Sequence comparison by compression

Motivation

- similarity as a marker for homology. And homology is used to infer function.
- Sometimes, we are only interested in a numerical distance between two sequences. For example, to infer a phylogeny.

	C	D	E	G	H
Cow	-	52	26	52	53
Dog		-	58	14	15
Eleph.			-	58	59
Gorila				-	5
Human					

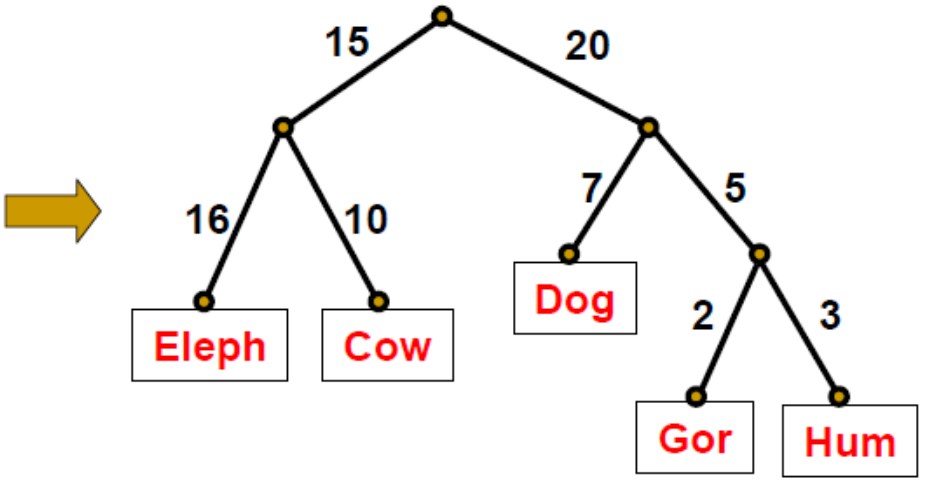


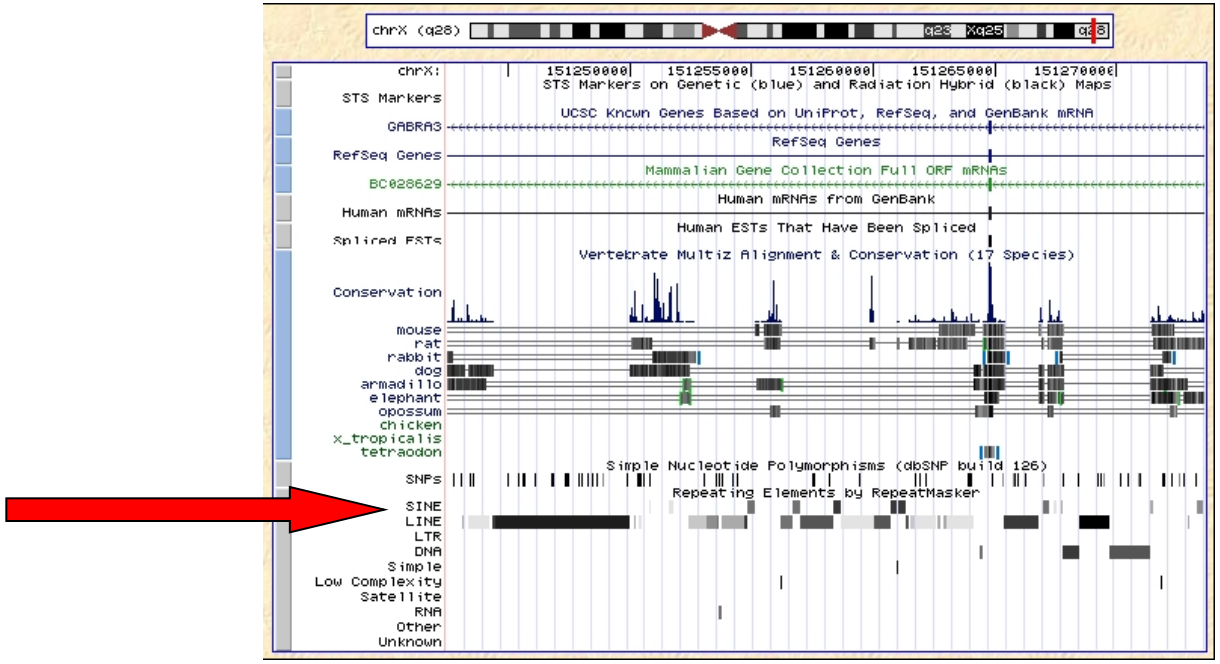
Figure adapted from http://www.inf.ethz.ch/personal/gonnet/acat2000/side2.html

Text- vs DNA compression

- `compress`, `gzip` or `zip` – routinely used to compress text files. They can be applied also to a text file containing DNA.
- E.g., a text file *F* containing chromosome 19 of human in fasta format – |*F*|= 61 MB, but |`compress`(*F*)|= 8.5 MB.
- 8 bits are used for each character. However, DNA consists of only 4 different bases – 2 bits per base are enough: A = 00, C = 01, G = 10, and T = 11.
- Applying a standard compression algorithm to a file containing DNA encoded using two bits per base will usually not be able to compress the file further.

The repetitive nature of DNA

- Take advantage of the repetitive nature of DNA!!
- LINEs (Long Interspersed Nuclear Elements), SINEs.



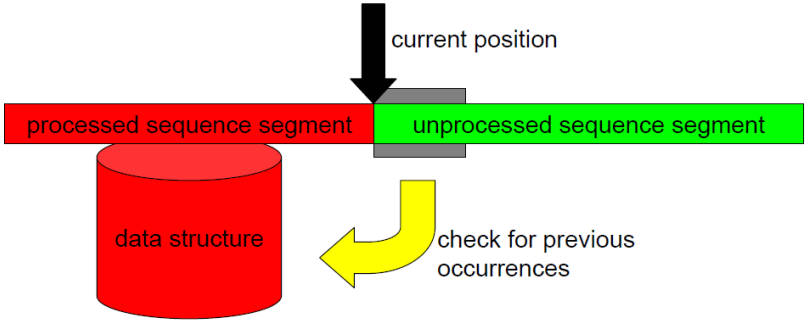
- UCSC Genome Browser: http://genome.ucsc.edu

DNA compression

- DNA sequences are very compressible, especially for higher eukaryotes: they contain many repeats of different size, with different numbers of instances and different amounts of identity.
- **A first idea:** While processing the DNA string from left to right, detect exact repeats and/or palindromes (reverse-complemented repeats) that possess previous instances in the already processed text and encode them by the length and position of an earlier occurrence. For stretches of sequence for which no significant repeat is found, use two-bit encoding. (The program Biocompress is based on this idea.)
  - Data structure for fast access to sequence patterns already encountered.
  - Sliding window along unprocessed sequence.

DNA compression

- **A first idea:** While processing the DNA string from left to right, detect exact repeats and/or palindromes (reverse-complemented repeats) that possess previous instances in the already processed text and encode them by the length and position of an earlier occurrence. For stretches of sequence for which no significant repeat is found, use two-bit encoding. (The program Biocompress is based on this idea.)
  - Data structure for fast access to sequence patterns already encountered.
  - Sliding window along unprocessed sequence.



DNA compression

- **A second idea:** Build a suffix tree for the whole sequence and use it to detect maximal repeats of some fixed minimum size. Then code all repeats as above and use two-bit encoding for bases not contained inside repeats. (The program Cfact is based on this idea.)
- Both of these algorithms are lossless, meaning that the original sequences can be precisely reconstructed from their encodings. An number of lossy algorithms exist, which we will not discuss here.
- In the following we will discuss the GenCompress algorithm due to Xin Chen, Sam Kwong and Ming Li.

Edit operations

- The main idea of GenCompress is to use inexact matching, followed by edit operations. In other words, instances of inexact repeats are encoded by a reference to an earlier instance of the repeat, followed by some edit operations that modify the earlier instance to obtain the current instance.
- Three standard edit operations:

1.	Replace: ***(R, i, char)***	replace the character at position ***i*** by character ***char***.
2.	Insert: ***(I, i, char)***	insert character ***char*** at position ***i***.
3.	Delete: ***(D, i)***	delete the character at position ***i***.

Positions numbered from 0.

Edit operations

- different edit operation sequences:
  - (a) CCCCRCCCCC
    gaccgtcatt
    gaccttcatt
    or (b) CCCCDCICCCC
    gaccgt catt
    gacc ttcatt
- infinite number of ways to convert one string into another.
- Given two strings q and p. An edit transcript $\lambda(q, p)$ is a list of edit operations that transforms q into p.
- E.g., in case (a) the edit transcript is:

  $\lambda(\texttt{gaccgtcatt},\texttt{gaccttcatt}) = (R, 4, \texttt{t}),$
- whereas in case (b) it is:

  $\lambda(\texttt{gaccgtcatt},\texttt{gaccttcatt}) = (D, 4), (I, 5, \texttt{t})$.

  (positions start at 0 and are given relative to current state of the string, as obtained by application of all preceding edit operations.)

Encoding DNA

1. Using the **two-bit encoding** method, gaccttcatt can be encoded in 20 bits:

   10 00 01 01 11 11 01 00 11 11

   g a c c t t c a t t

a	00
c	01
g	10
t	11

   The following three methods encode gaccttcatt *relative to* gaccgtcatt:

2. In the **exact matching method** we use a pair of numbers *(repeat − position, repeat − length)* to represent an exact repeat. We can encode gaccttcatt as **(0, 4), t, (5, 5),** relative to gaccgtcatt. Let 4 bits encode an integer, 2 bits encode a base and one bit to indicate whether the next part is a pair (indicating a repeat) or a base. We obtain an encoding in 21 bits:

   0 0000 0100 1 11 0 0101 0101

   0 4 t 5 5

Encoding DNA

3. In the **approximate matching method** we can encode **gaccttcatt** as **(0, 10)**, **(*R*, 4, t)** relative to **gaccgtcatt**. Let us encode ***R*** by 00, ***I*** by 01, ***D*** by 11 and use a single bit to indicate whether the next part is a pair or a triplet. We obtain an encoding in 18 bits:

   0 0000 1010 1 00 0100 11

4. For **approximate matching**, we could also use the edit sequence **(*D*, 4)**, **(*I*, 4, t)**, for example, yielding the relative encoding

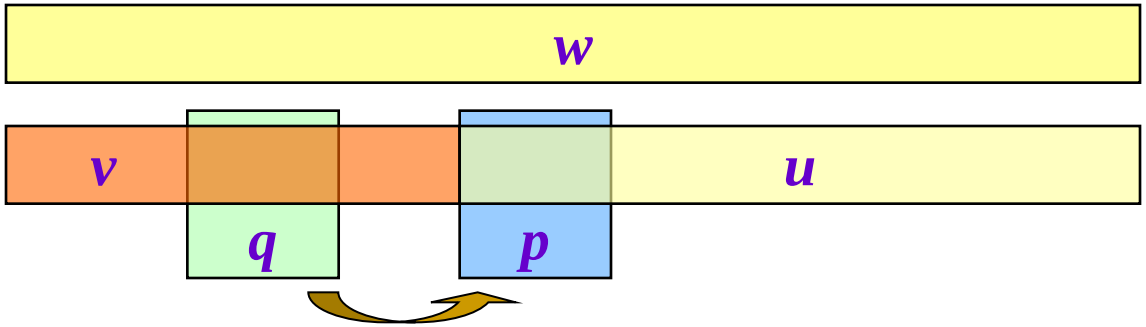
   (0, 10), (*D*, 4), (*I*, 4, t),

   which uses 25 bits:

   0 0000 1010 1 11 0100 1 01 0100 11.

GenCompress

- a one-pass algorithm based on approximate matching
- For a given input string w, assume that a part v has already been compressed and the remaining part is u, with $w = vu$. The algorithm finds an "optimal prefix" p of u that approximately matches some substring q of v such that p can be encoded economically. After outputting the encoding of p, remove the prefix p from u and append it to v. If no optimal prefix is found, output the next base and then move it from u to v. Repeat until $u = \varepsilon$.



The condition C

- How do we find an "optimal prefix" p? The following condition will be used to limit the search.
- Given two numbers k and b. Let p be a prefix of the unprocessed part u and q a substring of the processed part v. If $|q| > k$, then any transcript (q, p) is said to satisfy the condition $C = (k, b)$ for compression, if its number of edit operations is $\leq b$.
- Experimental studies indicate that $C = (k, b) = (12, 3)$ gives good results.
- In other words, when attempting to determine the optimal prefix for compression, we will only consider repeats of length $\geq k$ that require at most b edit operations.

The compression gain function
(the number of saved bits)

- We define a compression gain function G to determine whether a particular approximate repeat q, p and edit transcript λ are beneficial for the encoding:
- $G(q, p, \lambda) = \max \{ 2|p| - |(i, |q|)| - w_\lambda \cdot |(q, p)| - c, 0 \}$
- where
  - p is a prefix of the unprocessed part u,
  - q is a substring of the processed part v of length $|q|$ that starts at position i,
  - $2|p|$ is the number of bits that the two-bit encoding would use,
  - $|(i, |q|)|$ is the encoding size of $(i, |q|)$,
  - w_λ is the cost of encoding an edit operation,
  - $|(q, p)|$ is the number of edit operations in (q, p),
  - and c is the overhead proportional to the size of control bits.

The GenCompress algorithm

Input: A DNA sequence w, parameter $C = (k, b)$

Output: A lossless compressed encoding of w

Initialization: $u = w$ and $v = \varepsilon$ ← The empty word

while $u \neq \varepsilon$ do

Search for an optimal prefix p of u

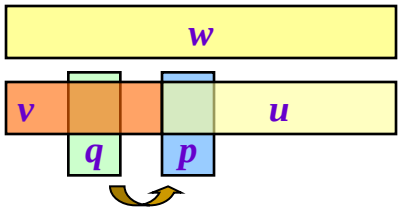
if an optimal prefix p with repeat q in v is found **then**

Encode the repeat representation $(i, |q|)$, where i is the position of q in v, together with the shortest edit transcript (q, p). Output the code.

else Set p equal to the first character of u, encode and output it.

Remove the prefix p from u and append it to v

end

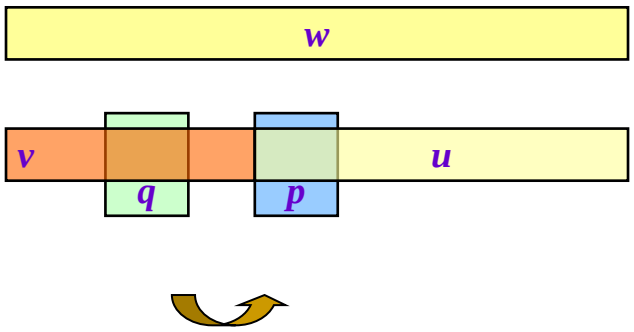


Implementing the optimal prefix search

- search for the optimal prefix – too slow
  - **Lemma**: An optimal prefix p always ends right before a mismatch.
  - **Lemma:** Let $\lambda(q, p)$ be an optimal edit sequence from q to p. If q_i is copied onto p_j in λ, then λ restricted to $(q_{0:i}, p_{0:j})$ is an optimal transcript from $q_{0:i} = q_0 q_1 \ldots q_i$ to $p_{0:j} = p_0 p_1 \ldots, p_j$.
  - simplified as follows:
    - to find an approximate match for p in v, we look for an exact match of the first l bases in p, where l is a fixed small number
    - an integer is stored at each position i of v that is determined by the the word of length l starting at i.

Implementing the optimal prefix search

1. Let $w = vu$ where v has already been compressed.
2. Find all occurrences $u_{0:l}$ in v, for some small l. For each such occurrence, try to extend it, allowing mismatches, limited by the above observations and condition C. Return the prefix p with the largest compression gain value G.



Performance of GenCompress

- any nucleotide can be encoded canonically using 2 bits, we define the compression ratio of a compression algorithm as

$$1 - \frac{|O|}{2|I|}$$

$|I|$ is the number of bases in the input DNA sequence

$|O|$ is the length (number of bits) of the output sequence

- Alternatively, if our DNA string is already encoded canonically, we can define the compression ratio of a compression algorithm as

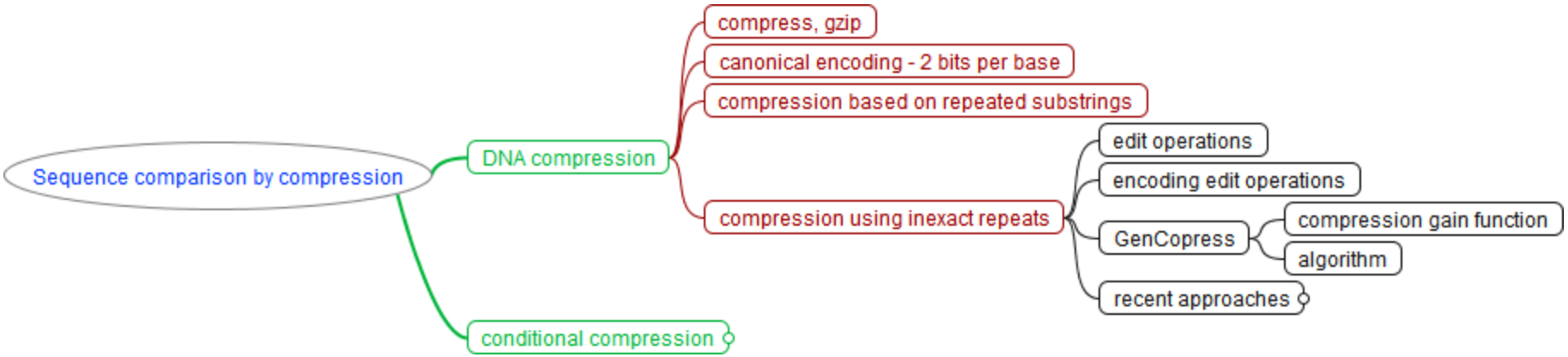
$$1 - \frac{|O|}{|I|}$$

$|I|$ is the number of bits in the canonical encoding of the input DNA sequence and $|O|$ is the length (number of bits) of the output sequence.

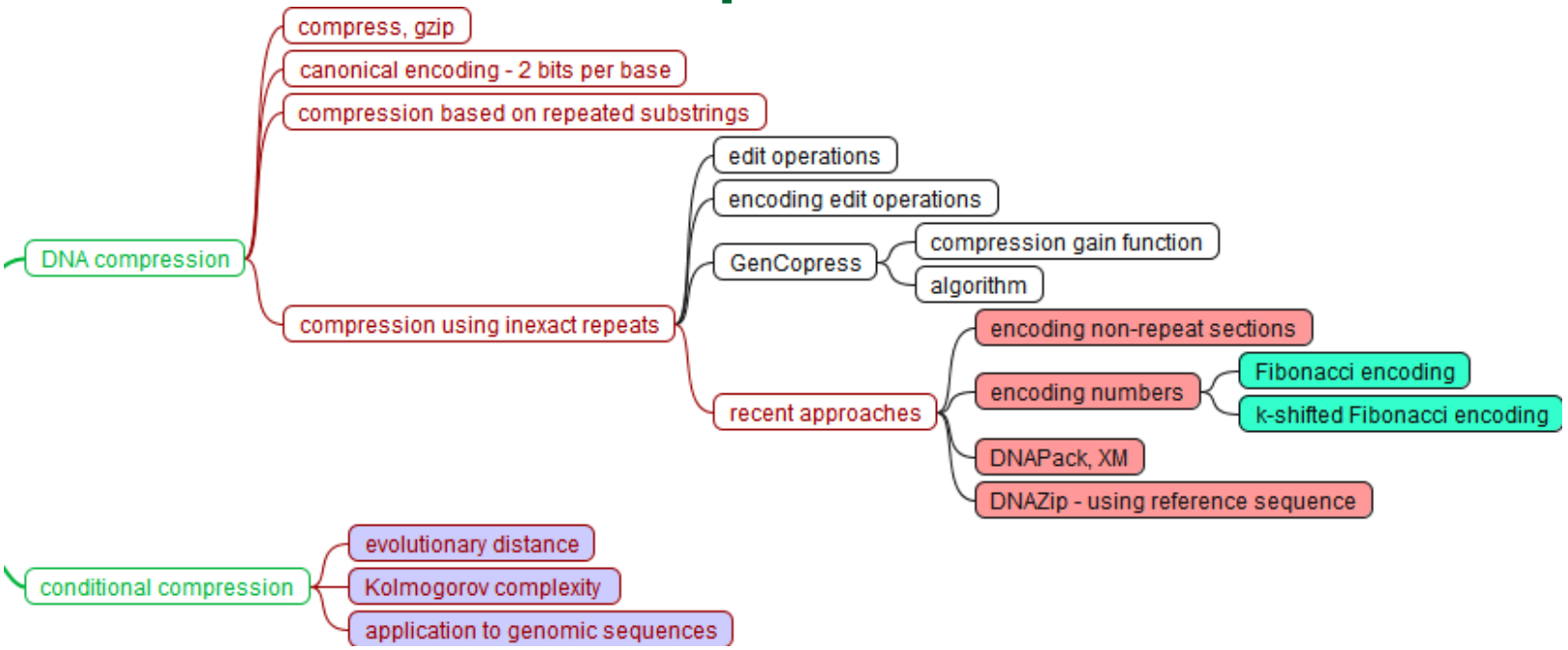
Performance of GenCompress

sequence	size	*compress*	*arith-2*	*Biocompress-2*	*GenCompress-1*	*GenCompress-2*
MTPACGA	100314	-5.81%	6.37%	6.24%	6.88%	6.88%
MPOMTCG	186608	-10.11%	1.72%	3.11%	4.71%	4.71%
CHNTXX	155844	-9.36%	3.31%	19.14%	19.27%	19.27%
CHMPXX	121024	-3.73%	8.17%	15.76%	16.38%	16.35%
HUMGHCSA	66495	-9.68%	3.11%	34.63%	44.99%	44.76%
HUMHBB	73323	-9.73%	4.08%	6.16%	8.98%	9.04%
HUMHDABCD	58864	-11.48%	2.87%	6.15%	9.27%	9.04%
HUMDYSTROP	38770	-11.66%	3.80%	3.69%	3.88%	3.87%
HUMHPRTB	56737	-10.12%	3.56%	4.67%	7.67%	7.67%
VACCG	191737	-8.37%	5.10%	11.93%	11.94%	11.93%
HEHCMVCG	229354	-10.65%	1.76%	7.60%	7.65%	7.65%

Till now

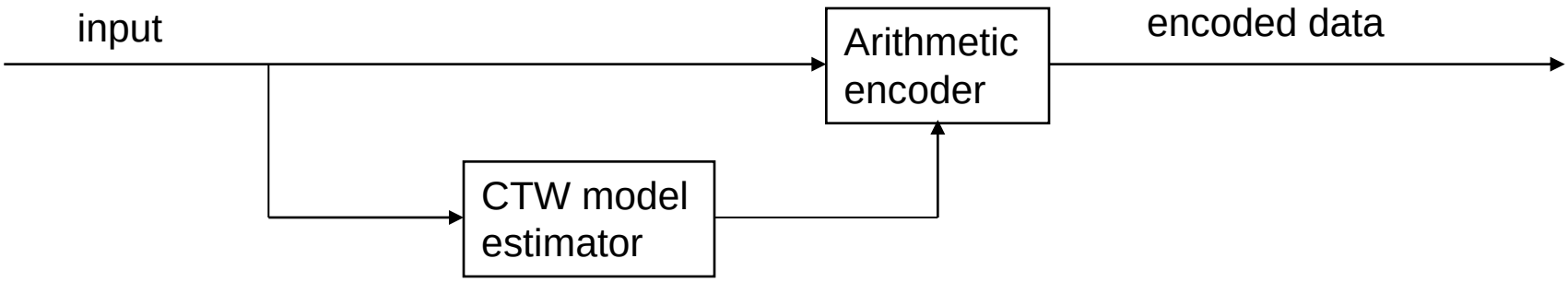

Sequence comparison by compression
DNA compression
compress, gzip
canonical encoding - 2 bits per base
compression based on repeated substrings
compression using inexact repeats
edit operations
encoding edit operations
GenCopress
compression gain function
algorithm
recent approaches
conditional compression

Next – recent approaches, conditional compression



Recent approaches
Encoding of non-repeat regions

- **Order-2 arithmetic encoding**–the adaptive probability of a symbol is computed from the context (the last 2 symbols) after which it appears –3 symbols code one amino-acid???
- **Context tree weighting coding (CTW)**–a tree containing all processed substrings of length *k* is built dynamically and each path (string) in the tree is weighted by its probability – these probabilities are used in an arithmetic encoder



Recent approaches
Encoding of numbers

- **Fibonacci encoding**
  - any positive integer can be uniquely expressed as the sum of distinct Fibonacci numbers so, that no two consecutive Fibonacci numbers are used
  - by adding a 1 after the bit corresponding to the largest Fibonacci number used in the sum the representation becomes self-delimited

1, 2, 3, 5, 8, 13, 21, …

	1	2	3	4	8	18
Fibonacci	11	011	0011	1011	000011	0001011

Recent approaches
Encoding of numbers

- ***k*- Shifted Fibonacci encoding**
  - usually there are many small numbers and few large numbers to encode
  - $n \in \{1,\ldots,2^k - 1\}$ – normal binary encoding
  - $n \geq 2^k$ – as 0^k followed by Fibonacci encoding of $n - (2^k - 1)$

1, 2, 3, 5, 8, 13, 21, …

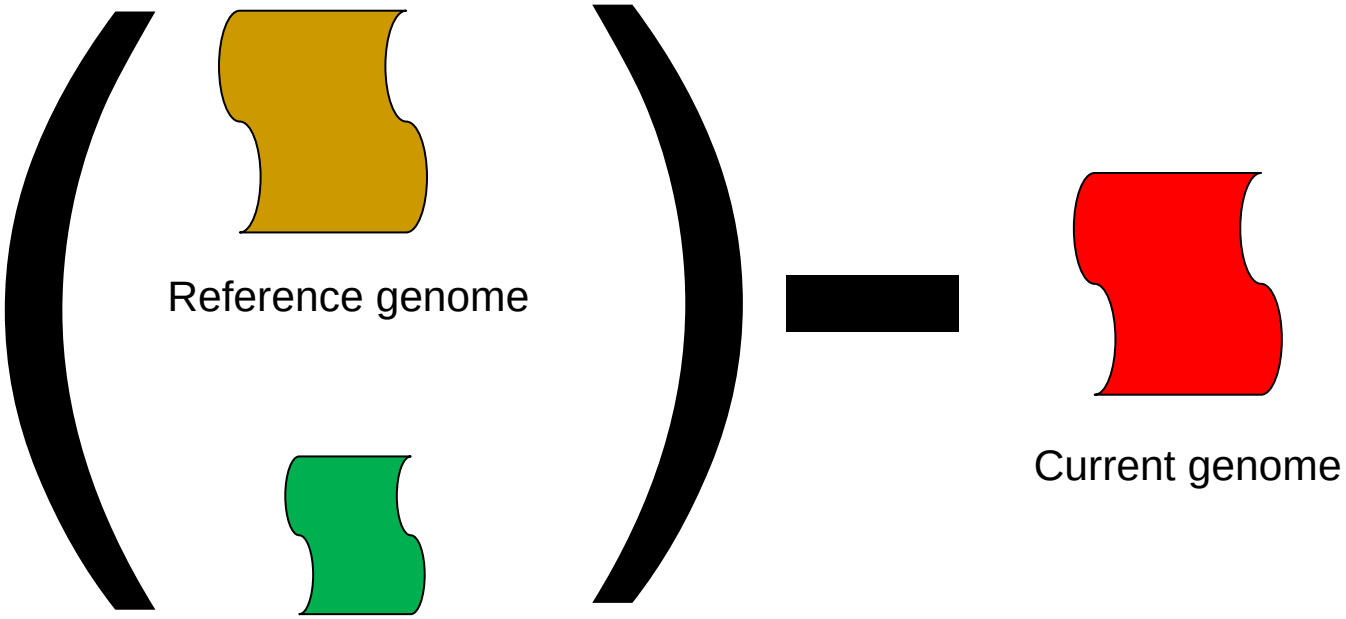
	1	2	3	4	8	18
Fibonacci	11	011	0011	1011	000011	0001011
1-shifted Fib.	1	011	0011	00011	001011	01010011
3-shifted Fib.	001	010	011	100	00011	000001011

Recent approaches

- **DNAPack**
  - uses dynamic programming for selection of segments (copied, reversed and/or modified)
  - $O(n^3)$ still too slow, hence heuristics used
- **XM**
  - use of both statistical properties and repetition within sequences
  - a panel of experts is maintained to estimate the probability distribution of the next symbol
  - expert probabilities are combined to obtain the final distribution, which is the used in arithmetic encoding

Recent approaches

- **DNAZip**
  - DNA sequence compression using a reference genome



SNP - single nucleotide polymorphism
List of places where similar genomes differ

Recent approaches

- **DNAZip**
  - DNA sequence compression using a reference genome
  - James Watson genome => 4101 kB
  - „Human genomes as email attachments“

Conditional compression

- Given sequence z, compress a sequence w relative to z
- Let $Compress(w \mid z)$ denote the length of the compression of w, given z. Similarly, let $Compress(w) = Compress(w \mid \varepsilon)$, where ε denotes the empty word. [*Compress* is not the unix compression program `compress`.]
- In general, $Compress(w \mid z) \neq Compress(z \mid w)$.
- For example, the *Biocompress-2* program produces:
  - $CompressRatio(brucella \mid rochalima) = 55.95\%$, and
  - $CompressRatio(rochalima \mid brucella) = 34.56\%$.
- If $z = w$, then $Compress(w \mid z)$ is very small.
- If z and w are completely independent, then

$$Compress(w \mid z) \approx Compress(w)$$

Evolutionary distance

- How to define evolutionary distance between strings based on conditional compression?
- E.g. $$D(w,z) = \frac{Compress(w \mid z) + Compress(z \mid w)}{2}$$

  is used in literature, but it has no good reason.
- We need a symmetric measure!
- $\Rightarrow$ we can use **Kolmogorov complexity**

Kolmogorov complexity

- Let $K(w \mid z)$ denote the Kolmogorov complexity of string w, given z. Informally, this is the length of the shortest program that outputs w given z. Similarly, set

$$K(w) = K(w \mid \varepsilon).$$

- The following result is due to Kolmogorov and Levin:
- **Theorem**: Within an additive logarithmic factor,

$$K(w \mid z) + K(z) = K(z \mid w) + K(w).$$

- This implies

$$K(w) - K(w \mid z) = K(z) - K(z \mid w).$$

- Normalizing by the sequence length we obtain the following symmetric map – "relatedness":

$$R(w,z) = \frac{K(w) - K(w \mid z)}{K(wz)} = \frac{K(z) - K(z \mid w)}{K(wz)}$$

What is the range of $R(w,z)$?

A symmetric measure of similarity

$$R(w,z) = \frac{K(w) - K(w \mid z)}{K(wz)} = \frac{K(z) - K(z \mid w)}{K(wz)}$$

- A distance between two sequences w and z : $D(w,z) = 1 - R(w,z)$
- Unfortunately, $K(w)$ and $K(w \mid z)$ are not computable!
- Approximation:
  - $K(w)$:= $Compress(w)$:= $|\ GenCompress(w)\ |$
  - $K(w \mid z)$:= $Compress(w \mid z)$:= $Compress(zw) - Compress(z)$ = $|\ GenCompress(zw)\ | - |\ GenCompress(z)\ |$
  - $$R(w,z) = \frac{Compress(w) - (Compress(zw) - Compress(z))}{Compress(wz)} =$$

$$\frac{Compress(w) + Compress(z) - Compress(zw)}{Compress(wz)}$$

Application to genomic sequences

Sequences*	*H. butylicus*	*H. gomorrense*	*A. urina*	*M. glauca*	*R. globiformis*	*L. sp. Nakagiri*	*U. crescens*
H. butylicus 1277†		2436‡ 2.53%§	2527 0.83%	2572 -0.02%	2534 0.68%	2546 0.43%	2572 -0.02%
H. gomorrense 1437	2779 2.10%		2892 -0.02%	2892 -0.02%	2869 0.38%	2892 -0.02%	2892 -0.02%
A. urina 1382	2738 0.81%	2782 -0.02%		2310 9.41%	2305 9.06%	2761 0.33%	2782 -0.02%
M. glauca 1339	2696 -0.02%	2696 -0.02%	2223 9.43%		2095 11.90%	2696 -0.02%	2696 -0.02%
R. globiformis 1465	2909 0.69%	2925 0.38%	2475 8.99%	2341 12.02%		2948 -0.02%	2948 -0.02%
L. sp. Nakagiri 1621	3241 0.33%	3261 -0.02%	3241 0.32%	3261 -0.02%	3261 -0.02%		2567 11.02%
U. crescens 1853	3725 -0.02%	3725 -0.02%	3725 -0.02%	3725 -0.02%	3725 -0.02%	3038 10.90%	

Table 4: Relatedness $R(u, v)$ between all pairs u, v.

*The sequences in the first column and first row are u and v, respectively.

†This is the length (# bases) of the input sequence.

‡This is the length (# bits) of conditionally compressed file between two u and v.

§This is $R(u, v)$.

Application to genomic sequences

Sequences*	*H. butylicus*	*H. gomorrense*	*A. urina*	*M. glauca*	*R. globiformis*	*L. sp. Nakagiri*	*U. crescens*
H. butylicus 1277†		2436‡ 2.53%§	2527 0.83%	2572 -0.02%	2534 0.68%	2546 0.43%	2572 -0.02%
H. gomorrense 1437	2779 2.10%		2892 -0.02%	2892 -0.02%	2869 0.38%	2892 -0.02%	2892 -0.02%
A. urina 1382	2738 0.81%	2782 -0.02%		2310 9.41%	2305 9.06%	2761 0.33%	2782 -0.02%
M. glauca 1339	2696 -0.02%	2696 -0.02%	2223 9.43%		2095 11.90%	2696 -0.02%	2696 -0.02%
R. globiformis 1465	2909 0.69%	2925 0.38%	2475 8.99%	2341 12.02%		2948 -0.02%	2948 -0.02%
L. sp. Nakagiri 1621	3241 0.33%	3261 -0.02%	3241 0.32%	3261 -0.02%	3261 -0.02%		2567 11.02%
U. crescens 1853	3725 -0.02%	3725 -0.02%	3725 -0.02%	3725 -0.02%	3725 -0.02%	3038 10.90%