A Compression Algorithm for DNA Sequences and Its Applications in Genome Comparison

Xin Chen [1] xchen@cs.cityu.edu.hk

Sam Kwong [2] cssamk@cityu.edu.hk

Ming Li [3] mli@wh.math.uwaterloo.ca

[1] Department of Computer Science, City University of Hong Kong, Hong Kong, China.
Permanent address: School of Mathematical Sciences, Peking University, China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[3] Department of Computer Science, University of Waterloo, ONT, N2L 3G1, Canada

Abstract

We present a lossless compression algorithm, *GenCompress*, for genetic sequences, based on searching for approximate repeats. Our algorithm achieves the best compression ratios for benchmark DNA sequences. Significantly better compression results show that the approximate repeats are one of the main hidden regularities in DNA sequences.

We then describe a theory of measuring the relatedness between two DNA sequences. Using our algorithm, we present strong experimental support for this theory, and demonstrate its application in comparing genomes and constructing evolutionary trees.

1 Introduction

With more and more complete genomes of prokaryotes and eukaryotes becoming available and the completion of human genome project in the horizon, fundamental questions regarding the characteristics of these sequences arise. In this paper, we study one such basic question: the compressibility of DNA sequences.

Life represents order. It is not chaotic or random [12]. Thus, we expect the DNA sequences that encode Life to be nonrandom. In other words, they should be very compressible. There are also strong biological evidences that support this claim: it is well-known that DNA sequences, especially in higher eukaryotes, contain many (approximate) tandem repeats; it is also well-known that many essential genes (like rRNAs) have many copies; it is believed that there are only about a thousand basic protein folding patterns; it also has been conjectured that genes duplicate themselves sometimes for evolutionary or simply for "selfish" purposes. All these give more concrete support that the DNA sequences should be reasonably compressible. However, such regularities are often blurred by random mutation, translocation, cross-over, and reversal events, as well as sequencing errors.

It is well recognized that the compression of DNA sequences is a very difficult task [5, 7, 16, 10]. The DNA sequences only consist of 4 nucleotide bases $\{a, c, g, t\}$, 2 bits are enough to store each base. However, if one applies standard compression software such as the Unix "compress" and "compact" or the MS-DOS archive programs "pkzip" and "arj", they all *expand* the file with more than 2 bits per base, as shown in Table 1, although all these compression software are universal compression algorithms. These software are designed for text compression [2], while the regularities in DNA sequences are much subtler.

It is our purpose to study such subtleties in DNA sequences. In this paper, we will present a DNA compression algorithm, *GenCompress*, based on approximate matching that gives the best compression results on standard benchmark DNA sequences.

One may treat compressibility study as the ultimate generalization of the simpler (and fruitful) biological studies such as G-C contents of various species. We are acquiring more and more complete genomes from the 5 megabase long *E. coli* to the 97 megabase long *C. elegans*. The 3 billion bases of *H. sapiens* will also be available soon. More sophisticated studies on these sequences will give us deeper understanding about the nature of these sequences. Different regions on a genome, different genes, different species may have different compression ratios. Such difference may imply, for example, different mutation rates in different genes [10].

In this paper, we study another interesting application of compressibility: to measure the "relatedness" between two DNA sequences or two genomes. We will present an elegant theory defining a symmetric distance between two sequences, and apply our compression algorithm to implement and support this theory. Our new approach enables us for the first time to quickly construct correct (consistent to the known phylogenies) trees from complete genomes.

The paper is organized as follows. In the next section, we discuss related work. In Section 3, we will present the design rationale of *GenCompress* based on approximate matching. We will present *GenCompress* in Section 4. Details of the algorithm will be given in that section as well. Experimental results will be given in Section 5 and we compare our results with the two most effective compression algorithms for DNA sequences: *Biocompress-2* and *Cfact*. Our algorithm provides significantly better results using standard benchmark DNA sequences in both cases. In Section 6, we describe a theoretically well-founded measure between two DNA sequences. We use our algorithm to compute a heuristic approximation to such a measure and demonstrate its application by constructing phylogenetic trees from it.

In this paper, if not otherwise mentioned, we will use lower case letters u, v, w, x, y to denote finite strings over the alphabet $\{a, c, g, t\}$. $|u|$ denotes the length of u, the number of characters in u. u_i is the i-th character of u. $u_{i:j}$ is the substring of u from position i to position j. The first character of u is u_0. Thus $u = u_{0:|u|-1}$. We use ϵ to denote empty string and $|\epsilon| = 0$.

2 Related Work

Grumbach and Tahi [7, 8] proposed two lossless compression algorithms for DNA sequences, namely *Biocompress* and *Biocompress-2*, in the spirit of Ziv and Lempel data compression method [21]. *Biocompress-2* detects exact repeats and complementary palindromes located earlier in the target sequence, and then encodes them by repeat length and the position of a previous repeat occurrence. In addition, they also use arithmetic coding of order 2 if no significant repetition is found. In fact, the difference between *Biocompress* and *Biocompress-2* is the addition of order-2 arithmetic coding.

É. Rivals *et al.* [16] give another compression algorithm *Cfact*, which searches the longest exact matching repeat using suffix tree data structure in an entire sequence. The idea of *Cfact* is basically the same as *Biocompress-2* except that *Cfact* is a two-pass algorithm. It builds the suffix tree in the first pass. In the encoding phase, the repetitions are coded with guaranteed gain; otherwise, two-bit per base encoding will be used. This is similar to the codeword encoding condition in *Biocompress-2* except that the order-2 arithmetic coding is not used in *Cfact*. É. Rivals *et al.* [17] also designed a compression algorithm as a tool to detect the approximate tandem repeats in DNA sequences.

Sadeh [18] has proposed lossy data compression schemes based on approximate string matching and proved some asymptotic properties with respect to ergodic stationary sources. However, we are *not* interested in lossy compression.

The lossless compression algorithm *GenCompress* in this paper achieves significantly higher compression ratios than both *Biocompress-2* and *Cfact*. Such improvement is vital to our applications.

In the second part of the paper we define a measure of "relatedness" between two DNA sequences. Then we apply this measure to construct evolutionary trees. Different approaches have already been proposed to define such a distance $d(x, y)$ in [7, 20, 19, 4]. [7] proposed to use the conditional compression. Using the ideas of Kolmogorov complexity (see [12]), Varre, Delahaye, and Rivals [20] defined "transformation distance". Essentially, this can be regarded as conditional compression using biologically related operations. While very attractive, both of these measures are not symmetric, and hence cannot be used in general. Authors of [20] also realized this problem and proposed to use $(d(x, y) + d(y, x))/2$. Obviously such a definition is not well-founded. Situation in fact can be salvaged by using a deep theorem proved in [3] (see Theorem 8.3.1 in [12]) and that leads to the original definition of “information distance”. However, information distance [3, 12], while symmetric, is also not a right measure in this case. On the other hand, biologists in [19, 4] proposed to use more involved and laborious methods like counting the number of shared genes in two genomes or comparing the ordering of the genes. We will introduce an elegant theory, a symmetric measure of relatedness between two DNA sequences. We then justify it by applying this theory to biological data (genes and genomes) using *GenCompress* as a tool.

3 Encoding edit operations

We consider three standard edit operations in our approximate matching algorithm. These are:

1. *Replace.* This operation is expressed as $(R, p, char)$ which means replacing the character at position p by character $char$.

2. *Insert.* This operation is expressed as $(I, p, char)$, meaning inserting character $char$ at p.

3. *Delete.* This operation is written as (D, p), meaning deleting the character at position p.

Let C denote “copy”, then the following are two ways to convert the string “gaccttca” to “gaccgtca” via different edit operation sequences:

C	C	C	C	R	C	C	C	
g	a	c	c	g	t	c	a	or
g	a	c	c	t	t	c	a	

C	C	C	C	I	C	D	C	C
g	a	c	c	g	t		c	a
g	a	c	c		t	t	c	a

The first involves one replacement operation. The second involves one insert and one delete. It can be easily seen that there are infinitely many edit sequences to transform one string to another.

A list of edit operations that transform a string u to another string v is called an *Edit Transcription* of the two strings [9]. This will be represented by an edit operation sequence $\lambda(u, v)$ that orderly lists the edit operations. For example, the edit operation sequence of the first edit transcription in the above example is $\lambda(\text{“gaccgtca”}, \text{“gaccttca”}) = (R, 4, g)$; and for the second edit transcription, $\lambda(\text{“gaccgtca”}, \text{“gaccttca”}) = (I, 4, g), (D, 6)$.

If we know the string u and an edit operation sequence $\lambda(u, v)$ from u to v, then the string v can be constructed correctly using λ. There are many ways to encode one string given another. Using the above example, we describe four ways to encode “gaccgtca” using string “gaccttca”.

1. Two bits encoding method. In this case, we can simply use two bits to encode each character, i.e. 00 for a, 01 for c, 10 for g, 11 for t. Thus “10 00 01 01 10 11 01 00” encodes ”gaccgtca”. It needs 16 bits in total.

2. Exact matching method. We can use (repeat position, repeat length) to represent an exact repeat. This way, for example, if we use three bits to encode an integer, two bits to encode a character, and use one bit to indicate if the next part is a pair (indicating an exact repeat) or a plain character, then the string “gaccgtca” can be encoded as $\{(0, 4), g, (5, 3)\}$, relative to “gaccttca”. Thus, a 17 bits binary string “0 000 100 1 10 0 101 011” is required to encode the $\{(0, 4), g, (5, 3)\}$.

3. Approximate matching method. In this case, the string “gaccgtca” can be encoded as $\{(0, 8), (R, 4, g)\}$, or “0 000 111 1 00 100 10” in binary, with R encoded by 00, I encoded by 01, and D encoded by 11, and 0/1 indicating whether the next item is a doubleton or triple. A total of 15 bits is needed.

4. For approximate matching method, if we use the edit operation sequence (I,4,g),(D,6). Then the string "gaccgtca" can be encoded as $\{(0,8),(I,4,g),(D,6)\}$, or "0 000 111 1 01 100 10 1 10 110", in total 21 bits.

From the above examples, we see that the approximate matching method in the third case has the minimal number of bits to encode the string "gaccgtca" with reference to "gaccttca".

4 GenCompress: an algorithm based on approximate matching

Lempel and Ziv proposed two algorithms in [21, 11] to compress universal data sequences. These are dictionary based compression algorithms that rely on exact repeats. The Lempel-Ziv algorithms can be viewed as having two components: the first component is to parse the input data sequence into variable-length strings based on the history of the dictionary. The second component is to replace the variable-length prefix by a proper binary codeword—concatenation of these codewords yields the encoder's output sequence in response to the input data sequence. We follow the same framework and generalize it to approximate matching for DNA sequences.

GenCompress is a one-pass algorithm. It proceeds as follows: For input w, assume that a part of it, say v, has already been compressed, and the remaining part is u, i.e. $w = vu$. *GenCompress* finds an "optimal prefix" of u such that it approximately matches some substring in v so that this prefix of u can be encoded economically. After outputting the code of this prefix, remove the prefix from u, and append it to the suffix of v. Continue the process till $u = \epsilon$.

4.1 Condition C, the compression gain function, and the optimal prefix

We adopt the following constraint in *GenCompress* to limit the search. If the number of edit operations located in any substring of length k in the prefix s of u for an edit operation sequence $\lambda(s,t)$ is not larger than a threshold value b, we say that $\lambda(s,t)$ satisfies the *condition* $C = (k,b)$ for compression. In *GenCompress*, we only search for approximate matches that satisfies condition C. This way we limit our search space. Experiments show that setting C to $(k,b) = (12,3)$ gives good results.

We also need to define a *compression gain function* G in order to evaluate if a particular approximate repeat provides profit in the encoding. G is defined to be:

$$G(s,t,\lambda) = \max\{2|s| - |(|s|,i)| - w_\lambda * |\lambda(s,t)| - c, 0\},$$

where

- s is a prefix of u,
- t is a substring appear at position i in v,
- $2|s|$ is the number of bits we otherwise use: 2 bits per base, $|(|s|,i)|$ is the encoding size of $(|s|,i)$,
- w_λ is the weighted value of encoding an edit operation,
- $|\lambda(s,t)|$ is the number of edit operations in $\lambda(s,t)$,
- and c is the overhead proportional to the size of control bits.

Let input $w = vu$ where v has already been processed. Given $G(s,t,\lambda)$ and C, the *optimal prefix* is a prefix s of u such that $G(s,t,\lambda)$ is maximized over all λ and t such that t is a substring of v and λ is an edit transcription from t to s satisfying condition C.

4.2 The parsing procedure based on approximate matching

The following parsing procedure finds the optimal prefix.

1. Initialize the condition C for compression and the compression gain function G; and set $u = w$ to the complete input string and $v = \epsilon$ to be the empty database.

2. Search for the optimal prefix s of u, with approximate match t in the database v.

3. Encode the repeat representation $|(|s|, p)|$, where p is the position of t, and the shortest edit operation sequence $\lambda(s,t)$. Output the code.

4. Remove prefix s from u, and append s to the database v. If $u \neq \epsilon$, goto (2); else exit.

If the condition C is set such that no edit operation is allowed in the search region of the DNA sequence, then we have an exact match algorithm. In addition, if the compression gain function G is set to be $|s|$, i.e. the number of exact matching characters, the above parsing procedure will be identical to the Lempel–Ziv parsing procedure.

4.3 The encoding procedure

Using the compression gain function G defined above, there are a lot of individual characters being emitted because $G(s,t,\lambda) = 0$ for all s, t, λ is often the case. That is, there does not exist an approximate match in the database for any prefix s of u that could help saving bits.

The encoding procedure is designed as follows [7]:

1. Initialize a data buffer to nil;

2. If $G(s,t,\lambda) = 0$, then append the first character of u into the data buffer;

3. If $G(s,t,\lambda) > 0$, then

   (a) If the data buffer is not empty, add a flag to indicate there is a zone of individual characters and encode them with the order-2 Arithmetic encoding [15, 2]; reset the data buffer to nil;

   (b) Add another flag to indicate there is an approximate match and then encode the approximate match $(|s|, i)$ by the Fibonacci representation (a self-delimiting code) of positive integers [1] and its edit operation sequence $\lambda(s,t)$, where i is the position t appears in v.

GenCompress also detects the approximate complemented palindrome in the DNA sequences. The method is similar.

4.4 Implementing the optimal prefix search

The goal of *GenCompress* is to obtain sharp compression ratio for DNA sequences. The time complexity consideration is secondary. However, since the genomes we are compressing are extremely long, often over several million base pairs, simple minded exhaustive search for optimal prefixes even with condition C takes too much time. To help reduce the search space, we first make a few observations.

Lemma 4.1. *An optimal prefix $u_{0:l}$ with $G(u,v,\lambda) > 0$ always ends right before a mismatch.*

Proof. Assume that $u_{0:l}$ matches with $v_{i:j}$ by an optimal edit transcription sequence λ. If the next character u_{l+1} matches with v_{j+1}, then we can include u_{l+1} into the prefix to form prefix $u_{0,l+1}$. So,

$$G(u_{0:l}, v_{i:j}, \lambda) - G(u_{0:l+1}, v_{i:j+1}, \lambda') = |Fibo(l+1)| - |Fibo(l)| - 2$$

where λ' is the natural extension of λ with one extra match, $Fibo(l)$ denotes the Fibonacci representation code [1] of integer l and $|Fibo(l)|$ is its bit size. We know from [16] that, for any positive integer i:

$$\left\lceil \frac{\log(i\sqrt{5}-1)}{\log(\frac{1+\sqrt{5}}{2})} \right\rceil \leq |Fibo(i)| - 1 \leq \left\lceil \frac{\log(i\sqrt{5}+1)}{\log(\frac{1+\sqrt{5}}{2})} \right\rceil.$$

Thus $|Fibo(l+1)| - |Fibo(l)| \leq 1$. Hence, we get $G(u_{0:l}, v_{i:j}, \lambda) < G(u_{0:l+1}, v_{i:j+1}, \lambda')$. This contradicts the fact that $u_{0:l}$ is an optimal prefix. ◇

In the search algorithm, we will first search for an exact repeat of a fixed small length l as the prefix of each approximate repeat. This limits the search space. To implement it, we assign each position in the sequence an integer value that is determined by the substring of length l that starts from this position. Thus, we can search for repeats among those positions in the database that have integer value equal to the current position. This method is easier than the one using suffix tree, which needs two suffix trees for direct and palindrome repeats, respectively.

Also observe that a delete followed by an insert is simply a replace operation. We therefore limit the search by not considering this kind of combinations. The following lemma states another simple observation whose proof is simple and is left to the reader.

Lemma 4.2. *Let λ be the optimal edit operation sequence from x to y. If y_i is copied from x_j in λ when converting y to x, then $\lambda(x_{0:i}, y_{0:j})$ is also the optimal edit operation sequence among all the edit operation sequences from $x_{0:i}$ to $y_{0:j}$.*

Combining these observations and more, our algorithm for optimal prefix search is outlined as follows. The search for palindromes is implemented similarly.

1. Let $w = vu$ where v has already been compressed.

2. Find all occurrences of $u_{0:l}$ in v, for some small l. For each such occurrence, try to extend it, allowing approximate match, limited by the above observations and condition C. Return the one with largest G value.

5 Experimental Results

We tested *GenCompress* on standard benchmark data used in [7]. These standard sequences, (available at [14]) come from a variety of sources and include the complete genomes of two mitochondria: MPOMTCG, PANMTPACGA (also called MIPACGA); two chloroplasts: CHNTXX and CHMPXX (also called MPOCPCG); five sequences from humans: HUMGHCSA, HUMHBB, HUMHDABCD, HUMDYSTROP, HUMHPRTB; and finally the complete genome from the two viruses: VACCG and HEHCMVCG (also called HS5HCMVCG).

We implemented two versions of *GenCompress*: *GenCompress-1* searches for approximate repeats with replacement operations only (i.e. Hamming distance); and *GenCompress-2* searches for approximate repeats based on the edit operations described in Section 3. The definition of the compression ratio is the same as in [7], i.e. $1 - (|O|/2|I|)$, where $|I|$ is number of bases in the input DNA sequence and $|O|$ is the length (number of bits) of the output sequence. The compression ratios of *GenCompress*, as well as those of it Biocompress-2 and some other compression algorithms, are presented in Table 1. We have also compared *GenCompress* with *Cfact* from [16] in Table 2, using the data from [16]. It is amusing to see that although *Cfact* looks for the best matches globally whereas our *GenCompress* only searches for the best approximate match from the current prefix to the part of the text seen so far, *GenCompress* has much better compression ratio than *Cfact*. From these experiments, we conclude that approximate matching plays a key role in finding similarities or regularities in DNA sequences.

GenCompress is able to detect approximate matches of any edit distance, including exact repeats. Fig. 1 shows the repeat pattern for sequences HUMGHCSA and VACCG. All these repeats can be positively compressed with a proper compression gain function. When there are not enough approximate repeats in the data sequence as shown in Fig. 1 for VACCG, *GenCompress* fails to achieve higher compression ratio than *Biocompress-2*.

In conclusion, the compression results of *GenCompress* for DNA sequences indicate that our method based on approximate matching is more effective than others. *GenCompress* is able to detect more regularities and achieve best compression results by using this observation.

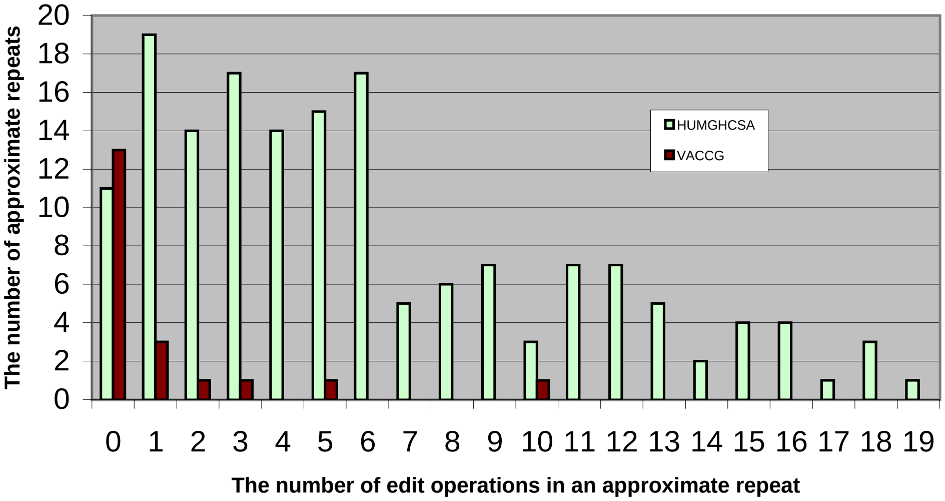

Figure 1: The number of profitable approximate repeats, found by *GenCompress*, in sequences HUMGHCSA and VACCG.

Table 1: Compression Ratios. The last column is the improvement of *GenCompress-2* over *Biocompress-2*.

sequence	size	*compress*	*arith-2*	*Biocompress-2*	*GenCompress-1*	*GenCompress-2*	improvement
MTPACGA	100314	-5.81%	6.37%	6.24%	6.88%	6.88%	10.26%
MPOMTCG	186608	-10.11%	1.72%	3.11%	4.71%	4.71%	51.45%
CHNTXX	155844	-9.36%	3.31%	19.14%	19.27%	19.27%	0.68%
CHMPXX	121024	-3.73%	8.17%	15.76%	16.38%	16.35%	3.74%
HUMGHCSA	66495	-9.68%	3.11%	34.63%	44.99%	44.76%	29.25%
HUMHBB	73323	-9.73%	4.08%	6.16%	8.98%	9.04%	46.75%
HUMHDABCD	58864	-11.48%	2.87%	6.15%	9.27%	9.04%	46.99%
HUMDYSTROP	38770	-11.66%	3.80%	3.69%	3.88%	3.87%	4.88%
HUMHPRTB	56737	-10.12%	3.56%	4.67%	7.67%	7.67%	64.24%
VACCG	191737	-8.37%	5.10%	11.93%	11.94%	11.93%	0.00%
HEHCMVCG	229354	-10.65%	1.76%	7.60%	7.65%	7.65%	0.66%

Table 2: Compression Ratios. The last column is the improvement of *GenCompress-2* over *Cfact*.

sequence	size	*LZW 15*	*arith-2*	*Cfact*	*GenCompress-1*	*GenCompress-2*	improvement
atatsgs	9647	-11.83%	2.44%	10.75%	16.58%	16.25%	51.16%
atef1a23	6022	-14.85%	0.30%	20.77%	22.96%	23.10%	11.22%
atrdnaf	10014	-15.00%	-0.46%	9.30%	10.59%	10.63%	14.30%
atrdnai	5287	-11.97%	0.28%	26.60%	29.00%	29.50%	10.90%
hsg6pdgen	52173	-8.42%	3.14%	3.59%	9.85%	9.44%	162.95%
xlxfg512	19338	-4.19%	3.86%	25.49%	31.67%	31.07%	21.89%
mmzp3g	10833	-12.18%	2.37%	4.44%	7.40%	7.22%	62.61%
celk07e12	58949	-5.41%	4.38%	14.33%	19.14%	18.73%	30.70%

6 Relatedness Between Two DNA Sequences

In many methods (quartet methods, neighbor joining, UPGMA, Fitch's least squares) of constructing phylogenetic trees, the first step is to evaluate the "distance" between pairs of DNA sequences. Sequences that are "close" to each other are required to be "close" to each other on the evolutionary tree. Distances such as minimum alignment score works for closely related genes. They fail to handle simple changes like reversal and translocation. They do not apply to more than one gene and noncoding regions. A tree constructed using one gene is often different from that using another. Measures such as genome rearrangement distance, reversal distance studied in the CS community and the number of shared genes [19] or gene order [4] studied recently in the biology community are examples of other specialized distances. These distances are not expected to be general distance measures.

How do we define a proper measure between a pair of DNA sequences? Authors in [7] and [20] propose to use conditional compression to evaluate the distance or "relatedness" between two DNA sequences. They further demonstrated how to use compression to construct an evolutionary tree. Although this looks like an attractive proposal, it has a fatal problem: the distance is *not symmetric*! Let us use the notation

$$Compress(u|v)$$

to mean the length of compressed result of u given v for free (as database), and we write $Compress(u)$ for $Compress(u|\epsilon)$, and

$$CompressRatio(u|v)$$

to mean the compression ratio of u given v for free. Then, theoretically [12],

$$Compress(u|v) \neq Compress(v|u), \text{ and}$$

$$CompressRatio(u|v) \neq CompressRatio(v|u).$$

This was even apparent in the experiments performed in [7] where the following ratios were obtained using *Biocompress-2*:

$$CompressRatio(\text{brucella}|\text{rochalima}) = 55.95\%, \text{ and}$$

$$CompressRatio(\text{rochalima}|\text{brucella}) = 34.56\%.$$

One wonders what if there is another sequence u such that

$$CompressRatio(u|\text{brucella}) = 45\%, \text{ and}$$

$$CompressRatio(\text{brucella}|u) = 44\%?$$

Then, is brucella closer to u or closer to rochalima? Authors in [20] have realized this problem and proposed to use something like $(Compress(u|v) + Compress(v|u))/2$. But neither does this provide the right theory, nor does it solve the practical problem.

We now formulate a symmetric distance using a beautiful theorem from Kolmogorov complexity. We will do this only informally. For formal definitions and proofs, we refer the reader to [12]. Let $K(u|v)$ be the conditional Kolmogorov complexity of string u condition on string v, that is, $K(u|v)$ is the length of the shortest program that outputs u given v as input. $K(u) = K(u|\epsilon)$ is the unconditional Kolmogorov complexity of u, or the length of the program that outputs u, on empty input. The following theorem (due to Kolmogorov and Levin) is well-known, see Theorem 3.9.1 in [12].

Theorem 6.1. **(Symmetry of Information)** *Within an additive logarithmic factor,*

$$K(u|v) + K(v) = K(v|u) + K(u). \tag{6.1}$$

Rearranging Equation 6.1, we obtain

$$K(u) - K(u|v) = K(v) - K(v|u).$$

$K(u) - K(u|v)$ is *information* in v about u, and $K(v) - K(v|u)$ is *information* in u about v. Theorem 6.1 says the information in v about u is equal to the *information* in u about v. This measure is *symmetric* and it measures precisely what we want to measure: the mutual information or relatedness of two sequences. Taking the sequence length into consideration, we propose to use

$$R(u, v) = \frac{K(u) - K(u|v)}{K(uv)} = \frac{K(v) - K(v|u)}{K(uv)}$$

to measure their relatedness and $1 - R(u, v)$ as their distance. Initially, we used $|u| + |v|$, instead of $K(uv)$ as the denominator. But such a definition is problematic when measuring relatedness among u, v and uu. Now we can also explain why the proposal of using $K(u|v)$ is not correct but worked nicely in some cases [20]: this is because it ignores the self-compressibility of u. When u and v are roughly equally long and are both not very compressible (often this is the case), then $K(u|v)$ can be used to replace $K(u) - K(u|v)$. Information distance, defined as $\max\{K(u|v), K(v|u)\}$ ([3, 12]), fails for several other reasons.

It has not escaped our notice that the distance measure we have postulated can be immediately used to construct evolutionary trees from DNA sequences that cannot be aligned, such as complete genomes. Let us use $Compress(u)$ to heuristically approximate $K(u)$ and use $Compress(u|v)$ to heuristically approximate $K(u|v)$. We have converted our *GenCompress* program into the conditional version and performed small scale preliminary experiments. We have randomly taken 16S rRNA (and 18S rRNA for Eukaryotes) genes from the GenBank, [14], for the following species:

- Archaebacteria: *H. butylicus*, *Halobaculum gomorrense*
- Eubacteria: *Aerococcus urina*, *M. glauca* strain B1448-1, *Rhodopila globiformis*
- Eukaryotes: *Urosporidium crescens*, *Labyrinthula sp. Nakagiri.* These are 18S rRNA genes in eukaryotes, corresponding to 16S rRNA in prokaryotes.

The $R(u, v)$ for each pair of these genes is calculated in Table 4. Observe the symmetry in the table: $R(u, v) \approx R(v, u)$, as predicted by our theory. From these distances, we obtain directly the evolutionary tree in Fig. 2 (left tree), which is the phylogeny given in the GenBank.

We have further experimented on two sequences *atrdnai* and *atrdnaf* from [16]. They have very different lengths: $|atrdnai| = 5287$ bases; $|atrdnaf| = 10014$ bases. But,

$$Compress(atrdnai) - Compress(atrdnai|atrdnaf) = 6887 \quad bits$$

$$Compress(atrdnaf) - Compress(atrdnaf|atrdnai) = 6937 \quad bits, \quad \text{and}$$

$$R(atrdnai, atrdnaf) = 37.29\%; \qquad R(atrdnaf, atrdnai) = 37.56\%$$

showing our heuristics is very close to the theoretical prediction.

We expect this approach to be an alternative to simple alignment distance when comparing *not closely related sequences* and genomes. To further demonstrate the power of this method, we have performed another small scale experiment with seven complete genomes from [14]:

- Archaea Bacteria: *Archaeoglobus fulgidus* (u1), *Pyrococcus abyssi* (u2), *Pyrococcus horikoshii* OT3 (u3)
- Bacteria: *Escherichia coli* K-12 MG1655 (u4), *Haemophilus influenzae* Rd (u5), *Helicobacter pylori* 26695 (u6); *Helicobacter pylori*, strain J99 (u7).

Table 3: $Compress(u)$ (# bits) of sequences u .

Sequences u	H. butylicus	H. gomorrense	A. urina	M. glauca	R. globiformis	L. sp. Nakagiri	U. crescens
Compress(u)	2571	2891	2781	2695	2947	3260	3724

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 $R(u, v)$.

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%	

The distance matrix and the derived phylogeny are given in Table 5 and Fig. 2 (right tree). Currently together with J. Badger, P. Kearney, H. Zhang, we are doing extensive studies on genome phylogeny using this method on the mitochondrial genomes and all the complete genomes in [14]. Preliminary results are very encouraging. Those results will be reported elsewhere.

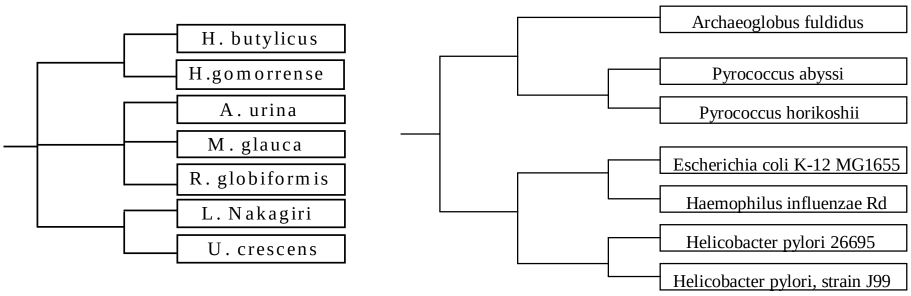

Figure 2: An rRNA tree and a genome tree.

Acknowledgments

This work was supported in part by City University of Hong Kong Grant No. 7000875, NSERC Research Grant OGP0046506, CITO, a CGAT grant, and the Steacie Fellowship. Ming Li's work was partly done while he was visiting City University of Hong Kong. We wish to thank Professor Qing-Yun Shi for her generous support, Fariza Tahi for providing the *Biocompress-2* program, Jonathan Badger, Kevin Lanctot, and Huaichun Wang for providing some data and very helpful comments. Jonathan provided references [19, 4].

Table 5: Relatedness $R(u, v)$ between all pairs u, v.

Sequence*	*u1*	*u2*	*u3*	*u4*	*u5*	*u6*	*u7*
u1 2178400†		4226821‡ 0.018326§	4226743 0.019550	4228299 -0.000548	4228411 -0.002399	4228356 -0.001765	4228392 -0.002259
u2 1765118	3443540 0.023072		3391432 0.797546	3445299 0.000089	3445242 0.000988	3445257 0.000812	3445264 0.000705
u3 1738505	3362288 0.023055	3310372 0.794383		3364086 -0.000391	3363996 0.000617	3364031 0.000109	3364045 -0.000109
u4 4639221	8920179 0.000373	8920362 -0.001084	8920294 -0.000537		8914204 0.048760	8919249 0.008160	8919224 0.008371
u5 1830138	3440205 0.000274	3440216 0.000145	3440229 -0.000044	3434165 0.049059		3439033 0.018303	3439068 0.017776
u6 1667867	3079174 -0.002217	3078992 0.000307	3079021 -0.000140	3077924 0.009068	3077935 0.016523		1226333 43.069863
u7 1643831	3075330 -0.001314	3075285 -0.000782	3075238 -0.000062	3074059 0.009796	3073952 0.019680	1219515 43.171044	

References

[1] Apostolico, A. and Fraenkel, A.S., Robust transmission of unbounded strings using Fibonacci representations, *IEEE Trans. Inform. Theory*, IT-33(2):238–245, 1987.

[2] Bell, T.C., Cleary, J.G., and Witten, I.H., *Text Compression*, Prentice Hall, 1990.

[3] Bennett, C.H., Gács, P., Li, M., Vitányi, P., and Zurek, W., Information Distance, *IEEE Trans. Inform. Theory*, 44(4):1407–1423, 1998. (Also in *STOC '93*.)

[4] Boore, J.L. and Brown, W.M., Big trees from little genomes: mitochondrial gene order as a phylogenetic tool, *Curr. Opin. Genet. Dev.*, 8(6):668–674, 1998.

[5] Curnow, R. and Kirkwood, T., Statistical analysis of deoxyribonucleic acid sequence data – a review, *J. Royal Statistical Society*, 152:199–220, 1989.

[6] Gardner, E.J., Sinnoms, M.J., and Snustad, D.P., *Principles of Genetics*, 8th edition, John Wiley & Sons, 1991.

[7] Grumbach, S. and Tahi, F., A new challenge for compression algorithms: genetic sequences, *J. Information Processing and Management*, 30(6):875-866, 1994.

[8] Grumbach, S. and Tahi, F., Compression of DNA sequences, *Proc. IEEE Symp. on Data Compression*, 340–350, 1993.

[9] Gusfield, D., *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[10] Lanctot, K., Li, M., and Yang, E.H., Estimating DNA sequence entropy, to appear in *SODA '2000*.

[11] Lempel, A. and Ziv, J., Compression of individual sequences via variable-rate coding, *IEEE Trans. Inform. Theory*, IT-24:530–536, 1978.

[12] Li, M. and Vitányi, P., *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, 2nd ed., 1997.

[13] Milosavljevic, A. and Jurka, J., Discovery by minimal length encoding: a case study in molecular evolution, *Machine Learning*, 12:69–87, 1993.

[14] National Center for Biotechnology Information, Entrez Nucleotide Query, `http://www.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=n_s`.

[15] Nelson, M., *The Data Compression Book*, M&T Publishing Inc., 1991.

[16] Rivals, É., Delahaye, J.-P., Dauchet, M., and Delgrange, O., A Guaranteed Compression Scheme for Repetitive DNA Sequences, LIFL Lille I University, technical report IT-285, 1995.

[17] Rivals, É., Delgrange, O., Delahaye, J.-P., Dauchet, M., Delorme, M.-O., Hénaut, A., and Ollivier, E., Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences, *CABIOS*, 13(2):131–136, 1997.

[18] Sadeh, I., Universal data compression algorithm based on approximate string matching, *Probability in the Engineering and Informational Sciences*, 10:465–486, 1996.

[19] Snel, B., Bork, P., and Huynen, M.A., Genome phylogeny based on gene content, *Nat. Genet.*, 21(1):108–110, 1999.

[20] Varre, J.-S., Delahaye, J.-P., and Rivals, É, The transformation distance: a dissimilarity measure based on movements of segments, *German Conference on Bioinformatics*, Koel, Germany, 1998.

[21] Ziv, J. and Lempel, A., A universal algorithm for sequential data compression, *IEEE Trans. Inform. Theory*, 23(3):337–343, 1977.