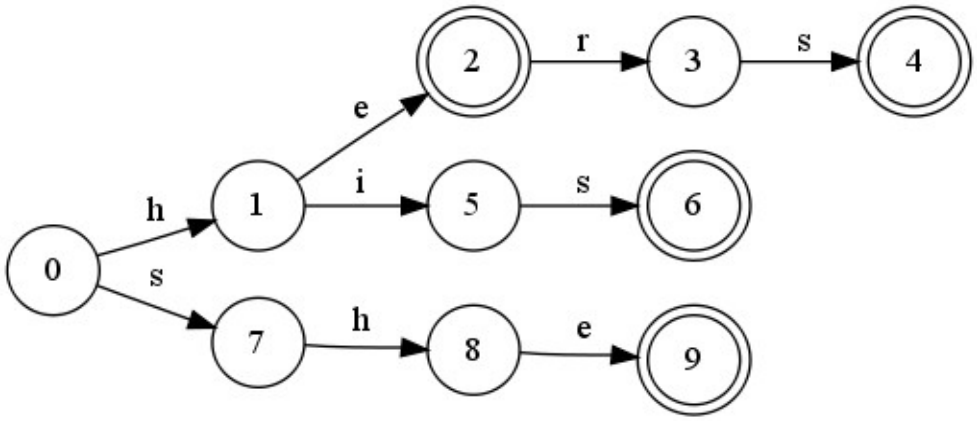
A program that finds all occurrences of the w-word lists generated by BLAST

Query: . . .VPSRREMARATAGPALRDFRHVVLTAT. . .



Generate a key word tree

Word List **= {he, she, his, hers}**



A key word tree can be built in O(n) where n is the sum of word lengths in the list

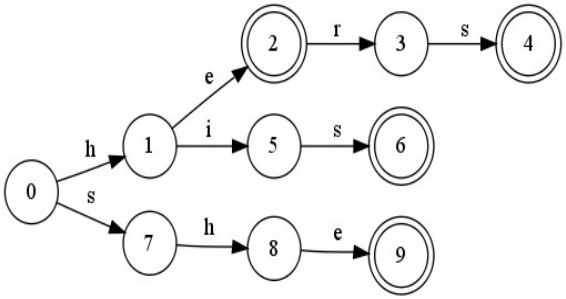
Word List = $\{W_1, \ldots, W_k\}$

Begin with a root node only and insert one word after the other

To insert W_i, start at the root and follow the path labeled by characters of W_i.

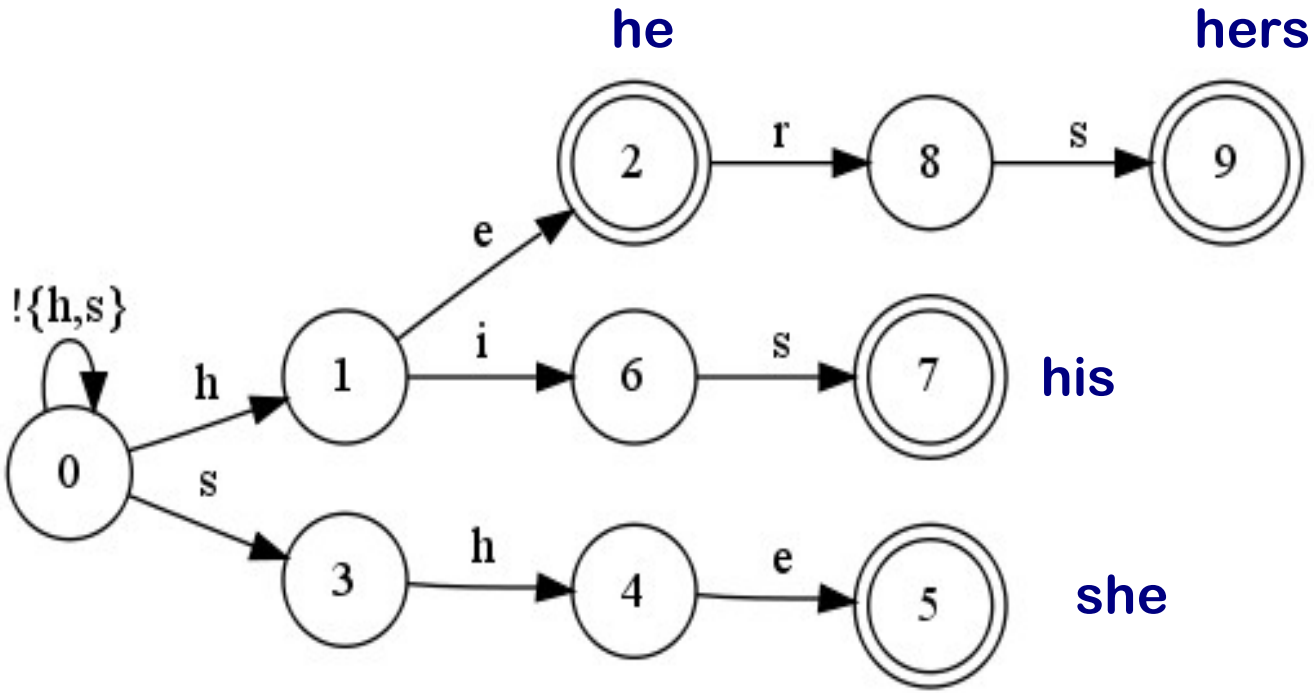
If the path ends before W_i, continue the branch by adding new edges and nodes for the remaining characters of W_i

Make the terminal node of the path an output node for W_i



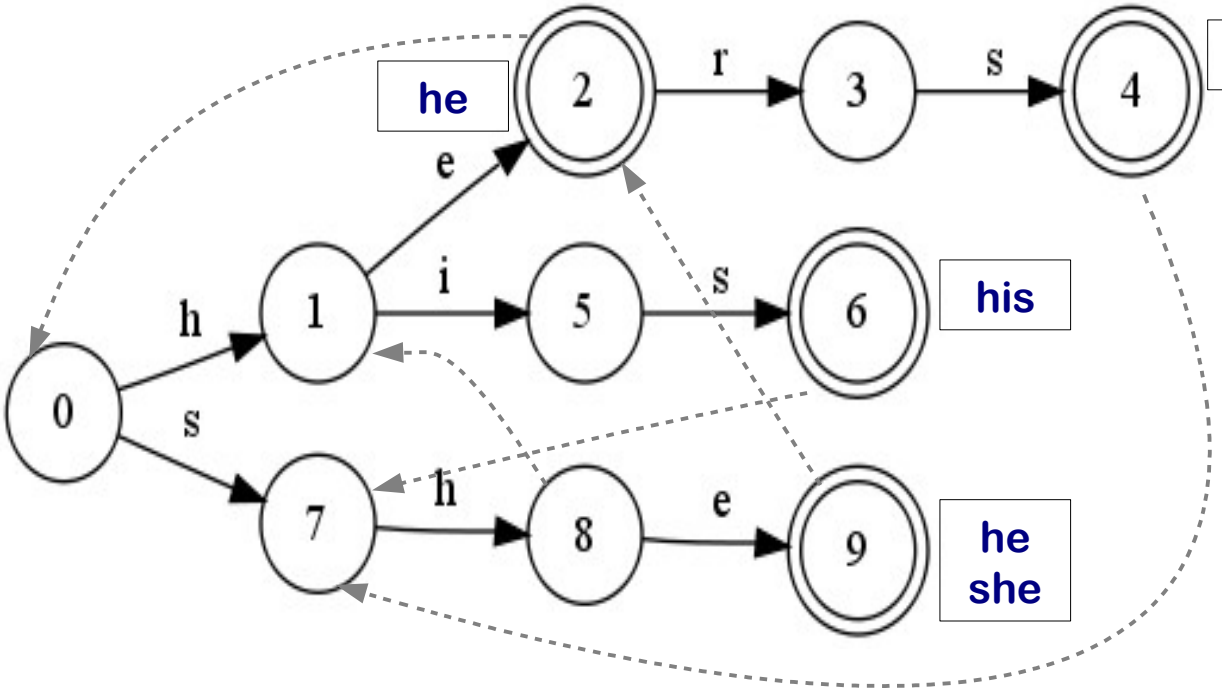
Use the key word tree as automaton

Automaton for W = {he, she, his, hers}:



What about the auxiliary transitions?

The auxiliary transition ensures that read characters are reused if necessary

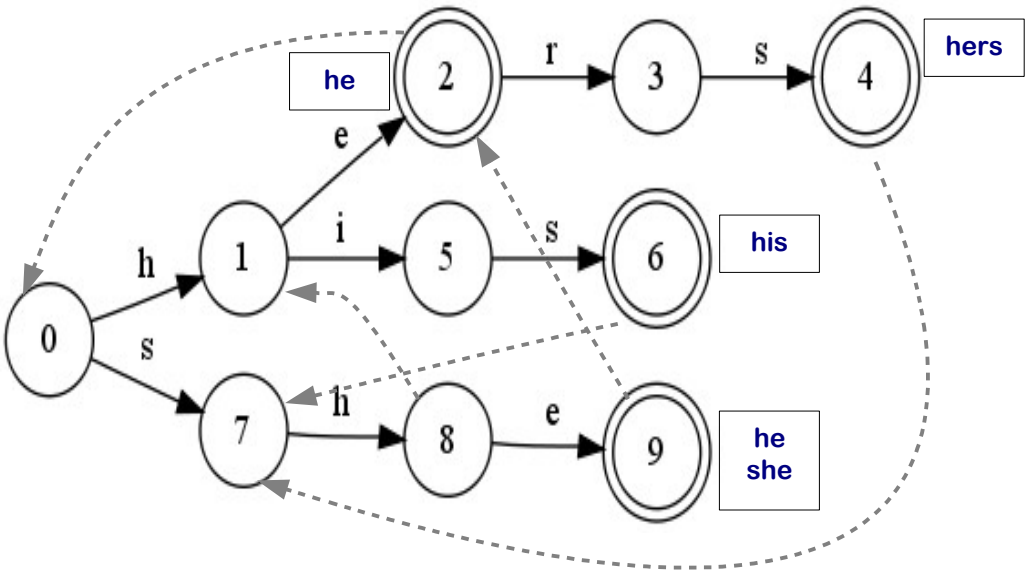


'he' is *longest proper suffix* of 'she'

This algorithm is called the Aho-Corasick algorithm

Why is it fast?

The Aho-Corasick algorithm needs to read every character of the database only once and finds all occurrences of any of the words from BLASTs list



Every character in the database is a transition in the automaton

Whenever we pass an output node there was a hit (end of word)

The search for all words is parallelized.

End of Chapter 10