DNA.EXE: A Sequence Comparison between the Human Genome and Computer Code

Josiah Seaman

University of Colorado, Anschutz Medical Campus, Computational Bioscience Program, Aurora, CO 80045–0511, USA. Josiah.Seaman@ucdenver.edu

Abstract

This study presents evidence that executable computer programs and human genomes contain similar patterns of repetitive code. When viewed with sequence visualization tools, these similarities are both striking and pervasive. The primary similarities are listed in order of scale: (1) homopolymers, (2) tandem repeats, (3) distributed repeats, (4) isochores, (5) and entire chromosome/file organization. Most strikingly, data visualization reveals that executable codes regularly make extensive use of tandem repeats which exhibit similar visual patterns as seen in higher genomes. In biology these tandem repeat patterns are normally attributed to replication errors, insertions, deletions, and substitutions. Similarly, on a larger scale, executable codes display regions with different ratios of 1's and 0's which parallel the isochore patterns within chromosomes, caused by local variation in the number of A/T vs. G/C. Further, blocks of data are stored at the beginning or end of a file, while the primary instructions occupy the middle of a file. This creates the same organizational patterns observed in human chromosome arms, where repetitive sequences are grouped near the telomeres and centromeres.

I propose that these similarities can be explained by universal constraints in efficient information encoding and execution. The genome may be viewed as the executable program that encodes life. Given the evidence that computer programs and genomes use many of the same patterns of organization, despite having very different context, it should be informative to explore the ways in which knowledge of computer architecture can be applied to biology and vice versa.

Key words: computer code, alu, tandem repeats, junk DNA, small RNA, biological computer, retrotransposon, programming, cybernetics, data visualization, data analysis

Introduction

The study of the human DNA sequence has been dominated by the study of protein coding genes. These protein coding regions, called exons, constitute a mere 1.2% of the human genome [1]. Exons use a very simple code called the codon code that can be expressed in terms of a single table of 64 values. Without the key of the codon code, exons would appear to be meaningless nonsense to us. Thankfully, the codon code is a (relatively) straightforward and known entity and with it we can

predict the amino acid sequence of most genes in the nucleus. The codon code represents the first code in the human genome that we were able to decipher.

Those with even a passing understanding of human genetics should understand how incomplete this picture is. Exons using the codon code do not stand in isolation but are intertwined and dependent on numerous other genome elements which employ their own codes. Transcription of genes is regulated by promoters which use a transcription factor binding site code along with protein combinations. The newly formed pre-mRNA transcript contains elements called introns that use a third code, the splicing code, for determining how all of the exons are spliced together. All three of these codes are separate yet interdependent on each other to make the right protein product at the right time. Many other codes have been, and will continue to be discovered.

What is a code? A code is a precise mapping from a set of symbols to specified meanings, actions, and objects. We use codes for many purposes such as naming parts (e.g. A, B, and C) in an assembly manual. Human language itself is a type of code, though one that is much more elaborate and flexible than any other code. Cyphers used to conceal meaning for cryptography are sometimes called "codes" but cyphers are just one subset of codes. However, their use underscores a very important attribute of encoding: If one does not know the code in which something is written, it will appear to be meaningless nonsense.

In the previous century, the study of genomics was largely constricted to protein coding exons, which were already a formidable challenge to study. The other 98% of the human genome was dismissed as junk because it appeared to be meaningless nonsense [2, 3]. Slowly, exons' deadlock on genomics was loosened, first through gene expression analysis that showed the importance of promoters and enhancers, and then through the realization that alternative splicing was critical to understanding the complexity of the human body. The work of Barash *et al.* in 2009 was just the first step to cracking open the complexity of the splicing code [4]. There has been rapidly mounting evidence that various non-coding RNAs inside the cell serve useful functions and that there is a veritable zoo of RNA types. In 2007, the ENCODE project opened up the field by showing that over 90% of the human genome is transcribed [5, 6]. This forces us to conclude that either the cell wastes energy on extensive junk transcription, or, in keeping with the discovery of new RNA types, that the majority of the human genome is functional [7]. Transitioning from a 2–3% functional genome to >90% function is understandably unattractive to some, because it means that the majority of genetic research to date has only scratched the surface of all that the human genome actually encodes. An enormous task lies before us, as we endeavor to comprehend the many undiscovered functions of the multifaceted genome. To do this we need better tools and new approaches.

Methods

Skittle Genome Visualizer (Skittle for short) is a new sequence visualization tool suite [8]. This tool is especially sensitive for detection of any type of repeating pattern within sequence data. Although it was developed to analyze DNA sequences, it is very effective in analyzing repeat patterns in any type of sequence — including RNA, protein, written text, music, or computer code.

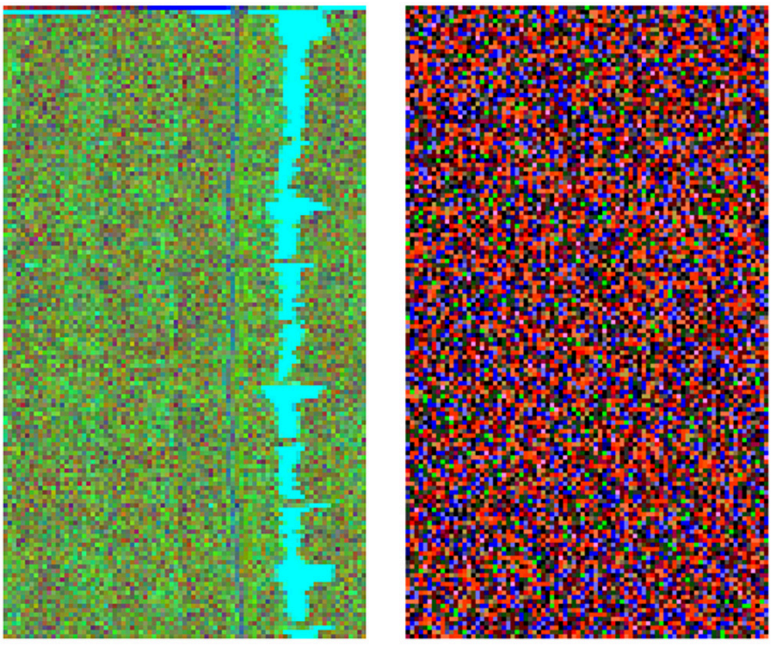
When DNA sequences from higher eukaryotes were examined using Skittle, extensive repetition was very clearly seen [8]. This might not seem surprising, as such repetition was previously known, and was presumed to be the result of numerous types of copying errors — generating a large amount of junk DNA.

During the course of browsing a number of chromosome sequence files using Skittle, a non-genome sequence file was opened accidentally. The program was directed to its own executable file: SkittleToo.exe. The same visualization that had been so successful in studying chromosome sequences was accidentally applied to computer machine code. Surprisingly, computer code revealed the same patterns and types of variations seen in the human genome. Yet none of these repeating patterns could be attributed to copying errors – every bit in the repeat patterns was there for a reason, and therefore the repeating patterns reflected the essential and inherent architecture of the computer code information system.

In order to make these visual comparisons easier, the executable computer code was converted to a base-4 symbol set of 'ACGT' (00 = A, 01 = C, 10 = G, 11 = T). In the figures, programs that have been converted to base-4 will have a .fa extension to indicate the change. For example, SkittleToo.exe becomes SkittleToo.fa when converted to the base-4 symbol set for comparison.

The main innovation in Skittle is to transform a sequence into an image by representing each letter with a color. This engages human visual recognition for structures and patterns instead of the target recognition that the brain uses when reading a sentence. In each of the figures (except Figure 2), the sequence is read from left to right starting at the upper right corner in the same way that one would read English text. Instead of displaying simple text, each letter is replaced with a colored square "pixel" that represents that letter. At Scale = 1 each pixel represents one letter or nucleotide. The visualization can "zoom out" by increasing the scale such that each pixel is the color average of multiple values (Figures 4B, 5, 6, 7). At Scale = 10 the color for one pixel is computed by taking the next 10 letters in the sequence, converting them into 10 colors and then averaging the colors together.

In addition to this, Skittle contains a suite of visualizations specialized for specific tasks (Figure 5, 7). For a more in depth explanation of the visualization methods and the pattern recognition algorithms used in this paper refer to "Skittle: a 2-Dimensional Genome Visualization Tool"[8].



(A) (B)

Fig. 1. A) MP3 recording of Beethoven's Moonlight Sonata. MP3 files consist of thousands of copies of a repeat monomer. In this example, each line has a homo-polymer of variable length. In general, data files are a single repeat of one format in contrast to computer programs which contain many different types of information. B) Text of Moby Dick. The Nucleotide Display essentially looks like colored static with no changing color bias. Similarly, the figure shows no repeats. English prose actually shows almost no tandem repetition. The only detectable tandem repeat in the entire text of Moby Dick is a short song about the sea that repeats the chorus 3 times.

Results

Negative Controls

This study examines significant patterns of sequence repetition within genomes and within executable computer code. To determine how these patterns are distributed in other information formats a whole variety of file types were examined. Among the file types visualized in Skittle were: exe, dll, cab, zip, png, bmp, jpg, tif, mp3, wav, cod, and txt. Human text (cod, txt), such as books, had almost no discernible patterns (Figure 1B). On the other hand, data files (png, bmp, jpg, tif, mp3, wav) appeared as a single uniform tandem repeat (Figure 1A). Compressed regions of files (cab, zip) lack most visible patterns (data not shown). Executable programs (exe, dll) were the only kind of information examined that showed the same variation and diversity of repeat patterns that can be seen in eukaryotic genomes. Examples of the similarities between genomes and executable code were found at every scale.

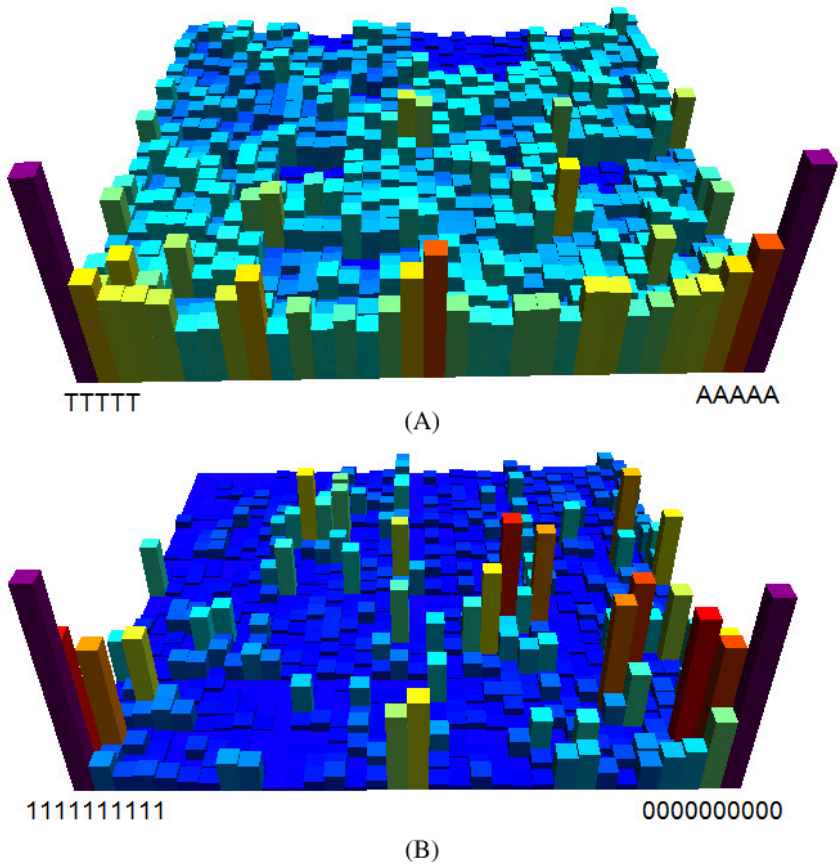


Fig. 2. These bar graphs show the abundance of all strings of length 5 (base-4). Strings are sorted by the CGR algorithm [9]. Coloring in this graph is based on abundance. Low abundance words are blue, medium are green to orange, with high abundance words being purple. A) All 5-mer strings in chromosome 21. In this graph, TTTTT is in the lower left, AAAAA is the lower right, CCCCC is the upper right, and GGGGG is the upper left. The poly-A and poly-T strings are by far the most frequent short strings in the genome. Also dominant in the lower center are two bars correspond to TAAAA and ATTTT respectively. B) All 5-mer (base-4) strings in the SkittleToo.fa computer code file. In this graph, 1111111111 is in the lower left, 0000000000 is the lower right. In computer programs the homopolymer-dominant pattern is just as strong as in the human genome. This computer code example contains the same pair of matching peaks (purple) as well as two smaller matching peaks in the bottom center, which correspond to TAAAA and ATTTT in the genome.

Results – From Small to Large

These results show five primary ways in which computer programs and chromosomes are similar. These five levels of similarity, arranged from smallest to largest, are: homopolymers, tandem repeats, distributed repeats, isochores (sequence bias), and whole chromosome/program structure. These five levels of organization are well known attributes of human chromosomes. Paradoxically, it was actually

easier to find strong examples of these patterns in computer code than it was to find examples in DNA. Overall, computer code appears to be significantly more repetitive than the human genome.

Homopolymers — The human genome has an overabundance of strings that consist entirely of AAAAA or TTTTT (Figure 2). Since Adenine and Thymine bind less strongly than Guanine and Cytosine, these areas of the sequence are spots where the double helix can open more easily. Poly-A strings are found at the ends of mRNA strands as well. In computer code, 0 is often used as padding for a variety of reasons. Data files also contain homo-polymers in structured locations, but prose does not (Figure 1). A small positive number will have a long string of 0's at the beginning while a small negative number will have a long string of 1's. In both code and genomes they can be used as a dividing marker between different elements, similar to a space or paragraph break. Figure 2 demonstrates a strong similarity in the homopolymer patterns within the human genome and executable computer code.

Tandem repeats — The most striking visual patterns seen when higher genomes are visualized with Skittle are the tandem repeats (Figure 3). Tandem repeats in the human genome have long been considered junk DNA left over from replication errors. Tandem repeats are useful in forensics because of their anomalously high mutation rate, which can be up to one million times higher depending on the estimation technique. Both Weber and Brinkmann report mutation rates of at least 7×10^{-3} per locus per haploid per generation [10, 11], while the background mutation rate for the whole genome has recently been measured at 1.1×10^{-8} per position per haploid genome per generation [12]. Given that the patterns of variation visible within genomic repeats do not appear to be random (Figure 3) [8], it is reasonable to consider the possibility that such variation may not be the result of an entirely random mutation process.

In computer programs, tandem repeats are often used to store data in a structured format. When examined with Skittle, computer code shows remarkably similar tandem repeats as those seen in eukaryotic genomes. Also seen are the same types of internal structured variation as genomic tandem repeats (Figure 3).

In the genome we do not understand the function of tandem repeats, but in executable code, tandem repeats can be traced back to the original source code written by a programmer and to its function in the program. For example, visible tandem repeats can be mapped to data files (typically, columns of letters or numbers). Computer data consists of tokens that are often larger than one byte. When a token varies from the consensus, we observe straights columns or "covariance" (Figure 3B) and when a token has a variable length, we observe wavy columns or "indels" (Figure 4B).

Figures 3B and 4B show two specific examples of repeats in machine code, extracted from openofficeorg32.msi. This file is freely available, and is responsible for installing the Open Office software suite. When visualized in Skittle, the file is found to contain all the major features that biologists associate

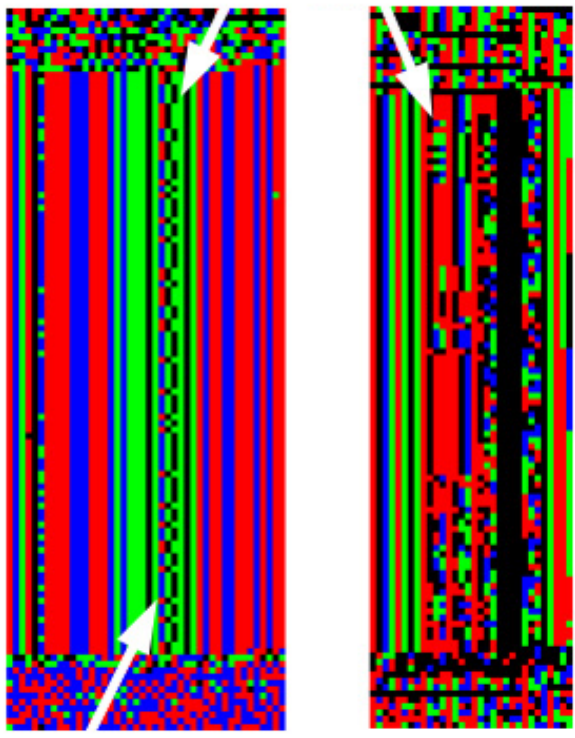
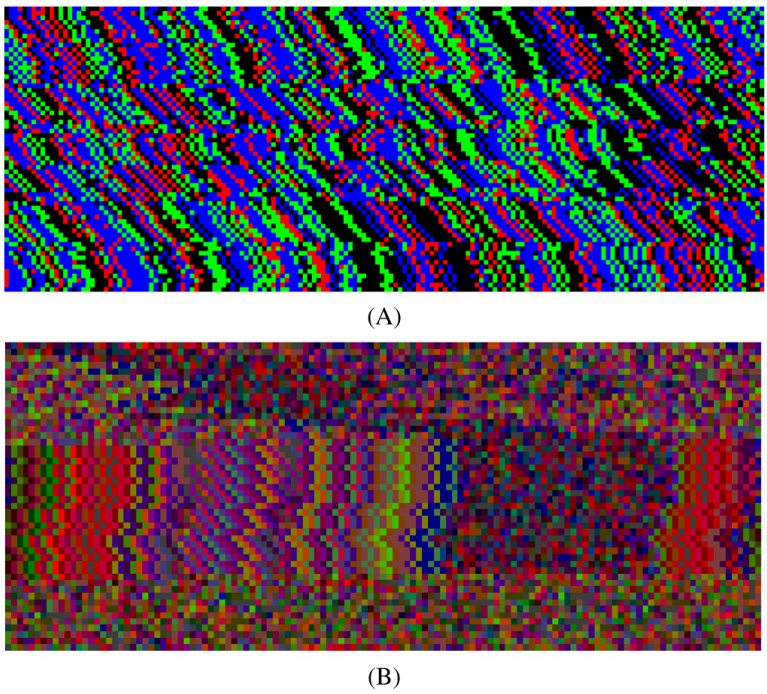


Fig. 3. Structured variation in tandem repeats. Pixel coloring: 00 = A = Black, 01 = C = Red, 10 = G = Green, 11 = T = Blue. A) Chr Y: Start: 392,304 bp Length: 7,128 bp. This tandem repeat lies near the telomere of chromosome Y and was used in the first Skittle paper [8] as a strong example of nucleotide covariance. The arrows point to the substitutions in the 3 columns near the center, that show covariance. B) OpenOfficeorg32.msi: Start: 39,554,645 Length: 4,672 (x2) bits. This repeat shows a strong resemblance to Figure 3A, but it's found in computer code. The nucleotide covariance (arrow) is caused by replacing a token longer than 2 bits in the repeat, which involves simultaneously changing of a series of contiguous bits. Both repeats look like straight vertical bars because they contain no "insertions or deletions". Contrast this with the wavy, staggered appearance of Figure 4. Both repeats have columns that are highly variable, and columns that are entirely invariant.

with chromosomes. It contains isochores, segmental duplications, tandem repeats, distributed repeats, and sequence variations that have the same appearance as mutations. The lower sixth of the file consists of a segmental duplication with a length of 231,600 bytes per repeat monomer with 7 copies. The other half of the file is two larger segmental duplications of 295,852 bytes per repeat monomer.

Figure 4B shows a tandem repeat from openofficeorg32.msi visualized in Skittle. The text inside this particular example is in English, so we can see the content, while most computer code (in Hex) would be unreadable to the average person. A sample of the text in Figure 4B is shown in as follows.



(A)

(B)

Fig. 4. Tandem repeats with indels. Pixel coloring: 00 = A = Black, 01 = C = Red, 10 = G = Green, 11 = T = Blue. A) Chr19: Start: 32,611,935 bp Length: 27,702 bp. Every human chromosome has a centromere, which is primarily a large tandem repeat. This alpha satellite repeat shows both substitutions and indels (wavy columns). B) OpenOfficeorg32.msi: Start: 212,581 Length: 8,001 (x2) bits Scale: 4 bp/pixel. This repeat was picked from computer code as an example that has both substitutions and indels (wavy columns). In this case the variable columns are concentrated together in the middle. These columns encode the unique ID that's written into the registry (see text). For clear visualization, each pixel is one byte. The color is an average of 4 "base pairs" per pixel or 8 bits per pixel. For a more detailed look at visualizing structured variation inside tandem repeats, including nucleotide covariance and indels please see "Skittle: a 2-Dimensional Genome Visualization Tool" [8].

…_REGISTRY_OPENOFFICEORG32{F0B285B1–7227-CDDC-6CEA-FA264CF46679} (REGISTER_DOCX=1) AND (WRITE_REGISTRY…

…_REGISTRY_OPENOFFICEORG32{CF642EF8–3237–7F5A-6D31–18FFF1ACB2C1} (REGISTER_DOT=1) AND (WRITE_REGISTRY=…

…_REGISTRY_OPENOFFICEORG32{98C7CE2B-EA3B-AB57–7F43–6987BCFF2C7E} (REGISTER_DOTM=1) AND (WRITE_REGISTRY…

…_REGISTRY_OPENOFFICEORG32{C2E5C8BB-4D40–61A3–8D84–625E34119744} (REGISTER_DOTX=1) AND (WRITE_REGISTRY…

…_REGISTRY_OPENOFFICEORG32{73E532F7-BAD7-F137–00C6–9188EA72701C} (REGISTER_POT=1) AND (WRITE_REGISTRY=…

…_REGISTRY_OPENOFFICEORG32{837B2E93-F7D2–61BB-D711-E65E54F951AC} (REGISTER_POTM=1) AND (WRITE_REGISTRY…

…_REGISTRY_OPENOFFICEORG32{1F1B63BC-3767–643B-9973–1A893B7488E5}
(REGISTER_POTX=1) AND (WRITE_REGISTRY…

…_REGISTRY_OPENOFFICEORG32{04A22EC4–39CD-4254-A2AD-22E5F170B043}
(REGISTER_PPS=1) AND (WRITE_REGISTRY=…

The repeated instruction "REGISTRY_OPENOFFICEORG32" provides context for the data. These are registry entries to be written during installation. The variable parts of this tandem repeat are the unique entries being entered e.g. "{F0B285B1–7227-CDDC-6CEA-FA264CF46679}". This setup means that each repeat entry can be read independently. The variable columns correspond to the unique entries while the duplicated text provides context. Also notice the next token lists various file types: e.g. "(REGISTER_DOCX=1)". DOCX, DOT, DOTM, PPS are all file types that Open Office can read. Most file types are 3 letters (TXT) but some file types are four letters long (DOCX). This token will vary length by one letter, creating the shifts referred to in biology as indels. Computer code also contains tandem repeats in the form of repetitive instructions.

Distributed repeats — Distributed repeats are sequences that are nearly identical and are found in many locations in a genome. The most common distributed repeats in the human genome are LINES and SINES, which have many functions, including regulation of transcription [14]. Similarly, computer programs have specific commands that are used frequently such as ADD, STORE, and LOAD. These common commands create a distributed repeat pattern. Distributed repeats are also the one repetitive pattern that can be observed in English prose. Words like "the" occur much more often than chance along with longer sentence fragments that are frequently reused. "For example", "the reality is", "a few", "a little", "about time", and "at this stage" are all used more often than would happen by chance. Figure 5A shows a given distributed repeat in the human genome, as mapped by Skittle, and Figure 5B shows a similar distributed repeat within SkittleToo.exe.

Isochore Patterns — In eukaryotic chromosomes, the sequence shows usage bias changes between G/C vs. A/T. This change in bias has been known since at least the 1970s, creating visible bands under a microscope when the chromosome is stained [15, 16]. These bands correlate with the presence of much larger DNA elements, and represent a basic division in eukaryotic sequences. G/C rich regions contain more genes, CpG islands, and are physically unpacked in the interphase nucleus. Unpacked regions are referred to as open chromatin because they are less dense [13]. A/T rich regions are associated with closed chromatin and lower levels of transcription.

Surprisingly, the isochore type of pattern is even stronger in computer code. We can observe the same kind of character usage bias, revealing both gradual changes, and sharp disjunctions. The change is due to the large-scale organization of types

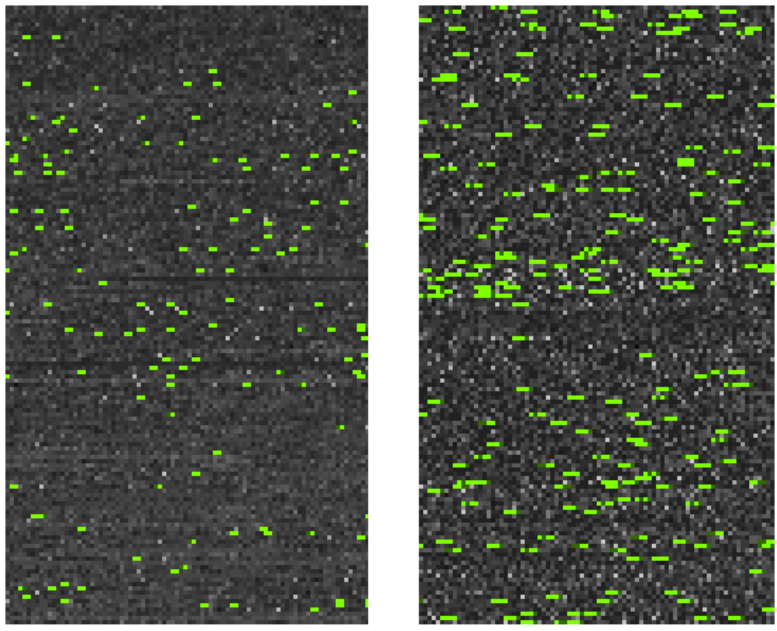
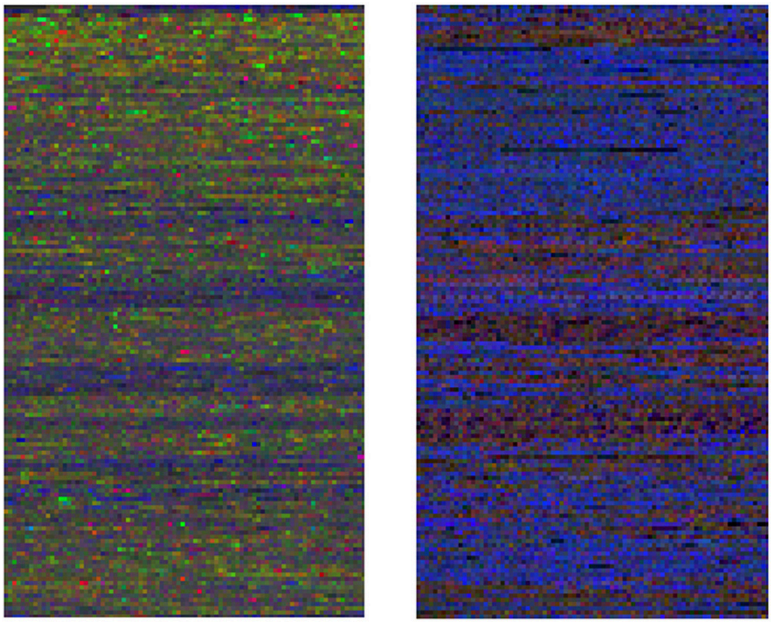


Fig. 5. Distributed repeats are highlighted in green using the Skittle Sequence Highlighter similar to searching for text in a document. Gray pixels are non-matching sequence. A) Chr19 Start: 97,281 bp Length: 338,428 bp. Sequence: TGGGATTACAGGTGTGAGCCACCGCGCCCG at 80% similarity. The human genome is filled with distributed repeats, but their positioning is not entirely random. Some bands of the chromosome will have very few of a certain repeat sequence, while others will be very dense. These concentrated bands on the chromosome follow the isochore patterns (Figure 6). For example, Alu repeats are concentrated in GC rich regions along with genes [13]. B) SkittleToo.fa Sequence: AGAGCCACAAGCAAACAGAGCAC (0010001001010001000010010000000100100010010001) at 80% similarity. The banding pattern of repeats observed in human chromosomes is actually easier to see in computer code because the computer programs are more repetitive. Note the horizontal bands where many repeats are highlighted in green and the dark region just below it with no repeats highlighted.

of elements in code. Execution code and data are separated. Since data types are contiguous, the same encoding scheme (data format) will show up in discrete blocks. Elements in the program employ different codes and some codes are used in combination with each other. Each code will have a different bit distribution. For example, English letters occupy only 52 of the 256 possible values on the ASCII table. This means that by using Skittle one can visually differentiate English text from other types of codes. Isochores in programs are the result of large-scale organization and the use of multiple codes.

Whole chromosome/program architecture — Tandem repeats are much more common near the ends of chromosomes and near the centromere. Computer



(A) (B)

Fig. 6. Examples of isochore-type structure seen at much larger scales. Pixel coloring: 00 = A = Black, 01 = C = Red, 10 = G = Green, 11 = T = Blue. This figure is a zoomed out view of a whole chromosome arm where each pixel is the color average of thousands of nucleotides (see Scale). A) Chr19: Start: 1 bp Length: 22,272,512 bp Scale: 1,061 bp/pixel. Using color averaging, the changing bias in GC content can be clearly seen on the short arm of chromosome 19. GC content has a high correlation with many other genome elements. B) SkittleToo.fa Start: 10,509,078 Length: 3,715,584 (x2) bits Scale: 177 bp/pixel. Isochore-type patterns can be clearly seen in computer code, even more clearly than the genome. Regions with many 1's in the code appear blue while the areas more rich in 0's have a dark reddish color. Even the variation in the size of the isochore bands is similar in both A and B, though the scale is different. Genomic tandem repeats appear at this scale as spots and streaks of bright color (usually green or red), and a similar pattern can be seen in the computer code in the small black horizontal lines that litter the image. (This image's contrast was increased for clarity in printing.)

programs also store large repetitive blocks of data at the end of programs. Many executable files actually contain a majority of repetitive sequences because of icons and graphics stored in the executable. Repeats are packed in at the end of computer programs because they are organized to make them easier to use. This organization extends to the RAM, where memory is allocated into the Stack and the Heap. The Stack contains the primary program and just a few local variables. The Heap contains the majority of the data and frequently changes size and composition during execution. Similarly in DNA, repetitive elements are packed towards the end of chromosome arms and tandem repeats show much higher mutation rates (often

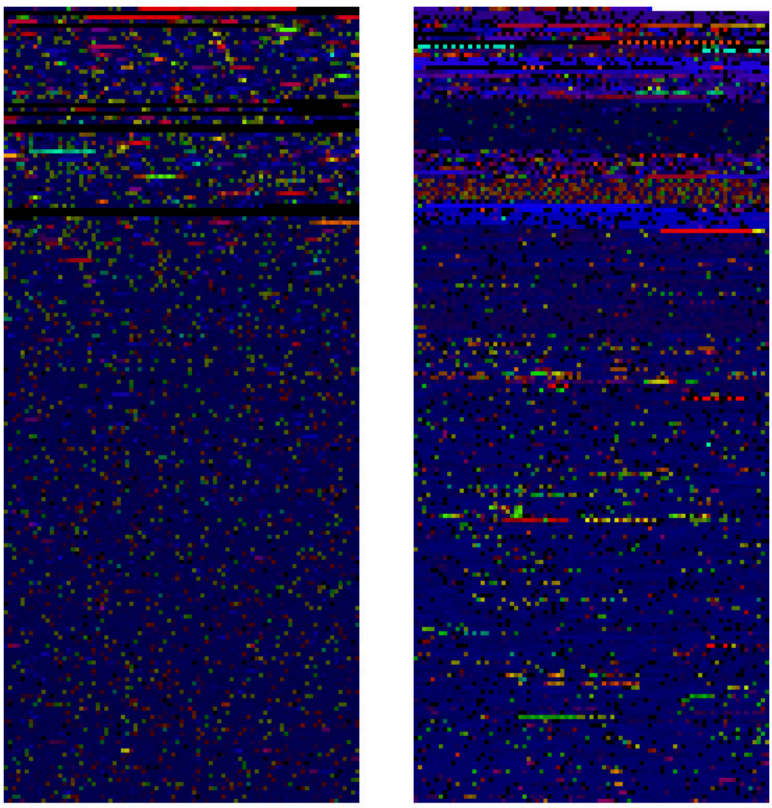


Fig. 7. Skittle's Repeat Overview is used to highlight tandem repeats in bright colors to show the structure and distribution of a whole chromosome arm or an entire program. A) Chromosome X: Start: 1 bp Length: 8,670,000 bp Scale: 500 bp / pixel. Tandem repeats displayed in bright colors are primarily concentrated near the telomere in the top of this image. Large black lines are areas that were not sequenced, often because these regions are large tandem repeats. B) SkittleToo.fa Start: 26946779 Length: 21,501,600 (x2) bits Scale: 1,240 bp / pixel. The second half of the Skittle executable code can be seen here with repeats in bright colors and non-repetitive regions in dark blue. The image has been flipped vertically for comparison with Chr X. The less repetitive control code occupies the lower ⅔ of the image, while the repetitive icons, data tables, and other program resources are stored at the end of the file (displayed at top). In both sequences, repetitive structures are far more abundant near the ends.

seen as changes in size) than anywhere else in the genome [10, 11]. Like in computer programs, eukaryotic genomes tend to segregate highly repetitive sequences, which is further exemplified by recent research showing that chromosomes are organized in 3D space in the nucleus. For example, in the yeast genome, the centromeres of all the chromosomes can cluster together in one spot while the arms of the chromosome stretch out from that point. This is called a Rosette pattern [17].

The mammalian genome organization is noticeably more complex [18], but the pattern of aggregating similar sequences is still a major factor.

Discussion

The striking structural similarities between higher genomes and computer code strongly suggest they operate on similar principles, and that genomes and computer code may each instruct us on how to more fully understand the other. The results of this paper show that computers and cells use very similar encoding patterns despite the fact that the first code compilers could not have been designed to mimic the genome because the invention of compilers predates genome sequencing technology.

I propose that the simplest explanation for this similarity is that these findings represent convergent evolution driven by similar design constraints. Computers were not developed all at once. Instead, a number of different possibilities were tested. Similarly, compilers have gone through a series of revisions and optimizations. As an ongoing process of refinement, computer architecture is subject to many of the same constraints as biology, meaning that many of the optimal encoding patterns will be the same.

The comparison is half analogy and half reality. Obviously, there are major differences between the molecular computing of DNA and the electronic architecture of modern computers. Yet the first computer conceived by Alan Turing was an entirely mechanical apparatus moving along a tape — which has more resemblance to a polymerase on DNA than it does to modern computers [19]. Computer Science has shown that computation can take many forms, yet the fundamentals principles and constraints seem to remain the same.

The following are suggested as possible parallels between computers and biological systems:

Biology	Computer	Comments
DNA	**Hard Drive**	DNA is analogous to a hard drive because it serves as the canonical, non-volatile copy that is copied but not frequently edited.
RNA	**RAM**	RNA is analogous to the RAM in a computer because it acts as the active, working copy of the information that is edited, used, and then discarded.
Tandem Repeats	**Data blocks**	This explains the anomalously high mutation rates. Cells are purposefully storing inherited information in the DNA strand.

Continued

Continued

Polymerase + Ribosomes	**Processors**	The cell is a multi-processor system, with multiple parallel events occurring and being communicated through epigenetic modification and RNA. Variety in protein/RNA complexes are processors specialized for different tasks.
Cytoplasm Phenotype	**Output**	Most of the computation that goes into the decision process is never obvious to the user.
Nucleus	**Motherboard**	Computational center for the cell with hard drives integrated as closely as possible.
Nucleolus	**CPU**	Central area where most of the processors and memory is congregated for speed reasons.

Differences — While there are striking similarities between genomes and executable programs, there are also very important differences. These differences serve to highlight why the similarities are so informative: they reveal the underlying design constraints at work in both.

- DNA lacks large blocks of numbers sorted in ascending order.
- DNA does not have as many zero values as code in large blocks (such as padding), though the human genome does have a strong bias towards strings of A's or T's.
- Computers usually use a fixed word length, which shows up as a periodicity in the frequency graph. Exons in DNA show this same pattern because the codon code follows a fixed length look-up table, but there are many variable length elements as well.
- For structured variation in tandem repeats, computer code will often have zeroed out fields as part of covariance. A "zero value" has not been directly observed in biological sequences. With better token recognition, zero values could simply be skipped, in which case they would look like deletions.

These findings provide new tools to direct future research. In computer science, engineers use the attributes of an object to determine the type of object. This is called duck typing because it follows the phrase, "If it walks like a duck, and quacks like a duck, it's a duck". The comparison between computer programs and the human genome shows that elements in the genome share the same attributes with programming products. By applying duck typing, we get more than just a single hypothesis. We get a whole set of hypotheses about the function of any element in the genome that has similar properties to an element in a program. The starting hypothesis would be that the reason they look the same is because they are

fulfilling the same types of function. This gives us a useful road map for designing biological experiments and predicting function.

Until recently, previous research has focused on genes and promoters, which constitute at most 3% of the genome. The rest of the genome has been a complete mystery. Despite some advances, the task ahead still remains daunting. The model of the genome as a computer system can act as a paradigm for exploring the whole genome, not just protein coding genes. Historically, Egyptian hieroglyphics were not deciphered until the discovery of the Rosetta stone. This was crucial because they found the same things written in Greek as was written in the indecipherable hieroglyphics. This allowed real translation between the two. It is possible that computer program architecture may be the Rosetta stone for unlocking the rest of the genome. Without a working model, the human genome appears to be indecipherable junk. But using a comparable architecture, we stand a real chance of deciphering the whole genome, starting with the basic components that make up all executable programs.

The rewards for such an endeavor are enormous. There are essentially two fields of science that can benefit from adapting the principles and knowledge of one into experiments and techniques in the other. Computer science can offer biologists information on system design, encapsulation, encoding, and abstraction. But biological systems are vastly more sophisticated than modern computers. Computer science can learn many lessons from biology about massively parallel architectures, self-assembling machines, overlapping codes, etc. With complete understanding of the mechanisms of biology, biologists might program an organism's metabolism for a specific task. With a clear understanding of the computational components of the cell, engineers might harness yeast as an all-purpose computer that could self-replicate, giving humans access to exponentially increasing computer power.

Conclusions

Executable code and genomes show striking similarities in the way information is structured, despite the fact that their physical mechanisms are completely different. I propose that this is because both kinds of code are subject to many of the same constraints dictated by information theory. Given the striking similarities between genomes and computer code, it will be fruitful to study the architecture of executable computer code, so that we might better understand how genomes function. We know that in computer code, there is no "junk code", and that all the structure we are seeing is functional. Therefore it is reasonable to expect that function underlies all the analogous structures seen within the genome.

Acknowledgments

The basic research that led to this discovery was supported by FMS Foundation.

References

1. Shapiro JA, von Sternberg R (2005) Why repetitive DNA is essential to genome function. Biol Rev 80:227–250.
2. Ohno S (1972) So much "junk" DNA in our genome. In: Smith HH (ed) Evol Genet Syst, Brookhaven Symp Biol 23:366–370.
3. Doolittle WF, Sapienza C (1980) Selfish genes, the phenotype paradigm and genome evolution. Nature 284:601–603.
4. Barash Y, et al (2009) Deciphering the splicing code. Nature 465:53–59.
5. Kapranov P, Willingham AT, Gingeras TR (2007) Genome-wide transcription and the implications for genome organization. Nat Rev Genet 8:413–423.
6. The ENCODE Project Consortium (2007) Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. Nature 447:799–816.
7. Pheasant M, Mattick JS (2007) Raising the estimate of functional human sequences. Genome Res 17:1245–1253.
8. Seaman JD, Sanford JC (2009) Skittle: a 2-Dimensional Genome Visualization Tool. BMC Bioinformatics 10:452.
9. Jeffrey HJ (1990) Chaos game representation of gene structure. Nucl Acids Res 18(8): 2163–2170.
10. Weber J, Wong C (1993) Mutation of human short tandem repeats. Hum Mol Genet 2(8): 1123–1128.
11. Brinkmann B, et al (1998) Mutation Rate in Human Microsatellites: Influence of the Structure and Length of the Tandem Repeat. AJHG 62 (6): 1408–1415.
12. Roach J, et al (2010) Analysis of Genetic Inheritance in a Family Quartet by Whole-Genome Sequencing. Science 328: 636–639.
13. Lieberman-Aiden E, et al (2009) Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. Science 326: 289–293.
14. Faulkner GJ, et al (2009) The regulated retrotransposon transcriptome of mammalian cells. Nat Genet 41:563–571.
15. Macaya G, Thiery J-P, Bernardi G (1976) An approach to the organization of eukaryotic genomes at a macromolecular level. J Mol Biol 108:237–254.
16. Tykocinski ML, Max EE (1984) CG dinucleotide clusters in MHC genes and in 5' demethylated genes. Nuc Acids Res 12:4385–4396.

17. Duan Z, et al (2010) A three-dimensional model of the yeast genome. Nature Letters 465: 363–367.
18. Cremer T, et al (2001) Chromosome territories, nuclear architecture and gene regulation in mammalian cells. Nat Rev Genet 2: 292 – 301.
19. Turing, AM (1936) On Computable Numbers, with an Application to the Entscheidungs problem. Proceedings of the London Mathematical Society Ser. 2 Vol. 42: 230–65. 1937.