Error-Tolerant Coding and the Genetic Code

by

Alexander Gutfraind

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada 2006

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.
I understand that my thesis may be made electronically available to the public.

Abstract

The following thesis is a project in mathematical biology building upon the so-called "error minimization hypothesis" of the genetic code. After introducing the biological context of this hypothesis, I proceed to develop some relevant information-theoretic ideas, with the overall goal of studying the structure of the genetic code. I then apply the newfound understanding to an important question in the debate about the origin of life, namely, the question of the temperatures in which the genetic code, and life in general, underwent their early evolution.

The main advance in this thesis is a set of methods for calculating the primordial evolutionary pressures that shaped the genetic code. These pressures are due to genetic errors, and hence the statistical properties of the errors and of the genome are imprinted in the statistical properties of the code. Thus, by studying the code it is possible to reconstruct, to some extent, the primordial error rates and the composition of the primordial genome. In this way, I find evidence that the fixation of the genetic code occurred in organisms which were not thermophiles.

Acknowledgements

Foremost, I must thank my supervisor Dr. Achim Kempf for introducing me to and guiding me through the wonderful subject of mathematical biology. None of this would have existed if not for his selfless willingness to spend long days with me developing the ideas herein. I particularly appreciate the intellectual freedom he has given me. I must also thank my parents and friends for accepting my hermitic pursuit of science. Finally, I must thank everybody in the Kempf Lab for keeping my spirits up these two years: Larissa, Tom, Rob, Sven, Yufang, Rob, David, Cedric, William and Angus.

Contents

List of Figures

If you want to understand life, don't think about vibrant, throbbing gels and oozes, think about information technology. Richard Dawkins(1986)[14].

These semantic aspects of communication are irrelevant to the engineering problem. Claude Shannon(1948)[76].

Chapter 1

The Origin and Evolution of the Genetic Code

The last two decades brought many exciting advances in molecular and evolutionary biology. We have now in our possession a detailed picture of the biochemical processes that sustain life, as well as an account of the key evolutionary transformations on the road from pre-biotic chemistry to primates. The interest of this thesis is set on one particular transformation - the evolution of the genetic code. This area has also been studied in detail in the last two decades.

In this chapter, we review the biological function and evolution of the genetic code. We also discuss current theories about the evolution of life, and highlight the debate about the temperatures in which life originated. We also spend considerable time discussing the evidence underlying one particular theory for the evolution of the code, which views the genetic code as an error-coding device. Consequently, the topic of Chapter 2 is error-tolerant coding and information theory. Chapter 3 uses these biological and information-theoretic ideas to address the question of temperature above.

The outline of this chapter is as follows:

- The Genetics and Origins of Modern Organisms
- The Emergence of the Genetic Code
- The Error Minimization Hypothesis

1.1 The Genetics and Origins of Modern Organisms

Our first step is to review the function of the genetic code in modern organisms. We will briefly review the structure of the genetic machinery, discuss the genetic code and finally explain how they evolved. The key biochemical concepts of this section are summarized in Appendix B. In the next section we will focus on the genetic code, its structure and evolution. All of the concepts discussed here can be found in textbooks on genetics (e.g. Brooker [9]).

1.1.1 The flow of genetic information in modern organisms

The synthesis of proteins is one of the defining characteristics of all life-forms on earth. Proteins are so ubiquitous in part because they are able to act as catalysts ("enzymes") - increasing the rates of biochemical reactions a million-fold or more. They are also very specific in their function. These two properties of proteins allow cells to perform complex metabolic reactions, respond to their environment, self-regulate and reproduce.

Proteins are synthesized following instructions stored in the genome. The genetic machinery responsible for protein synthesis has four information-related functions: storage, transcription, translation and replication. Let us consider storage first. Storage in most organisms with the exception of some viruses is performed by several very long DNA ("deoxi-ribonucleic acid") molecules known as "the genome" (the genetic code is something else entirely). In its double-helix form, DNA is a polymer whose monomers are known as "nucleotides". Four nucleotides are found in DNA: Thymine, Cytosine, Adenine and Guanine (abbreviated T, C, A, G). Two matching DNA molecules form a DNA double-helix, by forming a "base-pair" at each nucleotide: C pairs with a G and T pairs with A. The nucleotides fall into two chemical families: while T and C belong to the pyrimidine family (abbreviated Y), characterized by a single ring, A and G are members of the purine family (abbreviated R), characterized by a double-ring. Although DNA is an acid, its only function is to store information. While most organisms use DNA, another polymer, RNA ("ribonucleic acid"), is also used (especially in viruses). RNA is in many ways similar to DNA, also being a polymer built from four nucleotides. While RNA, like DNA, uses Cytosine, Adenine and Guanine, RNA replaces Thymine with a chemically similar nucleotide

Uracil (abbreviated U) (Fig. 1.1). Also, unlike DNA, RNA is used for purposes other than information storage, as we shall see.

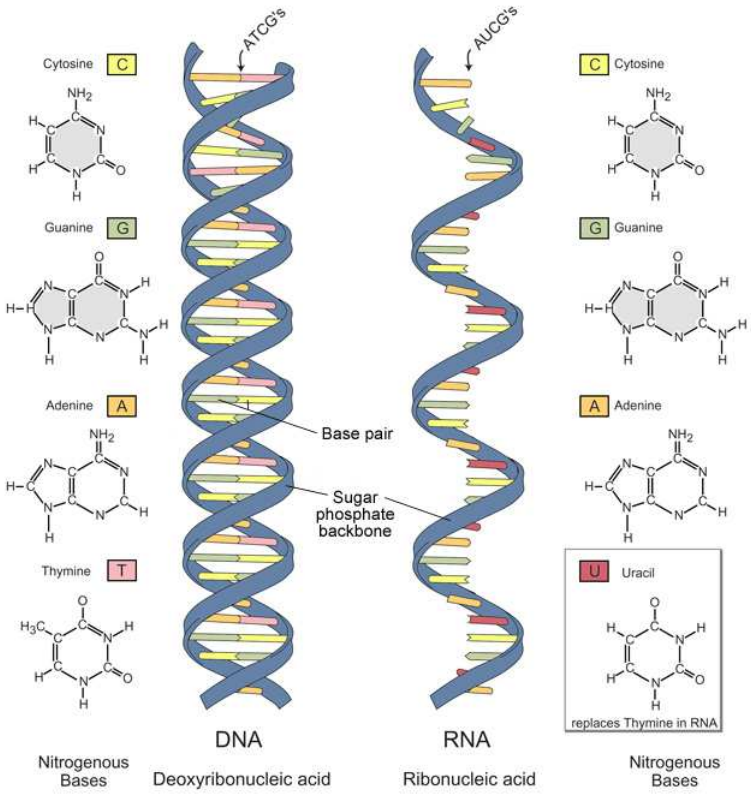

Figure 1.1: **The double-stranded DNA polymer**
Each DNA strand consists of a backbone to which attaches a sequence of nucleotides. Each nucleotide pairs uniquely with a nucleotide from the other strand. The pairing is through a double or a triple hydrogen bond. DNA differs from RNA in three ways: RNA has Uracil rather than Thymine, uses a slightly different backbone and typically appears in single strand form. Image adapted from an original by the National Human Genome Research Institute.

The four nucleotides of DNA are not always used in equal quantities. For instance, yeast DNA has 18.3% G, while in E. coli bacteria G forms 24.9% of the double-helix.

Nevertheless, some patterns exist. First, the rules for pairing nucleotides (G=C and A=T) imply that G and C must come in equal quantity, as must A and T - a rule discovered by Erwin Chargaff and famously used by Watson and Crick to infer the double helix. Second, and more surprisingly, the rule approximately holds within each strand of DNA (organelle such as mitochondria are an exception). The reasons for this regularity are not truly understood [55].Thirdly, it has been found that the frequencies of nucleotides G and C in RNA correlate with temperatures. Namely, organisms adapted to high temperatures ("thermophiles") have elevated frequencies of G and C [27], with G+C forming as much as 80% of double-stranded RNA (Fig. 1.2). This correlation is an evolutionary adaptation to high temperature [87], as follows. As compared to DNA, RNA is relatively unstable chemically especially in high temperatures. A partial solution to this instability is to use double-stranded as opposed to single-stranded RNA. Moreover, there are three hydrogen bonds holding the G-C pair, while there are only two such bonds in the A-U pair(see Fig. 1.3). Therefore, by using elevated G and C in their double-stranded RNA, thermophile organisms adapt to their hot environment. With this and other adaptations, some can live and flourish in temperatures exceeding 100°C.

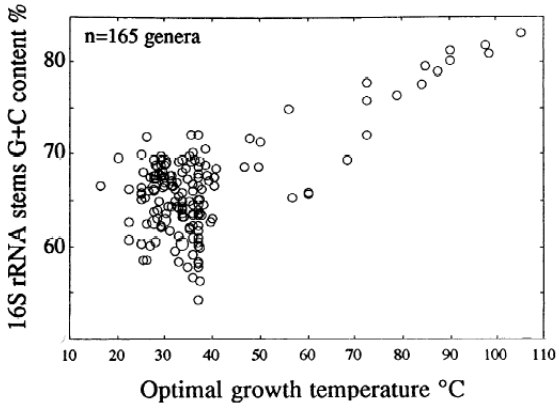

Figure 1.2: **Optimal growth temperature and G+C content of RNA**
This plot displays the G+C content in rRNA (a type of RNA) of 165 genera in the Prokarya domain. Most of the species here are mesophiles, as seen from the large cluster of points on the left. As the temperature increases, there is a statistically significant and large increase in the G+C content. No thermophiles have a low G+C in their rRNA. Image adapted from an original by Galtier *et al.* [27].

Following storage, genetic information undergoes transcription - an intermediate step in the processes of synthesizing proteins. The goal of transcription is to copy the information locked in DNA into mRNA ("messenger" RNA) which is later used in the production of proteins. The key step of transcription is performed by a protein, the RNA polymerase, which moves along one strand of DNA and produces a single-stranded mRNA "transcript" by pairing the RNA to the DNA (Fig. 1.4). Thus, for example, the sequence *AGCTT* is transcribed as *UCGAA* This means, curiously, that the sequence of DNA containing a gene (the "coding" or "sense" DNA sequence) actually stores the complementary sequence to the sequence that will be made into a protein. In some types of cells, to complete the preparation of the transcript it is necessary to modify the transcript, namely, to remove the introns (the "junk DNA").

Next we have translation - the process of reading the mRNA transcript and building a protein. Like DNA and mRNA, each protein is a polymer consisting of a chain of monomers known as amino acids. During transcription, the mRNA is mapped into an amino acid

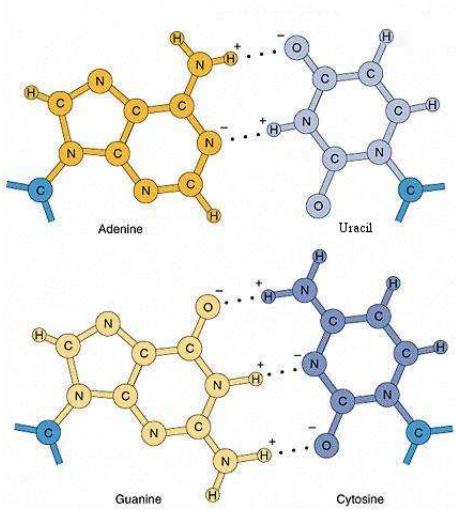

Figure 1.3: **Double and triple bonds in RNA**
The triple bond in the G-C pair of RNA is stronger than the double-bond of the A-U pair. Image adapted from an original by Kenneth G. Wilson.

chain with the aid of tRNAs ("transfer" RNA). The formation of a chain is performed by the ribosome which moves along the mRNA during translation, appending amino acids to the chain (Fig. 1.5). The process eventually terminates and the resulting chain or "polypeptide" folds to become a protein with a complex three-dimensional structure. This diversity of final folded structures is what gives proteins their high efficiency and specificity.

A crucial role in translation is played by tRNA. Like mRNA, tRNA is produced from the DNA in the genome using an RNA polymerase. However, unlike mRNA, the tRNA becomes active only after folding into a complex "tertiary" structure. tRNA is often considered to be an enzyme. However, the building blocks of tRNA are RNA nucleotides, while proteins are built from amino acids. One end of each tRNA is known as the anticodon arm. The key part of the arm is the anticodon - three nucleotides which bind to three nucleotides (known as triplet or "codon") in the mRNA. Another end of each tRNA, known as the acceptor stem, binds to an amino acid which is then attached to a growing chain of amino acids (Fig. 1.6). Beside tRNA, translation involves another specialized RNA enzyme, the rRNA

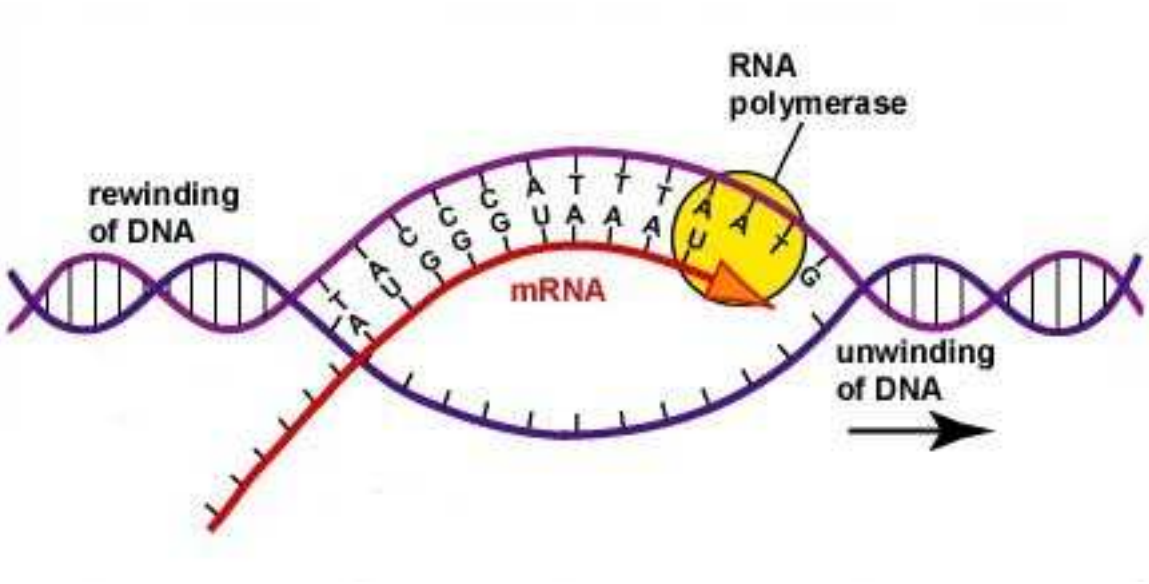

Figure 1.4: **Transcription**
RNA-polymerase attaches to a sequence of DNA containing a a gene and makes a complementary copy of mRNA, copying one nucleotide at a time. Image adapted from an original by Gary E. Kaiser.

(ribosomal RNA), which we will discuss in the next section. The role of tRNA is important from the information-theoretic view. During translation, tRNA chemically "reads" three symbols in the mRNA and translates them into a chemical message in the form of an amino acid. Each triplet in mRNA is always mapped uniquely to an amino acid, but up to 6 triplets may encode each amino acid. Such a many-to-one mapping (called "degeneracy" in molecular biology) is characteristic of many error-correcting codes, discussed in the next chapter. It is convenient to represent the genetic code in a table showing how each of the 64 codons is mapped by tRNA (Fig. 1.7). One can see that all but 3 of the codons map to one of 20 amino acids. The remaining three cause the translation process to terminate, and are therefore known as "stop codons" (Fig. 1.7).

The genetic code contains many patterns. Most discernible is the tendency of the third nucleotide in many codons to be partially or fully redundant. Biochemically, this

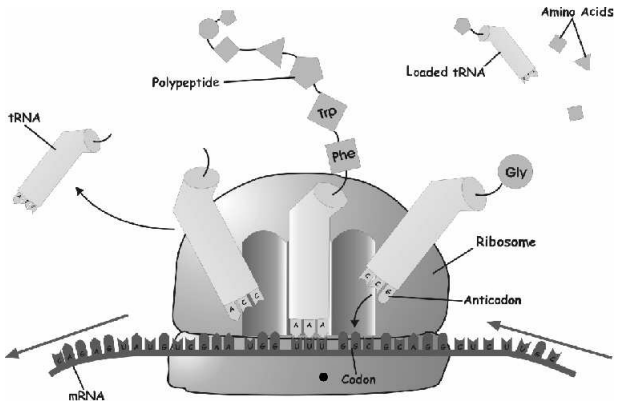

Figure 1.5: **Translation**
Translation resembles in many ways the process of decoding a telegraph onto a tape. Genetic information arrives in the mRNA and is decoded into a message - the protein. The decoding of each codeword is performed by tRNA which binds to triplets (or codewords) in the mRNA. The decoding is powered and catalyzed by the ribosome. Image adapted from an unsigned original at http://www.guidobauersachs.de/genetik/.

is known as "wobble" because the tRNA anticodon binds to a codon in mRNA that may be less than a perfect match for it. For instance, the codons UUU and UUC encoding Phenylalanine (abbreviated Phe) bind to the anticodon AAG, although the bond UUU-AAG is not perfect. Other patterns [83] are harder to see. First, amino acids which are coded by similar codons tend to be chemically similar themselves. For instance, amino acids such as Phe, Leu and Val are hydrophobic and all coded with codons containing U in the middle position. In contrast, His, Gln, Asp, Glu and Lys are hydrophilic and all have A in the middle position. Second, some evidence suggests that amino acids coded by nearby codons tend to lie in the same biosynthetic chain.

We have seen how the tRNA-based translation mechanism creates a mapping called the genetic code. It may seem peculiar that the genetic code is involved not only in translation, but also in replication, obliquely, for the following reason. Replication occasionally introduces mutations into the genome - mutations that find their way into mRNA and then proteins. However, the effect of mutations on the cell (the "phenotype") is determined by

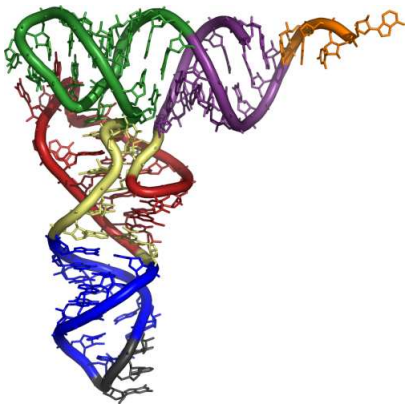

Figure 1.6: **Tertiary structure of tRNA**
The nucleotides are shown as small rings mostly in the interior of the molecule, and the RNA backbone is the thick tube. The acceptor stem is color purple in the top right. The anticodon is at the bottom in black. Image source: Public domain archived in Wikipedia

the genetic code, because it is the genetic code that defines what kind of amino acid the mutated sequence encodes. This has a substantial evolutionary effect as well. Because of the aforementioned "wobble" effect, mutations in the third position in the genome often do not change the amino acid encoded. In contrast, mutations in the middle position (that often determines hydrophobicity) may lead to a substantially different amino acid. As a result, such mutations often cause proteins to misfold and so are very harmful, even lethal. Thus, because of the structure of the genetic code, only mutations in the third position are likely to be inherited - a fact that is quite visible when comparing genetic sequences in related organisms: there is usually an abundance of variation in the third position but only a few changes in the sequences at the first and second codon positions.

How is it then, that the genome is replicated? In organisms that have a genome consisting of a double-stranded DNA, replication of DNA is in many ways similar to transcription into mRNA. The double-stranded DNA is unwound by specialized proteins, and the DNA polymerases travel along each strand attaching to each nucleotide its complement. Since the knowledge of one strand is sufficient to reconstruct the other, the process of replication produces two identical sequences of double-stranded DNA (Fig. 1.8). There is however,

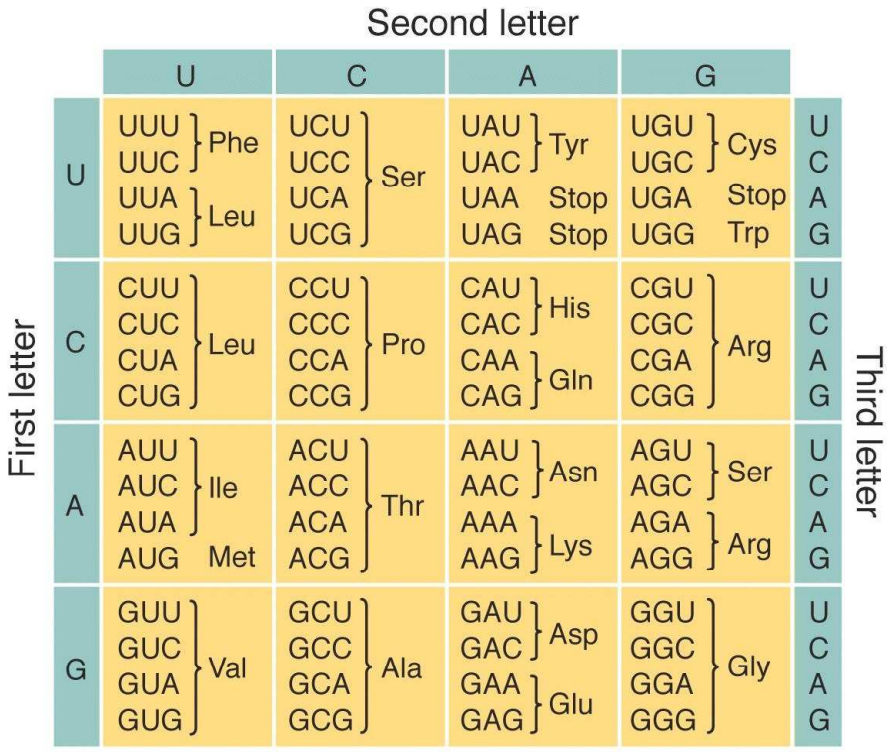

Figure 1.7: **The Standard Genetic Code**
The 64 codons of the genetic code can be put into a table. Written beside each triplet is the abbreviated name of the corresponding amino acid. Three codons act as stop signals and other codons (not shown) initiate translation. Image source: Steven M. Carr

an important detail. The polymerases are only able to travel in one direction (known as the 3'-5' direction) along each strand. This direction corresponds to the direction in which DNA is unwound in only one strand. As a result, replication along that strand is performed quickly. In contrast, replication of the other strand (the "lagging strand") is performed in reverse, by several polymerases working on different sections of the strand and with the assistance of auxiliary proteins.

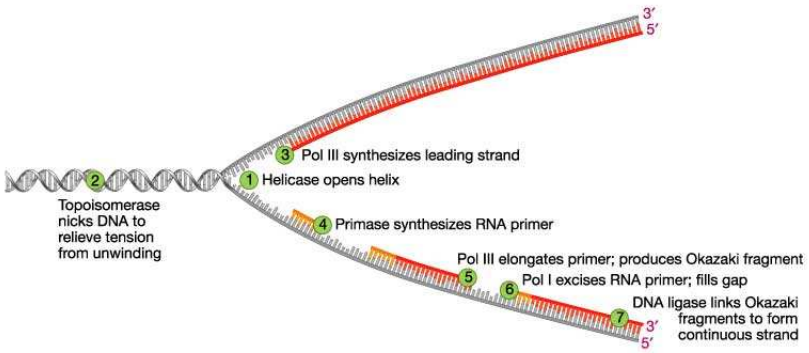

Figure 1.8: **Key steps in the replication of double-stranded DNA**
The numbers indicate the order of the steps, from (1) to (7). Notice that there are several polymerases (Pol I and Pol III) as well as auxiliary proteins involved in replication. Notable features of replication of the lagging strand are the RNA "primers" and the fragmentation of copying. Notice also how large parts of the lagging strand remain in single-strand form. As a result, they experience much more chemical degradation. Image source: Rebecca C. Jann

1.1.2 Genetic Errors

The processes of replication, translation, transcription and storage are error-prone. As a result, organisms have evolved many mechanisms to overcome these errors. For instance, the polymerases performing the replication have built-in proofreading mechanisms. If the wrong nucleotide is inserted accidentally, the polymerase is frequently able to detect the error, excise the ill-fitting nucleotide and repeat the replication. There are additional proteins involved in error correction. For instance, special proteins exist to repair double-strand breaks in DNA (see e.g. Jackson [39]). With those and other mechanisms in place,

error rates are quire low. For instance the error rate in replication is below 1 error per 10^8 nucleotides.

Despite the repair mechanisms, errors are an inescapable part of molecular genetics. There are several types of them. The most common [5] are the twelve substitution errors, in which one nucleotide is replaced by another. These can occur in replication, transcription or translation. If one examines empirical measurements of substitutions some patterns emerge (e.g. Fig. 1.9). The four most frequent errors are known as "transitions", while the

Pattern of nucleotide substitutions (in % of total)				
From **To:**	**T**	**C**	**A**	**G**
T	-	**8.2**	4.4	3.3
C	**21.0**	-	6.5	4.2
A	4.7	5.0	-	**9.4**
G	7.2	5.3	**20.7**	-

Figure 1.9: **Substitution rates**
The rates of substitutions were estimated by comparing sequences of related organisms. The four boldface numbers give the "transition" errors. Adapted from data in Li [51].

other eight errors are called transversions. Transitions are errors replacing a pyrimidine with another pyrimidine ($T \leftrightarrow C$) or a purine with another purine ($A \leftrightarrow G$) and are more likely than the transversion errors. The difference between the rates of transitions and the rates of transversion varies greatly. It can be as low as one to one and as high as 15 to one in mitochondrial genomes [51]. The transitions tend to be more likely than transversions primarily because transversion errors are easily detected, since they distort the geometric structure of the double-strand. In each base pair, a pyrimidine (a small single ring molecule) pairs with a purine (a larger, double ring molecule). A transversion error leads to a pair that is either too large or two small.

If some nucleotide X is substituted by another nucleotide Y in transcription or translation, the effect on the protein is direct. In contrast, such an error during replication may have a more involved effect. Recall that during replication a complementary strand is attached to each of the strands in double-stranded DNA, but a gene is coded on only one of the strands, called the "template" or the "coding strand" (CS). If $X \to Y$ error

occurs while forming the CS from its coding strand, then the error may be expressed in this new generation of cells (G1). If the error occurs in the non-CS (also known as the "sense strand"), then it will not be expressed in G1. However, whether the error occurs in the CS or the non-CS, in the generation G2 (after G1) there will always be one expressible mutation, albeit its nature would be different. If $X \to Y$ occurs in the CS, the descendants will inherit this mutation, while if it occurs in the non-CS, the descendants will inherit $X \to cY$, where cY is the nucleotide complementary to Y. Notice also that a gene in the CS is not the sequence that will be translated into a protein, but rather its complement. Thus, if the coding strand experienced the error $X \to Y$, the actual error in translated sequence would be $cX \to cY$.

Beside the twelve substitution errors, other types of errors occur as well. Occasionally, a nucleotide is simply not copied or knocked out from DNA by free radical or radiation. These deletion errors are usually very harmful because they lead to a "frame shift" in the remaining codons in the gene. Namely, the partition of the gene into a sequence of codons for translation is corrupted. For example, if in the sequence $(UCA, UCA, UC\mathbf{A}, UCA, UCA, UCA)$ the highlighted A is deleted, the sequence becomes $(UCA, UCA, UCU, CAU, CAU, CA)$. Thus, instead of encoding the amino acids sequence (Ser, Ser, Ser, Ser, Ser) the new sequence reads (Ser, Ser, Ser, His, His, ..).

1.1.3 The Last Universal Common Ancestor

Perhaps the most remarkable aspect of the genetic code is its universality. Despite the apparent diversity of life, all use essentially the same genetic code[1] [45]. This code, known as the standard code was displayed in Fig. 1.7. This universality is particularly mysterious because there appears to be no compelling biochemical reasons for a particular genetic code. Thus, its universality is not a case of converging evolution as was demonstrated when the code was changed in vitro (see e.g. Benner [7]). Rather, the explanation has two parts. First, it is exceedingly difficult for an evolutionary processes to change the genetic code. Evolution occurs in small steps, but any change in the assignment of even

[1]Recently it was found that alternative codes do exist (notably, in human mitochondria). However, all the alternative codes are very similar to the standard code, varying in just a handful of codons. Studies of tRNA structure suggest that all the variants diverged from the standard genetic code fairly recently.

one codon to its amino acids would be greatly maladaptive, as it will cause simultaneous mutations everywhere in the genome where the codon occurs. This near-immutability of the genetic code and its seemingly arbitrary nature has lead Crick to call it a "frozen accident" [11]. Second, it is proposed that the standard genetic code has been inherited by all known organisms from a primordial organism known as LUCA (Last Universal Common Ancestor). The LUCA hypothesis is also supported by the discovery of numerous nearly identical biochemical processes that are found in all modern organisms (including the proteins and RNA genes involved in transcription, translation and replication). It is supposed that those structures were also inherited from LUCA. Current estimates based on geological clocks suggest that modern forms diverged from LUCA as early as 3.465 billion years ago, and probably earlier([72], [57]).

In recent years, it became possible to estimate the relationships between all existing organisms producing the "Tree of Life", Fig. 1.10. The existence of this tree is the ultimate vindication of Darwin who speculated about its existence in his "Origin of Species" (such a tree was the only diagram in the book [13, pp.140-1]). One major endeavor of modern evolutionary biology is to describe LUCA as well as the crucial evolutionary transitions after LUCA. Most likely, LUCA is not a single organism, but rather a colony of related organisms that exchanged genetic material and existed for a long period of time. Since LUCA must have contained many of the features shared by modern complex taxa, LUCA must have been a rather complex organism.

It is a commonly argued that LUCA was a heat-loving organism (a "thermophile") [74], meaning that its optimal growth temperature was above 40°C. The main evidence for this hypothesis is presented in Fig. 1.11. It can be seen that organisms closest to the root of the tree of life, i.e. to LUCA, are thermophiles or even hyperthermophiles. Mesophile organisms like plants and animals are thought to have lost adaptations to high temperatures sometime during their evolution. Moreover, it is suggested that early organisms were thermophiles because life itself may have evolved in high temperatures. High temperatures would have increased the rates of pre-biotic chemical reactions which led to the emergence of life. Habitats of high temperatures may have also included a temperature gradient that would have provided a source of energy for metabolism without photosynthesis. Another argument for thermophility of LUCA is based on analysis of the amino acids encoded in the

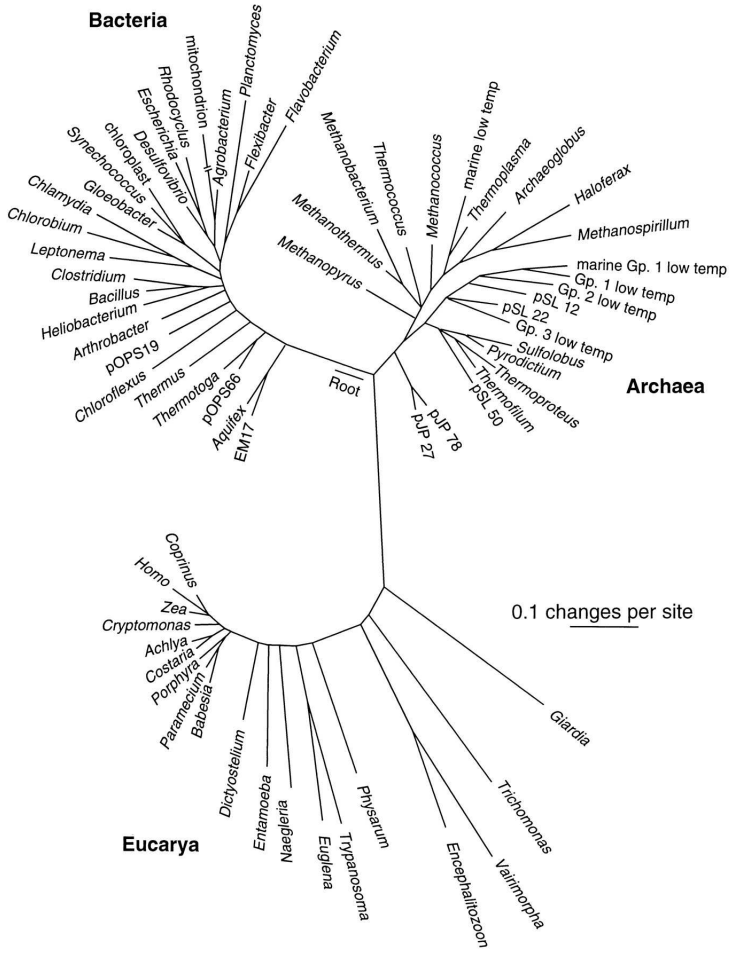

Figure 1.10: **Tree of Life**
The relationship between all organisms. All existing organism can be divided into three domains: bacteria, archaea, and eukaryotes. The tree is primarily based by comparing highly-conserved sequences of rRNA. These estimates are disputed and revised, especially due to uncertainty closer to the LUCA at the root. Image source: N.R. Pace [62]

standard genetic code. It is claimed [16] that if these 20 amino acids were used in proteins in quantities proportional to their degeneracies in the genetic code, then the proteins were heat-adapted.

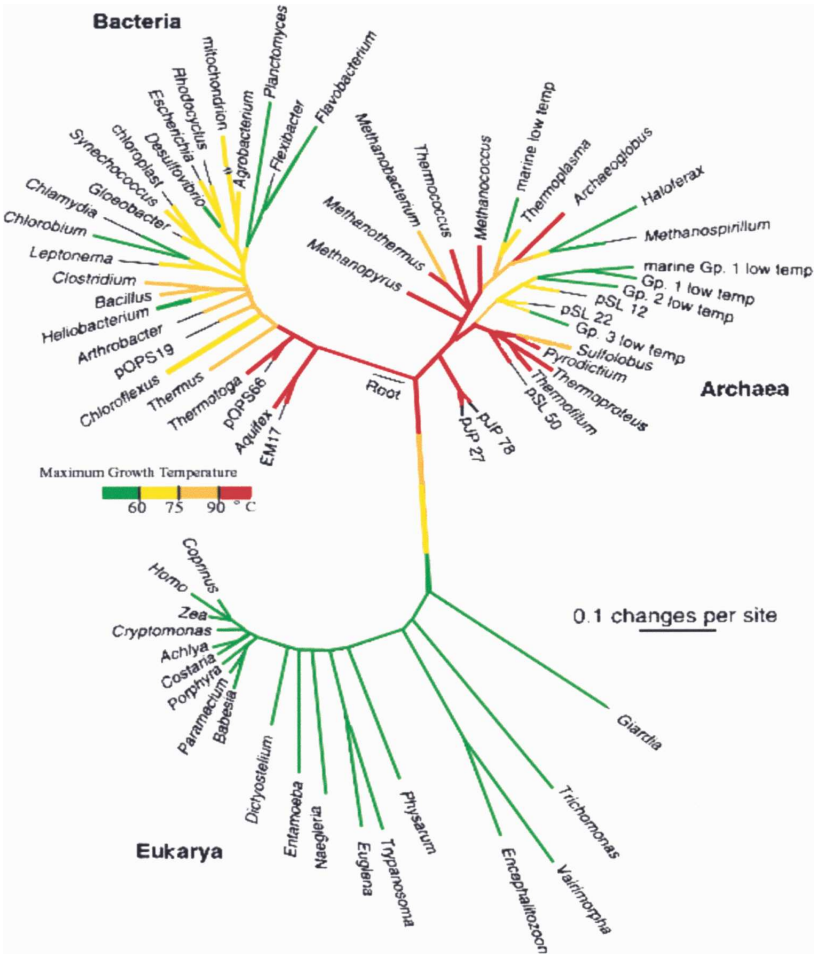

Figure 1.11: **Tree of life with optimal growth temperatures**
The relationship between all organisms with color used to indicate maximal growth temperatures. Notice that all mesophiles have hyperthermophilic ancestors while no hyperthermophiles or thermophiles have mesophilic ancestors with the possible exception of Archaeoglobus, a candidate for acquisition of hyperthermophily by gene transfer. Image source: N.R. Pace and Schwartzman *et al.* [74]

Despite the evidence from the tree of life, an argument could be made that LUCA and its ancestors were non-thermophiles ("mesophiles") [21, 67]. First, it is argued that the

evidence from the tree of life is inconclusive because the tree cannot be reliably calculated. One source of error is limitation of statistical techniques used to build the "phylogenetic" tree above - it is necessary to extrapolate how fast ancient organisms evolved from data about modern organisms. Another source of error are "lateral" transfers of genes between organisms, which is quite common in some types of unicellular organisms. It is also argued that the proteins that help modern organisms adapt to high temperatures are too sophisticated to have evolved by the time of LUCA. More generally, it is doubted that ancient organisms that were likely based on RNA could really survive in high temperatures. As was mentioned already, RNA (and many other bio-molecules) are very unstable in higher temperatures [53]. Moreover, Reanney [71] pointed out that high temperatures dramatically increase the rate of mutations during replication. An increase in environmental temperatures from $5°[C]$ to $25°[C]$ reduces the fidelity of the genetic machinery by between one and two orders of magnitude. This would have had a catastrophic impact on early organisms because it would have required them to have much shorter genomes (an issue modeled in Appendix A). In summary, there is support for evolution of life in either high or cold temperatures, but both views raise significant conceptual problems.

1.1.4 The RNA World Hypothesis

In the previous subsections we saw how complicated the genetic storage, transcription and translation systems are, and how these systems were inherited from an ancient common ancestor. Their complexity raises a fundamental paradox: the synthesis of proteins from information stored in DNA uses protein-based RNA-mediated machinery. Thus, there is a significant interdependence between DNA and proteins. The reading of DNA requires proteins, yet those proteins cannot be synthesized without the cell possessing a working protein-based synthesis machinery. Thus, usage of DNA requires proteins, but proteins cannot be made without DNA. Moreover, neither DNA nor proteins have significant non biochemical sources. This suggests that the modern genetic machinery evolved on top of the scaffolding of a simpler type of life and a simpler genetic machinery - machinery which evolution later expunged.

What was the ancient genetic machinery like, and what kind of organisms used it? It is thought that preceding modern organisms was an "RNA World" biosphere [30, 26]. Organ-

isms in the RNA world had no DNA. Rather, they used RNA genomes like some modern viruses (e.g. HIV). Furthermore, in the RNA world RNA molecules were also catalysts. The RNA world hypothesis envisions that the information-processing and metabolic machinery of earlier organisms has consisted exclusively of RNA. The RNA genome of early organisms contained the instructions for making RNA catalysts, called "ribozymes". It also contained the code for making replicators for copying the genome and reading it. This RNA World hypothesis has been studied extensively for the last twenty years. It is now thought that RNA organisms were not the earliest forms of life because the RNA polymers are themselves probably too complex to arise abiotically. Nevertheless, it is generally thought that preceding the modern biosphere was a less complex RNA-based biosphere, even if it was not the original form or life. For instance, it has been found that RNA can indeed act as an efficient catalyst in many reactions, most importantly in the polymerization reaction of RNA or DNA. Furthermore, it was shown that the crucial catalytic step of aminoacylation - the formation of chains of amino acids in proteins, is performed by the RNA part of the ribosome [59]. The RNA world hypothesis is very economical conceptually. As long as sufficient quantities and varieties of RNA chains are available, the RNA may begin functioning like a fully living organism - it can metabolize and self-replicate.

The RNA world hypothesis resolves the chicken-and-egg problem of modern DNA-protein biochemistry, by making it possible for proteins and DNA to evolve separately from one another and gradually devolve RNA. It is believed that proteins evolved because they have a higher structural diversity and thus higher catalytic efficiency compared to ribozymes. In a gradual process, they displaced ribozymes from most of their catalytic functions [83]. Like proteins, DNA evolved to do what RNA did before: store genetic information. Its advantage as a genomic molecule over RNA is its lower reactivity and lower tendency to break its own polymer bonds. As evidence for this transition, it is notable that in modern cells RNA is used to synthesize DNA with Uracil being the chemical precursor of Thymine.

One of the main lines of evidence for the RNA world hypothesis is the structure of the genetic machinery. As has been mentioned, RNA plays an important role in protein synthesis. For example, mRNA and tRNA transcribe and translate information encoded in the DNA. Moreover, rRNA (ribosomal RNA) catalyzes the actual production of pro-

teins. RNA also has a role in biochemical reactions as a "cofactor" (helper) to proteins. Comparison of divergent taxa suggests that the processes in which RNA is involved are very ancient. Yet, because in some bacterial taxa RNA has been partially replaced with proteins, its presence in modern organisms is not likely to be because it cannot be replaced with proteins. Rather, it stays mostly in those vital roles in which evolution strongly selects against any modification. Consequently, it is quite plausible that many of the functions of RNA are evolutionary relics - relics from the primordial RNA world [40].

An important detail about the genetic code is the question of who evolved first - proteins or DNA? For several reasons it is thought that it was DNA that followed proteins and not vice versa [24]. First, the synthesis of DNA from RNA involves a complex high-energy reaction and on chemical grounds it is thought that a reaction of this nature could not have been performed by ribozymes [68]. Second, despite the crucial role of RNA in facilitating translation, it is not involved in transcription. If RNA originally performed transcription because no proteins were available at the origin of DNA, it seems likely that some of this catalytic RNA should have been preserved in some critical roles, but it did not. In Chapter 3 we will rely on this conclusion to interpret our findings about the evolution of the genetic code.

The RNA world hypothesis may answer an important question - why are there four nucleotides in two base-pairs [84]. This question is particularly salient because experimentalists have successfully synthesized a large variety of nucleotides. Many of these are chemically stable. DNA polymerases are even able to catalyze the base-pairing of the new nucleotides. One simple explanation is based on the observation that RNA in an RNA world was used both as a catalyst and as a genetic store. The catalytic efficiency of RNA would have surely increased if it contained a more diverse set of nucleotides. However, this increase would have come at a cost of decreased replication fidelity. The more nucleotides there are, the greater the chance of incorrect base pairing because the nucleotides resemble each other more. This suggests that an optimal number of nucleotides exists and is small.

It may even be possible to show that the four nucleotide configuration is optimal. Szathmary [81] argues that while the increase in metabolic efficiency follows a power law in the number of nucleotides, the decrease in fidelity is super-exponential. Gardner *et al.* [28] have taken a different approach to find the same conclusions. They examined

different alphabet sizes *in silico* and investigated the ease with which RNA sequences evolve a predefined structure. They find that in conditions of low copying fidelity (as likely in an RNA world) a four-letter alphabet is superior to alternative alphabets. Notice that those considerations no longer apply for modern organisms because RNA is no longer extensively used in catalytic roles. Perhaps the most efficient way of encoding the 20 proteinogenic amino acids and a stop codon would be to use a genome consisting of just two nucleotides with five nucleotides per codon (giving $5^2 = 25$ codons). I will later show how the RNA world hypothesis may also explain why RNA uses U, C, A, G and not other four nucleotides. Since the explanation is based on error-tolerant coding, it will come only at the end of Chapter 2.

1.2 The Emergence of the Genetic Code

The genetic code has probably evolved in the RNA world, and its chemical implementation is based on RNA - the tRNA. However, the details are quite complicated because there was at some point a transition between an RNA genome that directly stores a ribozyme into an RNA genome that symbolically codes amino acids. The subsequent evolution of the genetic code has also been quite complex, subject to several evolutionary forces of different natures. However, if we are to draw conclusions from the genetic code, it is necessary to understand those forces.

1.2.1 The Emergence of the Genetic Code and Proteins

The genetic code arose sometime in the transition from the pure RNA world to a world of RNA and proteins - the "ribonucleopeptide" world. At first glance, it seems difficult to imagine how an organism whose genome encodes the instructions for ribozymes could use the genome to encode amino acids. There was no analogue to the genetic code in the pure RNA world. RNA genes encoded sequences of RNA that directly folded into three-dimensional catalytic structures. How is it then, that RNA came to encode triplets that gave instructions for attaching amino acids?

One well-developed theory for the emergence of the genetic code is the "Coding Coenzyme Handles" (CCH) hypothesis [82, 83]. It proposes that amino acids in the RNA world

acted as coenzymes to the ribozymes, helping improve catalysis. The amino acids attached to ribozymes thanks to special RNA-based "handles". The handles were the precursors of tRNA: one the one side they attached to an amino acid, and on the other side they attached to a site on the ribozyme. Attachment to the amino acid was based on stereochemistry, while attachment to the ribozyme was through base-pairing. The handles were soon able to attach an increasingly large palette of amino acids. The attachment of amino acids as co-enzymes was very advantageous: the amino acids provide a greater diversity of catalytic sites than is available in RNA alone. In fact, modern organisms use a variety of cofactors (e.g. metal ions) to further expand the catalytic efficiency [20].

The next important step was the evolution of special ribozymes that catalyzed the formation of amino acid chains. These were the ancestors of modern ribosomal RNA. With the arrival of ribosomal RNA, the handles (the tRNA) could be detached and reused. Free from the handles, the amino acids chains were able to fold into more and more complicated structures forming the first true proteins. The superior catalytic efficiency of proteins soon made it highly advantageous to use more and more amino acid coenzymes. According to the CCH hypothesis, organisms at this stage of evolution evolved a division of labor between two types of ribozymes. The first type encoded sites to which increasingly-long sequences of amino acids were attached - the messenger RNA. The second type were catalytically active ribozymes including those ribozymes that attached handles to amino acids and did other catalysis not yet performed by proteins. Many of the proteins at this stage had attached to them chains of RNA - chains that had used to be involved in a catalytic RNA molecule. These grew shorter and shorter with time. Over the biological history since the emergence of proteins, proteins took over most of the catalytic functions of ancient RNA, one function at a time. The large pieces of catalytic RNA present in modern cells like the rRNA remain there primarily because evolution of their catalytic pathways is highly constrained. Short sequences of RNA still attach to many modern proteins as helpers. These are especially common for proteins involved in the genetic machinery in part because the assist the proteins in binding other nucleotides. Thus the roles of the catalyst and its cofactor have been reversed.

Within the CCH hypothesis it is possible to explain why the genetic code uses codons of three nucleotides. Recall that according to the hypothesis, the code first emerged in

the form of handles that attached to individual amino acids. The handles (codons) would bind to sites (anti-codons) on ribozymes. Thus, it is conceivable that primordial organisms could have evolved doublet, quadruplet and other systems for the handles. There are at least three arguments for triplets over other nucleotide handles. Kazakov and Altman [43] have experimented with catalytic RNA sequences in the presence of magnesium ions (magnesium ions help catalyze base pairing in modern DNA polymerases). Remarkably, they find that the shortest catalytic RNA sequences are nucleotide triplets. Another important reason for the triplet code lies in the need for ribozymes to avoid accidental binding of handles [83]. Studies on binding strengths show that the binding strengths of free nucleotide sequences to their complements increases linearly with the length of the sequence. Thus, long handles (charged with an amino acid) would have the undesirable tendency of attaching to ribozymes at sites where the amino acid was not needed. Evolution thus opted for using shorter handles (triplets) whose attachment specific sites on the ribozymes would have required catalysts. Another argument for triplets proposes that triplets provide the optimal bond strength between the amino acid and the handle. A handle consisting of two nucleotides may be bound too weakly to the amino acid, while a handle four nucleotides of more would be bound too strongly and so be energetically costly.

1.2.2 The Structuring of the Genetic Code

In the past section we have seen how the genetic code emerged in the RNA world. In this section we will discuss various proposals about the subsequent evolution of the code.

The genetic code is mathematically a mapping of 64 triplets into 20 amino acids and stop codons. In general there are 21^{64} such mappings, but we need to exclude those that omit one or more of the 20 amino acids or those codes that lack a stop codon. Thus, we calculate using the inclusion-exclusion principle:

$$\begin{aligned}\text{Number of codes} &= (\text{All maps from 64 to 21}) - \sum_{i=1}^{21} (\text{All maps from 64 to 21 missing i}) \\ &+ \sum_{i,j=1}^{21} (\text{All maps from 64 to 21 missing i and j}) - \dots \\ &= 21^{64} - 21 \cdot 20^{64} + 21 \cdot 20 \cdot 19^{64} - \dots\end{aligned}$$

We can take the first three terms of this alternating and decreasing sequence as an approximation. Thus, the number of alternative codes is $3.3 \cdot 10^{84}$ with an error of at most $21 \cdot 20 \cdot 19 \cdot 18^{84} = 1.7 \cdot 10^{84}$. Here we used the alternating series theorem to find a bound on the error.

This calculation begs the question: why did LUCA and some of its predecessors choose the canonical code over the numerous alternatives? Of course, it is possible that other very different codes may have been used but for some reason their owners became extinct. Crick [11] has famously suggested that the genetic code is a "frozen accident". In other words, almost any code could have emerged during the early evolution of proteins. As soon as each codon was assigned to an amino acid (or stop codon), it could not evolve further - its evolution was frozen. Crick's argument is plausible because the change to the mapping of even one codon is tantamount to mutations everywhere the codon is used and is usually lethal. If Crick is right, then the genetic code is one of the most ancient parts of the genetic machinery and probably significantly predates LUCA.

To the surprise of many, dozens of alternative genetic codes have been found [45]. It is particularly significant that all the alternative codes are variations on the standard code. This indicates that they have emerged recently compared to the proposed age of LUCA. Moreover, the recorded changes occur in highly complex eucaryotic organisms like yeast. Thus, it is conceivable that even more extensive modifications were possible in simpler forms like LUCA and its predecessors. There are three leading theories as to which evolutionary mechanisms drove the structuring of the code [44]: the stereochemical hypothesis, the precursor-product co-evolution theory and the error minimization theory. It is quite likely that each of the theories captures one part of the evolution of the genetic code and all of the mechanisms acted during different periods of time [44, 25]. I will discuss the first two theories next, and dedicate the entire next section to the third.

The stereochemical hypothesis argues that the assignment of a codon to an amino acid was based on chemical affinity between the amino acid and some element of tRNA or its proto-tRNA. Recall that the CCH hypothesis for the evolution of proteins suggests that the earliest genetic code was based on stereochemical lock-and-key association between amino acids and tri-nucleotide handles. This reasoning is supported by evidence that both codons and anticodons are (sometimes) chemically attracted to the amino acids they represent [83].

Moreover, biochemists have evolved *in vitro* RNA sequences selecting for sequences with the greatest affinity for each amino acid [44]. In several but not all cases these sequences contained codons or anti-codons of the amino acid. This finding is startling because in tRNA the amino acid is bound to the so-called "acceptor stem", which is placed far from the anti-codon in the tRNA structure. Thus the chemical affinity is likely a chemical relic from the handle-based code - the precursor of tRNAs.

The available evidence for the stereochemical hypothesis does not show incontrovertibly that the standard code is the product of ancient stereochemical attachment. One problem is that the affinity found by experimentalists does not apply to all amino acids and could even be the product of chance. Moreover, amino acids were unlikely to attach to proto-tRNA at any important quantities without the assistance of special ribozymes (the precursors of aminoacyl tRNA synthetases - the enzymes that catalyze the charging of tRNAs with amino acids in modern organisms). It would have been advantageous for those ribozymes to have evolved in early organisms because they ensure that the attachment of amino acids to handles is specific enough. Thanks to these ribozymes, stereochemical attachment between amino acid and the handles may have became far less important. It is notable that modern aminoacyl tRNA synthetases do not recognize the correct tRNA by the anticodon and instead use other structural features of the tRNA. Thus, while stereochemistry may have played a role at the earliest stage of code evolution, it is not likely to have defined its final form.

The second theory about the structuring of the code, suggests that the biosynthesis of amino acids and the structure of the code co-evolved [89]. Indeed, it is likely that the earliest genetic codes used a subset of the 20 amino acids of the modern code. The co-evolution theory proposes that originally most of the codon space was divided between just a handful of amino acids that are particularly simple to synthesize. Since this codon space was likely based on chemical affinity, each amino acid would have large continuous blocks of codons. Next, evolution towards a more complex biochemistry would see new "product" amino acids being synthesized from the simpler "precursor" amino acids. Concurrently with the new chemistry, the genetic code evolved to encode the product amino acids. Several mechanisms were proposed. First of all, in many cases the product amino acid is chemically similar to its precursor. Thus, it could occasionally be attached to tRNAs (or

handles) by proto aminoacyl tRNA synthetases. Secondly, ribozymes could perform the synthesis of amino acids while they are attached to tRNAs (or handles). In both cases it would be advantageous for the organism to resolve the coding ambiguity, and this could occur as follows. If one of the synthetase genes was duplicated, each of the copies would be free to evolve greater specificity towards a particular amino acid - tRNA codon pair. Thus, some of the codons of the precursor would remain assigned to it, while others would be assigned to the product amino acid. The original contiguity of the codons would be retained but divided between precursors and products.

The chief support for co-evolution theory is the remarkable correspondence between biosynthetic pathways and codon similarity. Namely, codons related by single point mutations encode biosynthetically related amino acids beyond what is expected by pure chance [89]. Moreover, the mechanism by which codon space could be divided has been empirically observed: in some taxa, proteins synthesize product amino while its precursor is bound to a tRNA. Nevertheless, the theory has been disputed on several grounds. First, many randomized codes show the same single-base association between precursors and products. Second, the theory predicts that the evolution of aminoacyl tRNA synthetases and that of tRNA was congruent - which is not born by the evidence[44].

The discussion above begs the question: if the number of amino acids grew from a smaller initial set, why did the growth stop when it reached 20? It may appear that employing more amino acids in proteins would be advantageous because it increases the catalytic efficiency of proteins (just like increasing the number of RNA nucleotides would have increased the catalytic efficiency of ribozymes). However, increasing the amino acids palette would have come at the expense of a lower fidelity due to the greater complexity of the translation machinery. Namely, a greater number of amino acids would have made the attachment of amino acids to handles more error-prone. We have seen that in nucleotides the optimum is 4 because 6 or more would have come at the expense of lower replication fidelity. For amino acids the optimum is 20 and higher probably because poorly-assembled proteins are much less damaging to the organism then a mutated genome. Thus, the number of amino acids likely represents an evolutionary trade-off [58].

1.3 The Error Minimization Hypothesis

The Error Minimization Hypothesis proposes that the structure of the genetic code has undergone adaptation to minimize the impact of errors on the phenotype. Indeed, errors occur everywhere in the genetic machinery: there are replication errors during polymerase activity, chemical, thermal and radiation damage to DNA, errors in transcription and errors in translation. Because the fitness of organisms is determined by their phenotypes and not by their genotypes, genetic errors affect the fitness only if they lead to changes in the phenotype, and in particular, in the proteins being synthesized. The genetic code performs a crucial role here, converting genetic codewords into amino acids. If evolution could change the genetic code, it would be possible to reduce and minimize the deleterious effect of those errors.

The structure of the genetic code seems well-suited for the role of error minimization (Fig. 1.12). Codons coding for the same amino acids typically form blocks or boxes differing by just one letter (typically the third base and sometimes the first). Thus, when an error substitutes a nucleotide by another nucleotide, the original codon becomes a new codon such that (in many cases) they code for the same amino acid. These "synonymous substitutions" are possible because 64 codons code for just 20 amino acids plus the stop and so many amino acids have multiple codons in what is known as "degeneracy". Moreover, degeneracy most often occurs in the third position, which is exactly the position where most translation errors occur [63], at least in modern organisms. Beside degeneracy, another more subtle mechanism further improves the error-reducing quality of the genetic code: many nearby non-synonymous codons (e.g. ACG and AAG) encode amino acids with similar hydrophobicity properties. As a result, errors in the first position (like $UUC \rightarrow AUC$) and in the third position do not affect this most important property of amino acids and hence do not significantly change the shape of the folded protein [44].

I have recently performed a numerical simulation[2] highlighting the error-minimizing properties of the code (Fig. 1.13). In the simulation, I subjected a sequence of $10,000$ codons to errors. The encoded amino acids of the original sequence were compared to mutated sequences to find the average closeness in chemical properties, given by EX (de-

[2]The complete Matlab code for this simulation is found in Appendix C.

scribed in detail in chapter 3). As the error rate increased and the mutated sequence showed more errors, the EX value fell from its maximum. The fall was linear, for the following reason. When the error rate grows by a factor c, the mean number of codons that experience an error also grows linearly by a factor c (in the limit of low error rates), and with them, also the number of non-synonymous substitutions. But because the impact of errors on amino acid sequences is determined by the encoding, codes that have a lower average EX show a shallower decline in EX. The simulation also showed that the standard code is much better than random codes at error minimization, but most of its error-minimizing quality is due to the degeneracy in the third position, rather than the relative positions of the codons.

It is not clear from the available evidence which of the two error-prone processes - replication or translation had the decisive influence in shaping the genetic code. On the one hand, replication seems more significant because proteins are "disposable", i.e., a mistranslated protein would have caused little harm to the organism, while a mutation in a gene would have permanently impacted all proteins made of the mutated gene. On the other hand, translation - the making of proteins from mRNA, occurs far more often than replication. Also, as Woese pointed out [88], even a modest improvement to fidelity of translation in the genetic code would have been highly advantageous. If p is the average probability of correct translation of an amino acid in a protein consisting of n amino acids, then the probability of fully correct synthesis is p^n. Thus, even a slight increase in p would dramatically increase the fraction of correctly translated proteins. Moreover, because much of the genetic machinery is protein-based, there could be a positive feedback loop between correct translation and improved genetic machinery.

The Error Minimization hypothesis was originally proposed in 1957 [12] by Crick who argued that the genetic code is designed to eliminate frame-shift errors i.e. errors in which translation shifts by $1, 2, 4, 5, 7, 8 \ldots$ nucleotides. Not knowing the assignment of a single codon, he proposed the so called "comma-less" code - a code with the remarkable property of ensuring that errors in the reading frame would never occur (hence the name: reading of mRNA would not require knowing where each codon ends). This "beautiful" proposal [37] was shown incorrect once the codon-assignments were painstakingly found. In the modern form, the error minimization hypothesis was proposed by Sonneborn [79]

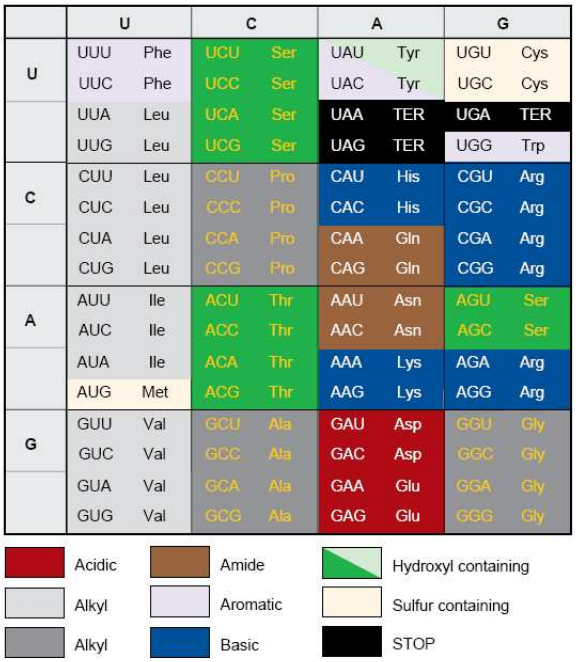

Figure 1.12: **Chemical Relatedness of Coded Amino Acids**
Amino acids that share structural or chemical similarity tend to be connected by single mutations in their codons. Here shading corresponds to "polar requirement(PR)" - a measure of hydrophobicity: lighter shades (black text), $PR < 6$ (hydrophobic); medium shades (yellow text), $PR = 6 \ldots 8$ (medium); darker shades (white text) $PR > 8$ (hydrophilic). Thus, amino acids whose codons have U at the second position tend to be unusually hydrophobic; those whose codons have A at the second position tend to be hydrophilic. The stop codons are black (but have no hydrophobicity value). Image source: Knight *et al.* [44]

and Woese [88]. Unlike Crick, they focused on substitution errors. Sonneborn and Woese argued for minimization of the phenotypical effect of substitution errors in replication and in translation, respectively. Their pioneering work was soon overshadowed by discoveries of alternative genetic codes, and has been largely forgotten until 1991. That year computer simulations offered the first large-scale statistical study of error minimization [35].

In this thesis I develop a research direction that has not yet been explored to date.

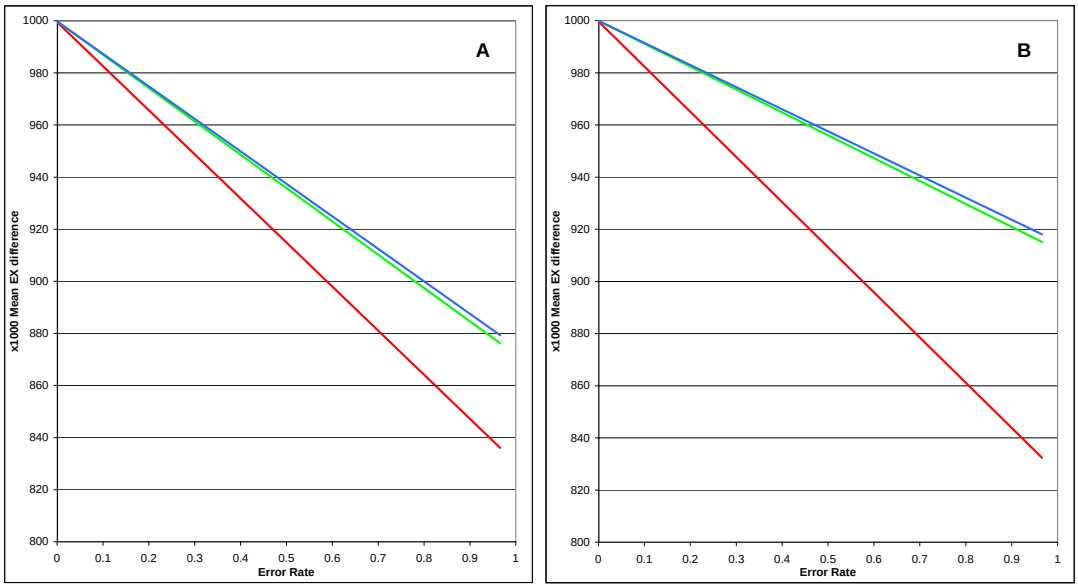

Figure 1.13: **Reduction by the genetic code of the phenotypical impact of errors** Because the standard code (blue) is very efficient at error minimization, it performs much better than a randomly generated code with the same 20 amino acids (red) at any given error rate. The standard code was only slightly better than codes of the same structure like the standard code but with the amino acids relabeled (green). In (A) errors were assumed to strike all the three nucleotides in the codon uniformly, while in (B) the first and third positions were assumed to experience 5- and 10-times as much errors as the second position, respectively. Comparison of (A) and (B) shows that the standard code is much better at reducing the impact of this skewed distribution of errors. To find the red and green series we took the mean performance of 100 codes, however, the calculated EX values still fluctuated by up to 5/1000 about the lines. Method: The test sequence was randomly generated so that all nucleotides were equally likely, and all codons were equally likely as well. In calculating the mutated sequences, all transitions errors were assumed to occur with the same probability which was four times greater than the rate at which all transversion errors occurred. See Appendix C.

Namely, I use the error minimization hypothesis as a tool to study the RNA world through the structure of the genetic code. Firstly, the error minimization hypothesis may give

insight into the RNA-world genetic machinery: If the genetic code has been shaped by the pressure to minimize errors, then its structure reflects the pattern of error rates and hence may tell us something about the genetic machinery of ribo-organisms. Secondly, the genome composition of the primordial organisms (i.e. the probability of each of the four nucleotides) affects the absolute probabilities of errors and is therefore imprinted in the structure of the genetic code. Hence, studying the structure of the code may allow one to "reverse-engineer" the error-minimizing evolution and to find the nucleotide frequencies. The interest in nucleotide frequencies comes from their relationship to environmental temperatures. Recall that modern organisms living in elevated temperatures use relatively more of the nucleotides G and C in their RNA [27]. This adaptation helps stabilize double-stranded RNA molecules in higher temperatures because G and C are have a triple hydrogen bond, unlike the double hydrogen bond in U and T. Thus, elevated G+C content of RNA is a marker of thermophiles. Consequently, by investigating the error-minimizing properties of the genetic code, it may be possible to find the environmental temperatures prevalent at a very early stage of evolution. That is the period when proteins have began displacing ribozymes, but organisms were still simple enough so that the genetic code could undergo extensive evolution. The attempt to reconstruct the nucleotide frequencies from the genetic code is not unlike palaeontological studies that attempt to determine the habitat of a long-extinct reptile by measuring its fossilized bones, except that here the fossil is the genetic code and the measuring instruments are information-theoretic. In summary, if the error minimization hypothesis holds true, then by studying the genetic code it may be possible to find whether ancient organisms were thermophiles.

The error minimization hypothesis is also interesting mathematically. If the genetic code is error-minimizing, then it is one of the most important examples of error-tolerant coding in biology. Error-detecting and error-correcting codes are usually found in high-tech communication systems. As we shall see, the genetic code represents a special type of coding unlike other codes considered in information theory.

1.3.1 Conceptual and Statistical Arguments

There exist both conceptual and statistical arguments for the error minimization hypothesis. The former highlights the importance of error minimization in the early evolution of

life, while the latter considers the genetic code against alternative codes and shows that chance alone cannot explain the error-minimizing properties of the standard code.

Probably the most important of the conceptual arguments is Eigen's paradox [19, 49], which shows that error minimization is a chicken-and-egg type of a problem. It highlights the strength of the evolutionary pressure for error minimization. Consider again the RNA world. Reliable replication by ribo-organisms would have required complex enzymes to do the replication with high fidelity. However, the instructions for those complex enzymes presumably require long RNA sequences, and hence a long genome. Yet, the genome cannot be reproduced correctly without reliable enzymes. How did then, the primordial organisms evolve both? For a time, it was thought that the answer comes from natural selection. Namely, although many organisms would have too many errors in their genomes to survive, chance alone would ensured that there would be sufficient number of those that have no errors. Then evolutionary competition would cull mis-replicated organisms allowing only those that have few errors to propagate and evolve. Notice that it would not eliminate the evolutionary force for error minimization, because selection would sustain the population and not individual organisms. Thus, each individual would gain a great increase in fitness if it could decrease the error rate it experiences, by e.g. evolving an error minimizing genetic code[3]. The greater the error rate, the greater the selective pressure and the greater the gain.

Moreover, it turns out that, as Eigen showed[4], that selection is not a panacea to high error rates - it cannot qualitatively change an inverse relationship between the replication fidelity and the genome length that can be sustained by evolution (Eqn. 1.1). More precisely, if σ is the selective advantage of the primordial RNA organism (sometimes called "*Riborgis eigensis*") over its mutants, then the maximum length of the genome L_{max} is related to the probability of error per nucleotide, μ, as:

$$L_{max} < \frac{\ln \sigma}{\mu} \tag{1.1}$$

Recent studies of ribozymes [49] have estimated both the replicative fidelity μ and the selective advantage σ. Based on their findings, they propose tentatively that a ribo-

[3]I provide a mathematical model of its effect on fitness in subsection 2.2.4

[4]A more detailed treatment of the problem is provided in Appendix A

organism may have been able to sustain a genome of as much as 7000 nucleotides, which might be enough for a viable organism. Studies of modern organisms suggest that about 200 genes are required for a minimal organism [29] of these, as few as 100 are needed in a riboorganism since it does not require genes for translation and protein synthesis. Because an RNA-gene may be as short as 70 nucleotides, 7000 might be just sufficient to encode the minimal set of 100 genes. It is notable that the hypothesis ribo-organism would have a genome far shorter than the shortest genome of a known extant organism, *Nanoarchaeum*, that has a genome $490kb$ (kilo base-pairs) long. Yet, even if the organism could survive, could it evolve further? Jeffares *et al.* estimate that a ribo-organism with protein-synthesizing machinery would have to have a genome of at least $10 - 15$kb. It would be even harder to evolve the complexity found in LUCA, which had at least 572 genes [54]. We know from modern organisms that those of them that have error rates as high as the upper bound of Eqn. 1.1 evolve various error-minimizing strategies such as multi compartmental genomes [69], duplicate genomes and recombination [70].[5] Thus, it is likely that organisms that evolved the genetic code would have been under strong evolutionary pressure to introduce error-minimizing modifications to the code, especially as is likely, those modifications would not have required making the genome longer.

The existence of organisms at the edge of sustainable genome length was not a stable evolutionary equilibrium. Instead, it is thought that a virtuous cycle for error minimization operated throughout the long transition from the RNA world to LUCA. This cycle, known as the Darwin-Eigen cycle, was driven by the fact that higher fidelity allows for longer genomes and hence supports the advent of novel adaptive traits. However, competition would drive organisms back towards the maximum allowed genome, prompting selection for even higher fidelity and so on. Likely, it was this cycle (Fig. 1.14) that over time caused error rates to decrease by seven to eight orders of magnitudes [66]. The cycle is the leading explanation for several breakthrough evolutionary transitions: one from RNA genomes to the much more stable DNA genomes, and the emergence of proofreading mechanisms in both replication and translation, Fig. 1.15. Although the replacement of ribozymes by proteins was driven by catalytic rather than error-minimizing considerations, the subsequent

[5] HIV is a notable case and has all three features, which is unsurprising given the fact that it is RNA-based and experiences an error rate of $10^{-3} - 10^{-4}$ per nucleotide.

evolution of the genetic code was likely strongly influenced by error-minimizing forces.

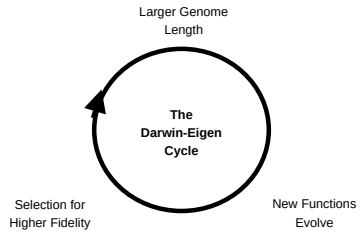

Figure 1.14: **The Darwin–Eigen Cycle**
This cycle acted over a period of hundreds of millions of years starting at the RNA world or even before and effectively ending by the arrival of LUCA. Because the genetic code has existed for much of that period and was subjected to error-minimizing pressures, there was plenty of time for evolution to improve its error-minimizing properties. Image adapted from Penny *et al.* [64]

The genetic code structure seems to be adapted to error minimization: synonymous codons mostly lie in boxes, the most common errors (transitions) are often harmless and similar amino acids lies close to each other in the space of codons. Nevertheless, it is argued by critics of error minimization that this structure has emerged as a by-product of other phenomena. It was Crick who noted that the degeneracy in the third position (e.g. UUU and UUC) is due to a "wobble" in how tRNA binds to the codon. Thus, what appears to be an adaptive property in the code might be a by-product of codon-anticodon chemistry. Even the simple stereochemical theory, namely that amino acids are assigned to codons to which they have chemical affinity, induces some degree of error minimization by coincidence: similar codons have may tend to attach to similar amino acids. More recently, proponents of the co-evolution theory argued that during the process of reassigning codons of precursor amino acids to product amino acids, contiguous blocks of codons became assigned to amino acids that tended to be chemically related and hence, the code appears to be error-minimizing, but did not specifically evolve for error minimization.

In response to those criticisms, proponents of the error minimization theory have performed statistical studies in an attempt to show that the genetic code is exceptionally efficient in minimizing errors. Thus, they attempt to statistically refute the hypothesis

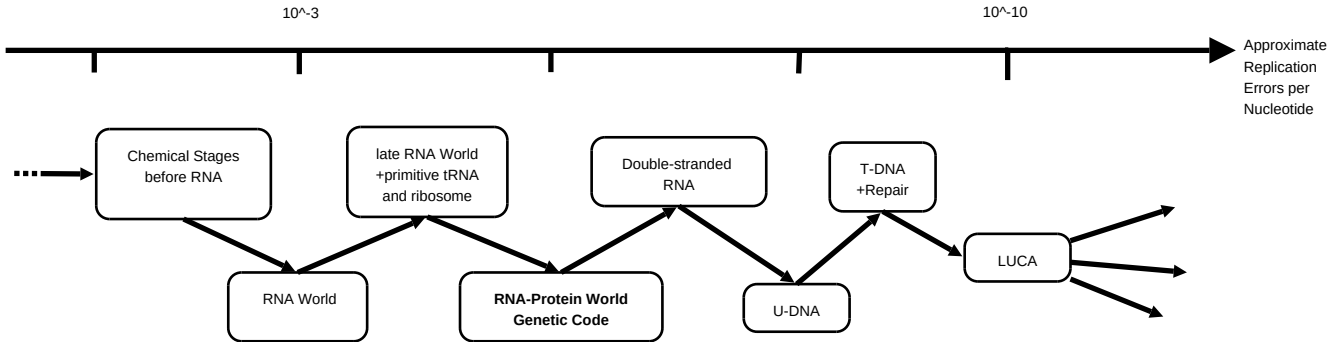

Figure 1.15: **The Coevolution of Error Minimization and Longer Genomes** The evolution of modern organisms from the RNA world coincided with rapid progress in achieving error minimization in the genetic machinery. Also, the "U-DNA" stage refers to a hypothesized DNA genome which retained the Uracil nucleotide from RNA. Its replacement by the modern T-DNA (that uses Thymine) allowed the cell to evolve mechanisms for correcting deamination errors - the spontaneous degradation of Cytosine to Uracil. Image adapted from Penny *et al.* [64]

that the error-minimizing properties of the code are random or incidental to other processes. All such studies have followed the same general approach [25], starting with the pioneering work of Alff-Steinberger [1] (1969) and continuing through studies in the last several years: First, define the set of possible alternative genetic codes. Second, quantify the susceptibility of a genetic code from that set to errors, whether mutations or translations. Finally, generate a large sample from that set of codes and find where the standard code lies relative to the rest of the sample according to that measure. While the exact result of these studies depends on the computational details, it is often found that the genetic code is better than a million (e.g. Freeland *et al.* [23]) or even a billion other genetic codes (e.g. Gilis [31]). Thus, there is evidence that not only is the genetic code error-minimizing, but that it is exceptionally so.

Let us consider the ingredients one at a time. As was mentioned earlier, there are over 10^{84} alternative genetic codes. Some of these codes might be highly efficient in minimizing errors, perhaps even more efficient than the standard code. Yet, it is almost certain

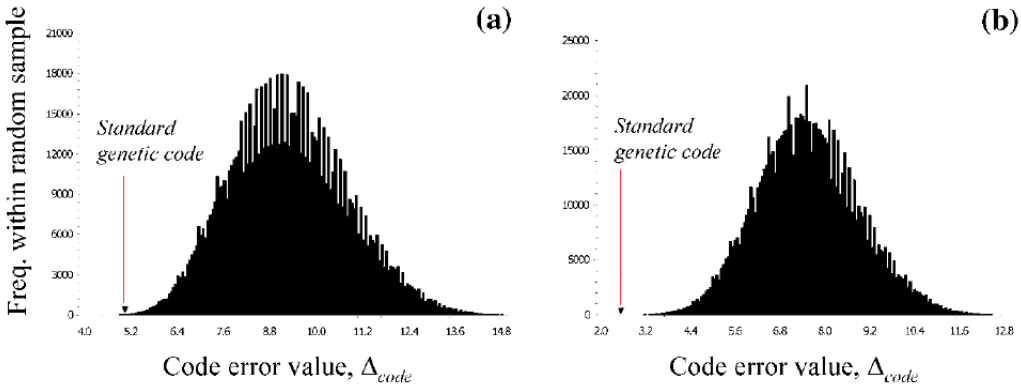

Figure 1.16: **Comparison of the standard genetic code with 1 million alternatives** While (a) does not include a bias for errors, (b) does, resulting in a dramatic increase in the optimality of the code compared to the alternatives. The horizontal axis represents the Δ_{code} function - the average change in the hydrophobicity when a substitution error occurs during translation. In both cases, the standard code is at the extreme left of the distribution, and is indicated by a red arrow. Image source: Freeland *et al.* [25]. Red arrows were added for clarity.

that large domains of this set remained untried because there are so many codes - more than there are atoms in the visible universe, estimated at about 10^{78}. Moreover, the error-minimizing code emerged from an evolutionary process and hence was "historically constrained" by the shape of its precursor code. These precursors probably included less than 20 amino acids and were based on stereochemical affinity. Thus, the evolution of the error-minimizing genetic code was constrained in some poorly-understood respects and not all of the 10^{84} alternative genetic codes were accessible for evolution. Studies that compare the genetic code to alternative code need to take into account some of these constraints. Typically, the question is addressed (not fully convincingly) by considering only the $20! \approx 2.4 \cdot 10^{18}$ codes that preserve the block structure of the genetic code (with the stop codons fixed at their usual places). Thus, the alternative codes are simply permutations of the amino acids seen in the code. Moreover, since 20! is far too many codes, it is necessary to randomly sample from this space.

A crucial and somewhat controversial role in such investigations is played by the measure of susceptibility or "fitness" mentioned above, which evolution purportedly optimized. Whether the standard code is indeed closer to the optimum of this function than randomly generated codes may depend heavily on how it is calculated. Most commonly, this fitness function represents the average (or root mean square) chemical difference between two amino acids when a substitution occurs in the genome. Such an average includes, therefore: first, a probability distribution of the different substitutions; second, a measure of difference between chemical properties of amino acids; and third, a probability distribution of the different codons in the genome.

In regards to the first, it is generally assumed the four transition errors occur more frequently than the eight transversions. This proposal is supported by data taken from translation errors in modern organism [51], although the difference is not as pronounced in mutations. Some theoretic considerations [86, 77] support the view that this difference would be present in the primordial genetic machinery. The difference may have been more pronounced in primordial organisms because they lacked the proofreading mechanisms for replication and translation seen in modern organisms.

The probability distribution for the errors may also account for the dependence of the errors on where in the codon the errors occurs. As can be easily seen, the genetic code is particularly adapted to errors in the third position in the codon. Coincidentally, translation errors do occur much more frequently in the third position [63], while mutation are position-uniform.[6] When this bias is taken into consideration in the fitness function, there is up to a two orders of magnitude improvement in optimality compared to random codes [25]. The inclusion of this bias in the fitness function is controversial because the patterns of translation errors in modern organisms may have evolved comparatively recently in response to the particular structure of the genetic code. Such a pattern need not have existed in primordial organisms and may have co-evolved with the genetic code. While very early codes might have had no have such bias, later organisms may have been able to reduce the overall error rate by simultaneously decreasing the error rates in the first and second positions, increasing the error rate at the third position and adapting the genetic

[6]The two types of errors are also different in that mutations affect DNA while translation uses RNA, although this distinction may not have been present in the past.

code to this pattern. I hope to bring some computational evidence to this question some time in the future.

In regards to the measure of chemical difference between amino acids, good data has not been available until very recently [90]. Amino acids differ in properties such as hydrophobicity, volume, and carbon content. Traditionally, one of two approaches was taken, neither entirely satisfactory. Either some experimental measure of hydrophobicity was used (because hydrophobicity has a strong effect on protein folding in the aqueous solution of the cell), or chemical difference was inferred from comparison of amino acids sequences of highly-conserved homologous genes (the argument being that evolution would allow amino acids to be substituted in proportion to their chemical similarity). The flaw in the first approach lies in the fact hydrophobicity is not the only important difference between amino acids, while in the second approach, the data is "contaminated" by the structure of the genetic code. Namely, some amino acid substitutions would occur more frequently in the phylogenetic record than others simply because of the proximity of amino acids in the space of codons. Recently (2005), Yampolsky *et al.* [90] have derived a better measure of chemical similarity by combining painstaking laboratory studies. These studies compared the enzymatic efficiency of pairs of proteins, such that in each pair the original protein was compared to a protein where a specific amino acid was substituted. To my knowledge, this measure has not yet been used in validating the error minimization theory.

One of the clear flaws in all existing studies is the assumption that all codons occur with equal probability. If modern organisms [46] are any guide, then primordial organisms would have displayed preferences in both codon and amino acid usage. Codon usage is also affected by the frequencies of nucleotide in the genome. Recall, organisms do not always have uniform probabilities of the nucleotides and in particular, RNA in thermophiles shows elevated G+C. If the genetic code evolved in RNA based thermophiles, then codons with G or C would tend to occur more frequently. Because the genetic code is quite efficient in minimizing the impact of errors in all codons, I do not expect a skew in the probability of nucleotides to have a decisive effect on the conclusion of those statistical studies.

The type of evidence considered so far compared the standard code to the null model of random codes. There exists another type of studies based on computationally searching for a genetic code optimizing a measure of fitness. The core idea is that if the genetic

code is highly optimized, it would be hard to obtain an alternative genetic codes of the same degree of error minimization. The actual results are of those studies are mixed and controversial. Freeland *et al.* [25] find that under a biologically realistic fitness function, it is indeed very difficult to obtain alternative codes. However, other studies did find such codes [17]. It must be noted that these "engineered" codes are methodologically suspect. First, the results may depend on the choice of the fitness function, and indeed Freeland *et al.* suggest that in all such studies the fitness function was not chosen correctly (*e.g.* did not consider the elevated rate of transition errors compared to transversions). Moreover, the genetic code may have been subject to additional and unknown structural constraints that are not included in the fitness function. Thus, codes one may engineer may in fact be excluded by those constraints. Second, engineering alternative codes in many cases uses mechanisms that are disconnected from actual evolutionary processes (discussed later in this chapter). For example, Di Giulio and Medugno [17] use an algorithm that swaps blocks of codons, despite the fact that such swapping is unlikely to have occurred. Another example is Judson *et al.* [42] who model the evolution of the genetic code with a genetic algorithm, despite the fact that speed of genetic algorithm is highly dependent on its implementation [56] Thus, none of the critiques puts the error minimization hypothesis into any serious question.

So far, we have considered the minimization of errors that interchange amino acids. Error minimization may also explain the placement of the stop codons in the genetic code [41]. The cause is not the substitutions errors, because these do not have a strong tendency to occur in particular codons. Rather, on rare occasions during replication, the DNA polymerases make deletion errors, and some codons are "hotspots" for such errors. In fact, deletions occur slightly more often at the stop codons then at other codons. Coincidentally, although deletion errors are highly deleterious, they cause the least damage when they occur at a stop codon: deletion in a stop codon occurs, by definition, at a point where the protein has already been synthesized. In fact, continuation of transcription beyond the stop is quite harmless, as it leads to the attachment of an amino acid chain that typically does not interfere with the function of the protein. Moreover, the chain is often quite short because the deletion causes the RNA sequences downstream to be misread as containing stop codons. The consequences of a deletion error in the middle of a gene are

far more grave: such an errors leads to a shift in the reading frame of all the nucleotides downstream and produce a completely damaged protein fragment. Although modern DNA polymerases are not the same as the polymerases that worked with primordial genomes, they are able, for example, to catalyze RNA base-pairing and may share the deletion error hotspots of the RNA polymerases that existed during the evolution of the genetic code. Thus, the placement of stop codons may be an adaptation to minimize the deleterious effects of deletion errors.

We see then, that genetic errors were in the past a fundamental obstacle to evolution and the genetic code evolved to minimize their impact. Some authors have gone further and argued that not only were errors being minimized, the genetic code is structured to increase the rate of evolution, by providing for a faster search in the space of genotypes [92]. This now seems unlikely, because primordial organisms were subjected to more than enough errors. We must recognize that the error regime in which primordial organisms lived is very different than the mode of existence of modern organisms. While for modern organisms, errors can be both beneficial - bringing a degree of variability that is vital for evolution - and deleterious (destroying genes and leading to e.g. cancer), for primordial organism errors were a fundamental evolutionary obstacle.

1.3.2 Codon Reassignment Mechanisms for Error Minimization

Evolution for error minimization requires mechanisms by which codons could take alternative amino acids without causing death. Without such mechanisms, there would be no variation upon which selection for error minimization could act. Yet, the very existence of such mechanisms is surprising, and understandably so, given that the code has remained unchanged while great many transformations took place in the planet's biosphere.

There are several well-understood mechanisms for the re-assignment of codons [75]. One mechanism, proposed by Osawa and Jukes [60] proposes that large fluctuations in the codon content of genes has allowed the tRNA to evolve. They observe that the codons used by various taxa strongly correlate with the overall nucleotide content of their genomes. Since the G+C content of the genomes ranges from approximately $25\% - 75\%$, codons involving a rare nucleotide can almost disappear from the genome. This outcome can set in motion codon reassignment. The tRNA encoding the codon in question may stop functioning

while at the same time another tRNA may "capture" the codon. Eventually, when the codon reappears in the genome it would map to a different amino acid. Such a mechanism predicts that codon capture would predominantly occur by nearby tRNA, which is indeed the case. However, this mechanism predicts that codon loss would occur most frequently when it involves rare nucleotide - in conflict with recent studies of mitochondrial genetic codes [47].

Another mechanism, was proposed by Woese [88]. It may have acted during the earliest evolution of the genetic code and it could be termed "ambiguity reduction". Unlike in modern organisms, simple organisms at the dawn of protein age must have had an error-prone genetic machinery. and in particular, single codons bound to two or more different handles (or proto-tRNAs). The resulting ambiguity afforded evolution an opportunity to shuffle some of its genetic code for error minimization or otherwise. For instance, selection may disable one of the tRNAs such that the remaining tRNA (and codon) is better at minimizing errors. Notice that because the genetic machinery was in general error-prone, the organisms were adapted to low-efficiency poorly synthesized proteins. Moreover, because they were simpler than most modern taxa changes to the code would have affected comparatively fewer genes. As a result, changes to the genetic code were much less deleterious than such changes in modern organisms.

Another codon reassignment mechanism was proposed by Schultz and Yarus [73]. Their mechanism, like the mechanism of Woese above is based on translation ambiguity. Unlike Woese, they propose that the translation of a codon may become ambiguous because of a deleterious mutation in its cognate tRNA (the original tRNA) or a mutation in a near-cognate tRNA. Despite the mutation in the near-cognate tRNA, it need not lose its ability to translate its own codon (because the binding of the codon to tRNA depends on the catalytic activity of the anti-codon stem and only partially depends on Watson-Crick base-pairing between the codon and the anti-codon). In this ambiguous translation state some malformed proteins would be synthesized, but overall the mutation in tRNA may well be non-lethal. Nevertheless, there will be substantial evolutionary pressure to reduce the translation ambiguity either by a reverse mutation or by further modifying the cognate tRNA, in effect, by changing the genetic code.

The Schultz and Yarus mechanism is supported by the discovery of species where the

decoding of certain codons depends on context [73, 83]. Moreover, the most common patterns of mistranslation correspond to patterns of codons reassignment [47], suggesting that ambiguous mistranslation is commonplace. Furthermore, there are several mutations known to alter codon recognition by tRNA. On a conceptual level, Schultz and Yarus reassignment is far more likely to occur than the Osawa and Jukes mechanism. First, it does not require a codon to disappear from the genome. Second, it does not require independent events - the loss of a cognate tRNA and the mutation of a near cognate - to occur coincidentally. Rather, it proposes that mutation leading to ambiguous translation would be deleterious and hence start evolution towards disambiguation.

Which of those mechanisms - Osawa and Jukes, Woese, or Schultz and Yarus - is most likely to produce error minimization? Osawa and Jukes mechanism poses a problem for the error minimization hypothesis: the evolution of an error-minimizing genetic code from a code that is not error-minimizing requires large modifications. However, successful error minimization would be difficult because the large modifications in question would involve large fluctuations in genome content - fluctuations that will disrupt the error-minimizing evolutionary pressure. A genetic code that is efficiently error-minimizing for a high G+C genome is not likely to be as efficient for a moderate G+C genome. Fortunately, the Woese or the Schultz and Yarus ambiguity mechanisms do not require such changes in G+C. Of the two, the Woese mechanism proposes a gradual increase in codon specificity and so it precludes large scale changes in the genetic code. It is possible that before evolution for error minimization, the genetic code was already quite efficient at error minimization. Both the stereochemical and the co-evolution theories predict that nearby codons would code for related amino acids. Nevertheless, as pointed out by Freedland *et al.* [25], the genetic code is much much better than a random code at error minimization. It seems implausible that this could have been achieved through mere tinkering of a pre-existing code. A more plausible scenario is that the Schultz and Yarus mechanism acted on the genetic code for a prolonged period of time.

It may be said that the debate over mechanisms of code evolution is unnecessary because many examples for changes in the genetic code are known in mitochondria. In fact, mitochondria are excellent examples of code evolution in organisms with short genomes (about 14k base pairs), similar in this respect to primordial organisms. It is likely that the

number of coding changes seen in mitochondria provide a very low bound on how many such changes occurred in primordial organisms, for several reasons. First, coding changes in mitochondria are suspected to be driven by some unknown process or perhaps simple drift, rather than by error minimization or genome compactification [47]. If such pressures were present in strength, many more coding changes might have been present. Second, mitochondria are not adapted to high error rates, and hence would experience considerably more fitness loss than primordial organisms if coding becomes ambiguous (as a prelude to coding change). If the mitochondrial code can change, then surely the code of those far more primitive primordial organisms can change as well. Moreover, because those organisms were not only simpler but also adapted to high rates of error, their genetic code could evolve quite frequently and extensively, potentially allowing the code to evolve a very high degree of error minimization.

We do not known exactly how this evolution proceeded, but it is not hard to think of an evolutionary process that would produce the standard code we see today. Likely, the genetic code experienced large scale changes in a stochastic fashion, with some variations in the code more likely to be fixed than others. The changes that tended to be fixed were mostly those that improved the ability of the code to minimize the most frequent errors - the transitions. However, minimization of transversion error also improves fitness and hence it did occur occasionally. Still, it is likely that the tendency of the code to evolve error minimization to a given error would be related to the probability of that error. The greater the probability of the error, the greater the tendency. Over the course of its evolution, the structure of the code was transformed from one having poor error minimization to all errors, to one with a much better error minimization of all types of errors, both transitions and transversions.

1.3.3 The Error Reduction Hypothesis

We have seen how the genetic code may have been structured to minimize the phenotypical effects of errors. However, strictly speaking, it is unlikely that the genetic code represents the optimal adaptation to the fitness landscape of errors. Firstly, the number of alternative codes is far too large for evolution to try any substantial part of, especially considering the fact that the genetic code is highly conserved. Second, the fitness landscape for the

genetic code is "rugged" because codon assignments are highly interdependent in their error-reducing ability. Namely, the error-reducing qualities of a single codon are determined by the $27 = 3^3$ neighboring (distance 1) codons. For example, there are no intrinsic reasons for a particular codon to code for Leucine. Rather, the fitness of such an assignment is high only when nearby codons code for Leucine or a related amino acid. This observation implies that many promising evolutionary pathways may be blocked because they require codon reassignments that decrease the error-reducing ability of nearby codons. Thirdly, the adaptation of the genetic code to errors occurred simultaneously with the lengthening of the primordial genomes. Thus, the pace of evolution gradually slowed since codon reassignment grew increasingly rare due to the lengthening genome. Hence, if the genetic code is situated at an optimum, it likely to be a local one.

The above argument suggest that the genetic code is not the optimal error-minimizing code. Rather, it is almost certain that the code is a highly efficient "error-reducing" code - a code found near the top of the fitness landscape which is perhaps locally optimal. This "error reduction hypothesis" is consistent with both conceptual and statistical studies presented earlier. Moreover, unlike the error minimization theory above, the error reduction hypothesis proposed here is consistent with the existence of some codes that are better at error reduction than the standard code.

The error reduction hypothesis does not preclude us from proceeding with trying to reconstruct the primordial evolutionary pressures that acted on it. Indeed, we have noted that the statistics of the errors are likely imprinted in how efficient the genetic code is in reducing their impact on the organism. A mathematical relation that quantifies this idea would be proposed and calculated in Chapter 3.

Chapter 2

Coding and Information Theory

In chapter 1 we reviewed the genetics background necessary for the reconstruction project. In this chapter we will develop the necessary ideas of information theory. The nature of information theory is probably best described in Shannon's pioneering work[76], as follows:

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have meaning: that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one selection from a set of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

The relevance of information theory to genetics has been noted very early in the history of both fields (e.g. Sinsheimer 1959, Golomb 1962)[78, 32].

There are obvious parallels between the information-theoretic setting (Fig. 2.1) and the replication and protein synthesis machineries. In the replication machinery (Fig. 2.2), the sender of information is the cell about to replicate and the recipient are the two daughter cells. Replication includes a stage where single stranded DNA is complemented. Although in principle a DNA single strand contains all the information of doubled-stranded DNA,

the information is not yet usable. The transmission of information is only complete when polymerases complement the strand to form a double strand. This processes is error-prone, although depending on the strand, the daughter cell may actually not experience the errors but pass them to the next generation (see Chapter 1).

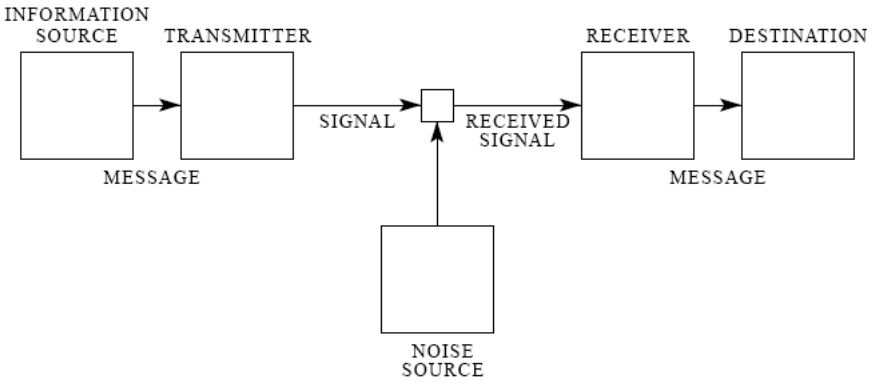

Figure 2.1: **The basic communication problem**
This illustration of information flow in a communication channel can be generalized. The "information channel" need not involve displacement in space, and could instead represent storage for a duration of time. Image source: C.E. Shannon[76]

The protein-synthesis machinery is more complex (Fig. 2.3). First, information flows from DNA to protein via mRNA. Thus, we can identify two channels: the DNA-mRNA channel of transcription and the mRNA-protein channel of translation. Second, both channels involve a change of alphabet. In the process of transcription, the DNA alphabet (T, C, A, G) is converted into the RNA alphabet (U, C, A, G). Moreover, since transcription is based on base-pairing, T, C, A, G actually become A, G, U, C and because this process is primarily information-theoretic rather than biological, the DNA sequence and its RNA transcript are said to be "the same", despite the fact that the are chemically different. Following transcription, the genetic information in the form of codons is translated - a lossy mapping - into the alphabet of amino acids.

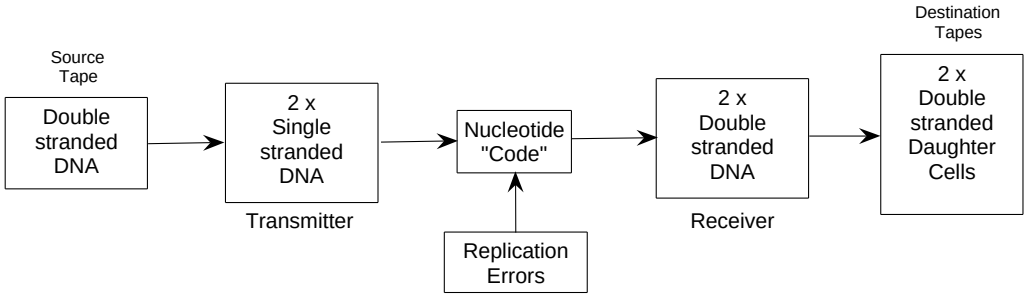

Figure 2.2: **Information-theoretic representation of DNA replication**
The replication errors in the diagram are primarily mismatched base-pairs that escape the proof-reading mechanisms of polymerases. The "nucleotide code" refers to the fact that proteins are stored as a complementary sequence of DNA. Also, this diagram is a simplification of actual replication. For instance, at no point in replication is the entire genome is in the form of a single strand. Rather, the small segment of DNA currently being replicated is in single-strand form.

It is notable that the flow of genetic information always proceeds from the genotype into another genotype (in replication) or into the phenotype (in synthesis), but never from the phenotype into the genotype or from the phenotype into the phenotype. This is a point of profound importance in evolution, and is sometimes termed "The Central Dogma of Molecular Biology". The original version of evolutionary biology, due to Lamarck, proposed a mechanism of "acquired modification" in which advantageous changes in the phenotype (like the arm of the blacksmith) would be inherited. This notion runs in contradiction to the Dogma, since it implies that information in the protein phenotype would flow into the genotype [91]. In modern understanding, such a flow of information never occurs directly but it does occur indirectly in populations. When natural selection responds to the phenotype, it channels information into the genotype. Because of this indirect flow of information, the genome of the organism is said to contain information about the environment.

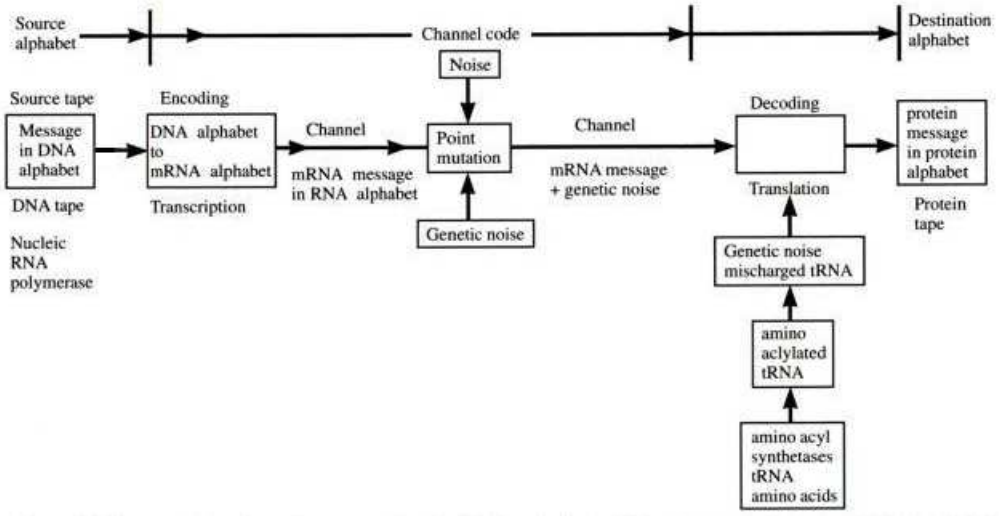

Figure 2.3: **Information-theoretic representation of protein synthesis**
Image source: H.P. Yockey[91]

We have seen how information about a physical system is often represented or stored in another system through a code - a mapping from a set of states of the first to a set of states of the second. In general, information systems use codes in at least two different tasks. First, codes make the message suitable for storage or transmission - a process known as "data compression" and "noiseless coding". For instance, the ASCII code is used to store 256 common characters as bit strings that can be processed in a binary computer system. Second, codes are used to make the information resilient under the impact of errors throughout the information system ("error-tolerant coding" or just "error coding"). Both types of codes are used in biological systems: Although the genome stores instructions for making proteins, proteins are not stored directly in the genome. Rather, the genome consists of nucleotides and the information contained in them becomes proteins after several decoding steps - exemplifying the first type of coding. The genetic code is a code in both senses: It allows protein-making instructions to be stored in DNA and it increases the resilience of those instructions to errors.

In this chapter we will point out the many connections between the engineering problem in information theory and the biological problem. We will discuss:

- Measuring and Storing Information
- Channels and Sources
- Constructive Error Coding

2.1 Measuring and Storing Information

In this section we show how information can be quantified and compressed. In particular, we spend considerable time discussing "entropy" and its properties.

2.1.1 Entropy - a Measure of Information

One of the main achievements of information theory is the quantification of information because it made it possible to study and compare different carriers of information, regardless of whether they are electronic or biological devices. In fact, there are different measures of information with some shared properties [48, p.143]. In the context of codes, one particular measure - the entropy - is most common. Entropy is especially interesting because it could be derived axiomatically. The existence of this axiomatic approach suggests that entropy is not an arbitrary construction and could be relevant to the study of any information-processing system, including those found in biology.

Information theory[76] proposes that the information contained in a physical system like a DNA sequence is dependent on our ignorance of its state. The less we know about the state, the higher the information the system contains. In other words, the lower is our ability to predict the state of the system, the more information we obtain by being told what the state is. Thus, information is a function of the probabilities we assign to the different states. Therefore, given a system with finitely many states labeled $i = 1, 2, \ldots, M$, we define a random variable, X, with probability distribution $p_i, i = 1..M$. The average amount of information we obtain from being told the actual state of the system is known as the entropy, $H(X)$, of system X.

Definition 2.1 *The* entropy *of a system X that is found in states $i = 1..M$ with probabilities $\{p_i\}$ is:*

$$H(X) = -\sum_{i=1}^{M} p_i \log p_i$$

It is conventional in information theory to use base 2 logarithms, and call the units of entropy "bits", although entropy is actually dimensionless.

As was mentioned before, the definition of $H(X)$ can be obtained axiomatically (up to a multiplicative constant) using four axioms. First, suppose all the states of the system are equally probable (with probability $p_i = 1/M, \forall i$). Thus, $H = f(M)$ where f is a function of just the number of states, M. In such a case, we expect and postulate that f is a monotonically increasing function of M.

Axiom 2.1 *If systems X and Y with M and M' states, respectively, satisfy $M' > M$ states, then*

$$f(M') > f(M)$$

Second, suppose there are two *independent* systems X and Y with M and L states, respectively. We expect and postulate that the combined system would contain exactly the sum of the information carried by the two individually. Thus, a DNA sequence made of two sequences that were sampled from a uniform probability distribution has as much information as the sum of the information contained in the two sequences:

Axiom 2.2 *Suppose there are two* independent *systems X and Y with M and L states, respectively, such that all events are equally probable (in both X and Y). Then, the joint entropy satisfies*

$$f(ML) = f(M) + f(L)$$

Third, we return to the general case of entropy H defined on states with possibly different probabilities, $p_i, \forall i$. Suppose that information about X is revealed to us in two stages (a) and (b). In stage (a), we are told whether the system is in one of the states $i = 1..r < M$, or in one of the states $i = r + 1..M$. In stage (b), we are told exactly in which of the states it is. The information we gain this way should be the same as if we were given the state of the system immediately. To express this relation, we note that in this

gedankenexperiment we have three random variables: the original state X with probabilities $\{p_1, p_2, .., p_M\}$, as well as two subsequent random variables Y_1 and Y_2 with (conditional on knowing X) probabilities $\{\frac{p_1}{p_1+..+p_r}, .., \frac{p_r}{p_1+..+p_r}\}$ and $\{\frac{p_{r+1}}{p_{r+1}+..+p_M}, .., \frac{p_M}{p_{r+1}+..+p_M}\}$, respectively. In other words, we require the following equation to hold for entropy:

Axiom 2.3

$$\begin{aligned} H(X) &= H(p_1 + .. + p_r, p_{r+1} + .. + p_M) \\ &+ (p_1 + .. + p_r)H(Y_1) \\ &+ (p_{r+1} + .. + p_M)H(Y_2) \end{aligned}$$

Four, we expect and require that if the system has just two states with probabilities p and $1 - p$, then entropy should depend continuously on p:

Axiom 2.4 *$H(p)$ is a continuous function of p.*

Theorem 2.2 *The only function satisfying the four given axioms is*

$$H(X) = -C \sum_{i=1}^{M} p_i \log p_i \tag{2.1}$$

where C is an arbitrary positive number and the logarithm base is any number > 1.

Proof:
[Sketch] It is not difficult to verify that definition 2.1 satisfies the axioms. What we must do is to show that the only choice in the definition is the choice of C (which is the same as the choice of the basis of the logarithm).

First, we show by induction from the second axiom that for all positive integers M and k we have:

$$f(M^k) = kf(M)$$

Second we show that $f(M) = C \log M$, as follows. For an arbitrary positive integer r, the following expression holds for some k:

$$M^k \leq 2^r \leq M^{k+1}$$

Applying to the previous result and the axiom (1), we obtain:

$$\begin{aligned} kf(M) &\leq rf(2) \leq (k+1)f(M) \text{ or} \\ \frac{k}{r} &\leq \frac{f(2)}{f(M)} \leq \frac{k+1}{r} \end{aligned}$$

Now if we apply log to that same expression we obtain:

$$\begin{aligned} k\log(M) &\leq r\log(2) \leq (k+1)\log(M) \text{ or} \\ \frac{k}{r} &\leq \frac{\log(2)}{\log(M)} \leq \frac{k+1}{r} \end{aligned}$$

Comparing the two we get

$$\left| \frac{\log 2}{\log M} - \frac{f(2)}{f(M)} \right| < 1/r$$

Now we take $r \to \infty$ to obtain that $f(M) = C \log M$.

Third we want to show that in the special case of two events, entropy is $H(X) = -C(p \log p + (1-p)\log(1-p))$. We first consider the case of a rational $p = r/s$. We conceive that the space of events is actually divided into s equally probable states, divided into one group of r and another group of $s - r$ states. We apply the third axiom (the "grouping axiom"), and together with the formula for $f(M)$ above, readily obtain the desired result (algebra not shown). The result extends to all real numbers because $H(p, 1-p)$ is continuous in p, and any real p can be written as a limit of a sequence of rationals.

Finally, we apply induction on M to obtain the formula for entropy of a general probability distribution $p_1, .., p_M$. The case of two events provides a convenient inductive base case, as well as a method for proving the inductive hypothesis. See Ash [3, pp.8-11] for full details. ∎

The Shannon H entropy has several basic properties. First, $H \geq 0$ - this follows from the fact that $-\log(p_i)$ is always positive.
Second, H is a strictly concave function. This becomes important in proving bounds on H.

Proof:
We will prove this result for the special case of binary entropy:

$$H_{bin}(p) = -p \cdot \log(p) - (1-p) \cdot \log(1-p) \tag{2.2}$$

A direct proof requires proving the inequality

$$H_{bin}(px_1 + (1-p)x_2) \geq pH_{bin}(x_1) + (1-p)H_{bin}(x_2)$$

and then showing that equality holds only in trivial cases. However, there exists a convenient theorem, as follows:

Theorem 2.3 *Suppose $f(x)$ has continuous second partial derivatives on some open convex set C in $\Re^n$. If the Hessian $Hf(x)$ of $f(x)$ is positive definite on C, then $f(x)$ is strictly convex on C [65, p.54].*

Returning to H_{bin}, clearly iff $-f(x)$ is strictly convex, then $f(x)$ is strictly concave. Taking the interval $C = (0, 1)$ the Hessian (just the second derivative) of $-H_{bin}$ is positive and continuous: $\frac{d(-H_{bin})^2}{dp^2} = \frac{1}{\ln 2}\frac{1}{p(1-p)} > 0$ in the interval C. The proof for entropies other than the binary entropy can be done similarly. ∎

Third, the uniform case is an upper bound on entropy, i.e.

Theorem 2.4 $H \leq f(M) = \log(M)$ *with equality achieved when the distribution of states is uniform.*

Intuitively, the more random are the outcomes, the more we learn from knowing them, on average.

Proof:

It is helpful to introduce the construction of **Relative Entropy**. If $p(x)$ and $q(x)$ are probability distributions on the same random variable, then we define:

$$H(p(x) \parallel q(x)) = -\sum_x p(x) \cdot \log(\frac{q(x)}{p(x)}) \tag{2.3}$$

It is easy to show that relative entropy is non-negative. This useful property follows from $-\log_a x \geq -(x-1)\ln a$ (with equality iff $x = 1$), implying that

$$\begin{aligned} H(p(x) \parallel q(x)) &\geq -\ln a \sum_x p(x) \cdot (\frac{q(x)}{p(x)} - 1) \\ &= -\ln a \sum_x q(x) + \ln a \sum_x p(x) = 0 \end{aligned}$$

Equality occurs iff $\log \frac{q(x)}{p(x)} = 0$ i.e. when $p(x) = q(x)$ for all x.

Returning to the proof, we introduce the relative entropy involving X and a uniformly-distributed random variable with M states, thus

$$\begin{aligned} H(p(x) \parallel \frac{1}{M}) &= -\sum_x p(x) \log(\frac{\frac{1}{M}}{p(x)}) \\ &= -\sum_x p(x) \log(\frac{1}{M}) + \sum_x p(x) \log(p(x)) \\ &= \log(M) - H(X) \geq 0 \end{aligned}$$

Thus, $H(X) \leq \log(M)$. ∎

Entropy can also be associated with multi-part systems:

Definition 2.5 (Joint Entropy) *Let X and Y be two random variables with each representing a subset of a physical system. Then the joint entropy of the combined system, is*

$$H(X, Y) = -\sum_{x,y} p(x, y) \log p(x, y) \tag{2.4}$$

By inspection, $H(X, Y) = H(Y, X)$.

Joint entropy has the important property of **Subadditivity**, namely:

Theorem 2.6 (Subadditivity) *If the entropies of subsystems X and Y are $H(X)$ and $H(Y)$, respectively, then*

$$H(X, Y) \leq H(X) + H(Y) \tag{2.5}$$

with equality only when X and Y are independent.

Thus, the information contained in the joint system is at most equal to the information contained in its parts.

Proof:

For the proof, we introduce the relative entropy of $p(x, y)$ and $p(x)p(y)$ (their distribution

is multivariate, but the proof of non-negativity of relative entropy can be easily extended for the multivariate case). Hence:

$$\begin{aligned} H(p(x,y) \parallel p(x)p(y)) =& \sum_{x,y} p(x,y) \cdot \log(\frac{p(x,y)}{p(x)p(y)}) \\ =& - H(X,Y) - \sum_{x,y} p(x,y) \cdot \log(p(x)p(y)) \\ =& - H(X,Y) - \sum_{x,y} p(x,y) \cdot \log(p(x)) - \sum_{x,y} p(x,y) \cdot \log(p(y)) \\ =& - H(X,Y) + H(X) + H(Y) \\ \geq& 0 \end{aligned}$$

Thus,

$$H(X) + H(Y) \geq H(X,Y)$$

Lastly, when the subsystems are independent, i.e. $p(x,y) = p(x)p(y)$

$$\begin{aligned} H(X,Y) &= -\sum_{x,y} p(x,y) \log p(x,y) \\ &= -\sum_{x,y} p(x,y)(\log p(x) + \log p(y)) \\ &= -\sum_{x,y} p(x)p(y) \log p(x) - \sum_{x,y} p(x)p(y) \log p(y) \\ &= -\sum_{x} p(x) \log p(x) - \sum_{y} p(y) \log p(y) \\ &= H(X) + H(Y) \end{aligned}$$

∎

An important criterion in engineering information systems is whether the information they store or transmit is the maximum possible. If we write a probability distribution for the states of the information system, then its information is maximized when the distribution is close to uniform. Thus, "highly concentrated" information has the appearance

of random noise. In many practical cases, the information content is not maximized, at the cost of increased storage space and time. For example, in written English the probability distribution of letters is skewed towards e.g. E and away from e.g. Z. Likewise, the probability of words is strongly skewed towards some sequences that are characteristic of the language (e.g. liberté is clearly not English). In biology too, sequences of nucleotides are not random and for instance, RNA of thermophiles contains G and C more frequently than U and A. The goal of maximizing information is often heavily constrained in natural systems. In the case of human language, making the distribution uniform (by changing the words) would so increase the number of consonants in the language, that it will likely become unutterable. In the biological case, a purely random distribution of words (i.e. codons, and hence amino acids) would be maladaptive to the goal of making enzymes. Rather, there is one group of amino acids - the frequent ones like Leucine[31] - which is involved with ensuring that proteins fold correctly, while another group - the rarer one like Cysteine - appears in special usages. We will see a bit later in this chapter how, in many cases, information can be compressed without loss to maximize entropy. In biological systems such a compression/decompression process is not feasible because it is algorithmically complex.

2.1.2 Conditional Entropy and Mutual Information

Certain types of entropy, which we will introduce next, are particularly important for the project of studying the genetic code because they may be used to describe the effect of noise on genetic information.

Conditional entropy reflects the fact that states of one system can suggest or even predict the states of another system. The most prominent example of this is in communication channels. When there are no errors, the input of the channel contains all the information of the output. When errors are present, there is some correlation between the two, and consequently, the overlap between input and output is partial.

Definition 2.7 (Conditional Entropy) *The entropy of Y conditional on knowing X is*

$$H(Y|X) = H(X,Y) - H(X) \tag{2.6}$$

where $H(X,Y)$ is the entropy of the combined system, and $H(Y|X)$ is the information found in Y and which is not provided by knowing X.

It can be seen that

$$\begin{aligned} H(Y|X) &= -\sum_{x,y} p(x,y)\log p(x,y) + \sum_{x} p(x)\log p(x) \\ &= -\sum_{x,y} p(x,y)\log \frac{p(x,y)}{p(x)} \\ &= -\sum_{x,y} p(x,y)\log p(y|x) \end{aligned}$$

Thus, if we have two systems, X and Y, then knowledge about X, say $X = x_i$ would reduce the entropy of Y, giving $H(Y|X = x_i) = -\sum_y p(y|x_i)\log p(y|x_i)$. Hence, $H(Y|X) = \sum_x H(Y|X = x_i)p(x_i)$ gives the average new information found in Y.

An important type of entropy related to conditional entropy is "mutual" information. It measures how much the value of one random variable tells us about another. When the first variable is the input and the second is the output of a communication channel, mutual information gives a measure of how much information is actually being sent from X to Y - the higher the mutual entropy, the more reliable is the channel, which is the opposite from conditional entropy.

Definition 2.8 (Mutual Information)

$$H(Y:X) = H(X) + H(Y) - H(X,Y) \tag{2.7}$$

It is also sometimes written $I(Y|X)$ or $I(Y:X)$. It can be seen from the definition of conditional entropy that

$$\begin{aligned} H(Y:X) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= -\sum_{x,y} p(x,y)\log \frac{p(x)}{p(x|y)} \end{aligned}$$

A graphical representation of the relationships between the various entropies is found in Fig. 2.4. They also can be written compactly as expectation values, $E[\cdot]$ of certain random variables, as follows:

$$\begin{aligned} H(X) &= E[-\log p(X)] && (2.8)\\ H(X,Y) &= E[-\log p(X,Y)] && (2.9)\\ H(Y|X) &= E[-\log p(Y|X)] && (2.10)\\ H(Y:X) &= E[-\log \frac{p(X)}{p(X|Y)}] && (2.11)\\ &= E[-\log \frac{p(Y)}{p(Y|X)}] && (2.12) \end{aligned}$$

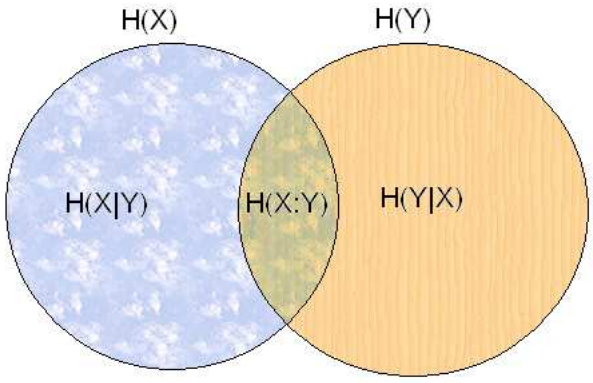

Figure 2.4: **Entropies in a Venn Diagram**
The information quantified by the different entropies can be represented in a Venn diagram. For example, we see that $H(X) = H(X|Y) + H(X:Y)$.

Theorem 2.9 *Mutual information is symmetrical, non-negative and equals zero if and only if X and Y are independent.*

Proof:
Recall that $H(Y:X) = -\sum_{x,y} p(x,y) \log \frac{p(x)}{p(x|y)}$. We introduce relative entropy

$$H(Y:X) = -\sum_{x,y} p(x,y) \log \frac{p(x)p(y)}{p(x,y)} \tag{2.13}$$

$$= H(p(x,y) \parallel p(x)p(y)) \tag{2.14}$$

$$= H(p(x,y) \parallel p(y)p(x)) \tag{2.15}$$

$$= H(X:Y) \tag{2.16}$$

Yet, $H(p(x,y) \parallel p(x)p(y)) \geq 0$ with equality iff $p(x,y) = p(x)p(y)$ for all x, y, meaning that X and Y are independent. ∎
Symmetry is somewhat surprising, because it says that the information contained in X about Y is the same as the information contained in Y about X. Indeed, unlike mutual information, conditional entropy is not symmetrical.[1]

Mutual information can be used to prove that $H(Y|X) \geq 0$ with equality iff $Y = f(X)$ for a deterministic function f.
Proof:
Working from the definition,

$$H(Y|X) = -\sum_{x,y} p(x,y) \cdot \log p(y|x)$$

Since $\log p(y|x) \geq 0$, we have that $H(Y|X) \geq 0$. As to the condition for equality, it is clear that if $Y = f(X)$ then $p(y|x) \in \{0,1\}$ giving the result. Conversely, if $H(Y|X) = 0$, then we must have that $p(y|x) \in \{0,1\}$ implying that for every outcome $X = x$, there is just one outcome $Y = y^*$ with $p(y^*|x) = 1$ and the other y have $p(y|x) = 0$. Thus, Y is known deterministically from X for all outcomes x. ∎

[1]For example, if X and Y are independent and uniformly distributed, and have M and $2M$ states, respectively, we have that $I(X:Y) = 0$, and so $H(X|Y) = H(X) = \log(M)$ but $H(Y|X) = H(Y) = \log(2M)$

2.1.3 Noiseless Coding

One of the most important applications of entropy is "noiseless coding". The basic problem of "noiseless coding" is to represent information in the most economical way, while at the same time discarding nothing. The process is known as "noiseless coding" because our methods will not consider the effect of errors. In truth, few physical systems are truly noiseless. Yet because many stages in the processing of information are relatively error-free (e.g. inside computer memory), it is a reasonable approximation.

Let us first consider the problem of representing information with the motivating example of the genetic code. One function of the genetic code is to represent an amino acid as a sequence of three nucleotides. Any genetic code, whether adapted to errors or not, would have this feature. The general features of such a code are as follows. The original system being encoded, X, is found in one of the M states which we may label $x_1, x_2, .., x_M$ with probabilities $p_1, p_2, .., p_M$. These M states could be e.g. the 20 amino acids, or e.g. the numbers from 1 to 10. The code represents these states as "codewords" $C = w_1, w_2, .., w_M$ built from its alphabet of D characters $\mathcal{A} = \alpha_1, \alpha_2, .., \alpha_D$. The alphabet may consist of e.g. nucleotides or e.g. from ones and zeros. In some codes, known as "block codes", all the codewords have the same length, but that need not be the case in general. The alphabet is often determined by characteristics beyond our control, like the design of an electronic communication system. However, we often do have control over the codewords.

We are interested in a code with the following property:

Definition 2.10 (Unique Decipherability) *A code $C = w_1, w_2, .., w_M$ is called "uniquely decipherable" if every finite sequence $(\alpha_{n(1)}\alpha_{n(2)}\alpha_{n(3)}\ldots)$ of the code characters $\mathcal{A} = \alpha_1, \alpha_2, .., \alpha_D$ corresponds to at most one message (cf. Fig. 2.5).*

One solution to the unique decipherability problem is to use a special class of codes:

Definition 2.11 (Instantaneous Code) *A code is called "instantaneous" if no codeword is a prefix of another. Thus, if A and B are any two codewords, A cannot be written as $sB = A$ where s is a string from the D characters.*

Fig. 2.6 shows one example of such a code. An instantaneous code is always uniquely decipherable because when we start reading from the first character, we are able to decode

State	Codeword
x_1	0
x_2	010
x_3	01
x_4	10

Figure 2.5: **A code that does not decipher uniquely**
The message 010 could be either x_1x_4 or x_2.Example taken from Ash [3, p.28].

State	Codeword
x_1	0
x_2	100
x_3	101
x_4	11

Figure 2.6: **An instantaneous code for M=4, $D = 2$**
Example taken from Ash [3, p.28].

as soon as we receive a complete codeword. The genetic code is clearly an instantaneous code because it is a block code: in a block code it is impossible to form one codeword from another by adding a non-trivial string. However, not all uniquely decipherable codes are instantaneous. For example, the code $x_1 = 0, x_2 = 01$ is not instantaneous, but it is uniquely decipherable because all occurrences of x_2 are distinguished by the character 1. There exists an algorithm for deciding in at most M steps whether a code is uniquely decipherable (see Ash [3]).

Out of the multitudes of codes that satisfy the requirement of unique decipherability, some are also able to compact the information contained in a given message. Such codes would ensure that communicating a message (e.g. a random sequence of the states of X) would require the least amount of space. Mathematically, if $n_1, n_2, .., n_M$ are the lengths of the M codewords, then we wish to minimize $\hat{n} = \sum_{i=1}^{M} p_i n_i$. The solution to this problem calls for using short codewords for states with high p_i, and longer codewords for

the rest. The genetic code does not compact information in this way because all amino acids are represented with codewords of the same length (three), regardless of their relative abundance. Nevertheless, it is worthwhile to discuss how to minimize $\hat{n}$, since it gives a new understanding of entropy, and may also help solve other code optimization problems that are biologically relevant.

We restrict ourself now to instantaneous codes, but we shall see shortly that this is done without loss of generality. Clearly, to minimize space it is convenient to use keywords that are as short as one can decipher uniquely. This cannot be done without bound, because, for example, it is impossible to encode more than 5 states x_i with two-bit binary strings. This is a special case of the following result:

Theorem 2.12 (Kraft inequality) *An instantaneous code with codeword lengths* $n_1, n_2, .., n_M$ *exists if and only if*

$$\sum_{i=1}^{M} D^{-n_i} \leq 1 \tag{2.17}$$

where D *is the size of the alphabet.*

Proof:
Suppose without loss of generality that the codewords are ordered by length, with $n_1 \leq n_2 \leq .. \leq n_M$. All possible codewords of a code can be represented as a collection of combinatorial trees of size (height) n_M each. The vertices of the tree are possible codewords and there are D edges branching out from every non-leaf vertex (Fig. 2.7). In an instantaneous code, a codeword of length n_k excludes $D^{n_M - n_k}$ vertices from being codewords - the number of words in which this codeword would be a prefix. Since the total number of vertices is D^{n_M}, the number of excluded vertices must be less than that, i.e.,

$$\sum_{i=1}^{M} D^{n_M - n_i} \leq D^{n_M}$$

The result follows when we divide by D^{n_M}. Conversely, suppose $n_1, n_2, .., n_M$ are the lengths of codewords in a code satisfying the inequality, and we wish to prove that they could be the codewords of an instantaneous code. We proceed by selecting a codeword of length n_1 which excludes $D^{n_1 - n_k}$ vertices from being codewords. Next, we select a

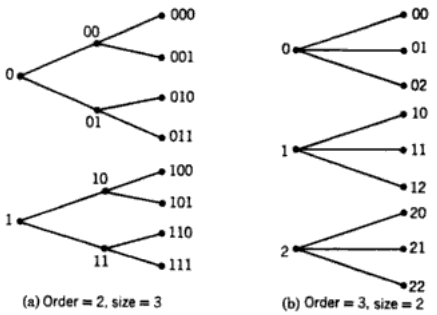

Figure 2.7: **Tree representation of several codes**
All of the vertices represent potential codewords. In an instantaneous code, the choice of codewords are restricted because no keyword is the prefix of another. If, for example, 00 is a keyword, then the strings 000 and 001 (that branch from 00) are excluded. Image source: Ash [3, p.34]

codeword of length n_2 *etc.* Because of the inequality, there is sufficient number of vertices to form an instantaneous code. ∎

The Kraft inequality applies to a more general class of codes:

Theorem 2.13 *If a uniquely decipherable code has word lengths* $n_1, n_2, .., n_M$*, then*

$$\sum_{i=1}^{M} D^{-n_i} \leq 1 \tag{2.18}$$

where D *is the size of the alphabet.*

Proof:
[Sketch] The basic idea of the proof is to separate all the possible messages into classes, where all the messages in class (n, k) encode exactly n symbols of X using exactly k characters of the alphabet. Since the code is uniquely decipherable, the number of messages in the class is at most D^k. The theorem follows when we count the number of messages in all the classes (see Ash for a proof [3, p.36]). ∎
Consequently, if a uniquely decipherable code $C = \{n_i\}$ can be constructed, an instantaneous code of the same codeword length can be constructed as well.

The main theorem of this section establishes a link between the entropy of the messages to be coded, and the design of the codes:

Theorem 2.14 (Noiseless Coding Theorem) *If $\hat{n} = \sum_{i=1}^{M} p_i n_i$ is the average codeword length of a uniquely decipherable code for the random variable X, then*

$$\hat{n} \geq H(X) \tag{2.19}$$

with equality if and only if $p_i = D^{-n_i}$ for $i = 1, 2, .., M$.

In the inequality we computed $H(X)$ with logarithm in basis D. If entropy is calculated for log to basis $a > 1$, the lower bound is instead $\frac{H(X)}{\log_a D}$.

Proof:

The condition

$$\hat{n} = \sum_{i=1}^{M} p_i n_i \geq H(X) = -\sum_{i=1}^{M} p_i \log_D p_i$$

can be rewritten as

$$\sum_{i=1}^{M} p_i \log_D \frac{p_i}{D^{-n_i}} \geq 0 \tag{2.20}$$

This inequality is similar to the non-negativity of relative entropy, $H(p_i \parallel q_i)$, in the case where we choose the probability distribution $q_i = D^{-n_i} / \sum_{i=1}^{M} D^{-n_i}$.

We proceed by making the above choice for q_i and write the non-negativity condition:

$$\begin{aligned} H(p(x) \parallel q(x)) &= \sum_x p(x) \cdot \log_D \left(\frac{p(x)}{q(x)} \right) \\ &= \sum_{i=1}^{M} p_i \log_D \frac{p_i}{D^{-n_i} / \sum_{i=1}^{M} D^{-n_i}} \\ &\geq 0 \end{aligned}$$

with equality if and only if

$$p_i = q_i = D^{-n_i} / \sum_{i=1}^{M} D^{-n_i} \tag{2.21}$$

Expanding the relative entropy, we get

$$\begin{aligned}
\sum_{i=1}^{M} p_i \log_D p_i - \sum_{i=1}^{M} p_i \log_D D^{-n_i} + \sum_{i=1}^{M} p_i \log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) &\geq 0 \\
-H(X) - \sum_{i=1}^{M} p_i(-n_i) + \sum_{i=1}^{M} p_i \log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) &\geq 0 \\
-H(X) + \hat{n} + \log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) &\geq 0
\end{aligned}$$

or

$$H(X) \leq \hat{n} + \log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) \tag{2.22}$$

By unique decipherability, the code must satisfy $\sum_{i=1}^{M} D^{-n_i} \leq 1$ i.e. $\log_D \sum_{i=1}^{M} D^{-n_i} \leq 0$. Hence,

$$H(X) \leq \hat{n}$$

with equality if and only if $p_i = \frac{D^{-n_i}}{\sum_{i=1}^{M} D^{-n_i}}$.

Now, it can be seen that the distribution $p_i = D^{-n_i}$ $(= \frac{D^{-n_i}}{\sum_{i=1}^{M} D^{-n_i}})$ yields the equality:

$$H(X) = -\sum_{i=1}^{M} p_i \log_D p_i = -\sum_{i=1}^{M} p_i(-n_i) \log_D D = \hat{n}$$

All that remains to be proven is that $H(X) = \hat{n}$ implies that $p_i = D^{-n_i}$. Applying Eqn. 2.22 to our case we get

$$0 \leq \log_D \left(\sum_{i=1}^{M} D^{-n_i} \right)$$

But by unique decipherability,

$$\log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) \leq 0$$

leaving the only possibility

$$\log_D \left(\sum_{i=1}^{M} D^{-n_i} \right) = 0$$

But by Eqn. 2.21 this means that $p_i = D^{-n_i}$. ∎

The noiseless coding theorem gives a new interpretation of entropy as a measure of compressibility. Suppose the original states $x_1, x_2, .., x_M$ are strings in the same alphabet as the codewords $w_1, w_2, .., w_M$ and the alphabet having D characters. In such a case, the theorem opens the possibility of finding a re-encoding of the M states such that the average word length is minimized and equals the entropy. No better compression algorithm exists for this data because such an algorithm would have $\hat{n} < H(X)$, in contradiction of the theorem.

Definition 2.15 *We shall call "absolutely optimal" any code that achieves the lower bound on average codeword length (e.g. Fig. 2.8).*

State	Probability	Codeword
x_1	1/2	0
x_2	1/4	10
x_3	1/8	110
x_4	1/8	111

Figure 2.8: **An absolutely optimal code**
Here $H = -(.5)\log(.5) - (.25)\log(.25) - (.125)\log(.125) - (.125)\log(.125) = \frac{7}{4}$.
This is the same as $\hat{n} = .5 + 2/4 + 3/8 + 3/8 = \frac{7}{4}$. Example taken from Ash [3, p.28].

Moreover, if constructed, the optimal code would be an instantaneous code, as can be seen from the following argument. Suppose we design the best instantaneous code C, but then find an uniquely-decipherable code C' with codeword lengths $n'_1, n'_2, .., n'_M$ that has a smaller $\hat{n}$ than C. By theorem 2.18, $\sum_{i=1}^{M} D^{-n'_i} \leq 1$. Yet, by theorem 2.17, an instantaneous code C'' must exist with those codeword lengths, in violation of the assumption.

Although the theorem opens the door for the construction of an absolutely optimal code, it does not really show that such a code exists. Yet, we do have the following result:

Theorem 2.16 (Optimal Instantaneous Code) *Given a random variable X with entropy $H(X)$, there exists an instantaneous code for X with alphabet size D whose average code-word length $\hat{n}$ satisfies*

$$H(X) \leq \hat{n} < H(X) + 1 \tag{2.23}$$

Proof:
We have seen in Eqn. 2.22 that the best choice for codeword lengths satisfies $p_i = D^{-n_i}$, i.e. $n_i = -\log_D p_i$. However, this number may be a non-integer. We *can* select the next integer, giving us

$$-\log_D p_i \leq n_i < -\log_D p_i + 1 \tag{2.24}$$

It is necessary to show that such a choice of codewords can be realized in an instantaneous code. To that goal, we must simply show that $\sum_{i=1}^{M} D^{-n_i} \leq 1$. Fortunately, $\sum_{i=1}^{M} D^{-n_i} \leq \sum_{i=1}^{M} p_i = 1$. The average codeword length is found by multiplying Eqn. 2.24 with p_i and summing over all i:

$$\begin{aligned} -\sum_{i=1}^{M} p_i \log_D p_i &\leq \sum_{i=1}^{M} p_i n_i < -\sum_{i=1}^{M} p_i \log_D p_i + \sum_{i=1}^{M} p_i 1 \\ H(X) &\leq \hat{n} < H(X) + 1 \end{aligned}$$

∎

In fact, it is possible to find an instantaneous code that is arbitrarily close to the theoretic bound by using large code blocks. The idea is to encode not an individual value of X but a sequence $Y = X_1 X_2 \ldots X_s$ of s such values. By theorem 2.22,

$$H(Y) \leq \hat{n_s} < H(Y) + 1$$

But X_i values are independently distributed, and hence $H(Y) = H(X_1) + .. + H(X_s) = sH(X)$. Thus,

$$\begin{aligned} sH(X) &\leq \hat{n_s} < sH(X) + 1 \\ H(X) &\leq \frac{\hat{n_s}}{s} < H(X) + \frac{1}{s} \end{aligned}$$

For large s, $\hat{n_s}/s$ (i.e. the average number of characters per value of X) approaches the entropic bound asymptotically. A systematic way of constructing such optimal instantaneous codes was found by Huffman [3, pp.42-3].

We note now that in an absolutely optimally encoded message, the D characters of the alphabet must be uniformly distributed, for the following reason. If, by contradiction, their distribution was skewed, then consider the distribution of pairs of characters X_iX_j. Pairs encoding the more common characters would appear more frequently than the rest, and consequently it would be possible to reduce the length of the message by re-encoding the character pairs. Thus, there would exist a code that would be "more optimal", i.e. a code that further decreased the average code length. Yet, this would decrease the $\hat{n}$ of the original message - which is precluded by the Noiseless Coding Theorem. Hence, we must conclude that probability of the D letters (but not the codewords) in in the code $C = w_1, w_2, .., w_M$ must be uniform. This implies that compressed data would look like random noise (when considering the distribution of the letters).

2.2 Channels and Sources

In this section we introduce a mathematical description of communication channels and information sources, whose understanding would provide the necessary background for studying codes like the genetic code. Our study of channels focuses on the simplest and most important case: the memoryless channel. We will also make a distinction between noiseless and noisy channels. Coding for the noiseless type is primarily interested in compact representation of information (as discussed earlier), while in the latter type we would also be interested in error-tolerant coding. Regarding the noisy channels, we will prove a fundamental theorem about the fundamental bound on reliable transmission. We will also consider the problem of information degradation during transmission through cascades of noisy channels - a problem with direct meaning in biological inheritance.

Definition 2.17 (Discrete Source) *A* discrete source *is a device that produces messages i.e. sequences of random variables $X_0, X_1, X_2, ..$ such that each X_i takes values in a finite set Γ, called the alphabet of the source.*

A source is sometimes classified as either memoryless, ergodic, stationary, or stochastic where each class is more general than the preceding one. For our purposes, it would be sufficient to consider only the memoryless source, i.e. a source in which each X_i is a random variable with distribution $p(X)$ - a distribution that has no dependence on either time or the preceding outcomes. Naturally, the entropy of such a source is just $nH(X)$, where n is the number of symbols in the message.

Messages from information sources become the inputs of information channels:

Definition 2.18 (Discrete Channel) *A* discrete channel *is a device that takes an input message in the form of a sequence of symbols $\alpha_1, \alpha_2, .., \alpha_n$ ($\alpha_i \in \Gamma$) and returns an output message in the form of a sequence of symbols $\beta_1, \beta_2, .., \beta_n$ ($\beta_i \in \Gamma'$). The length of the input and output could be arbitrarily long, i.e. $n = 1, 2, ...$ The output of the channel would in general depend on the initial state $s \in S$ of the channel.*

The relation between the input and output is described probabilistically via the distribution function $p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n; s)$. As a probability function, p_n has the following properties:

1. $p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n; s) \geq 0$, *for all possible arguments*
2. $\sum_{\beta_1,\beta_2,..,\beta_n} p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n; s) = 1$, *for all possible inputs and s values.*

The differences between the different types of channels are due to differences in the probability distributions $p_n(\beta_1, \ldots|\alpha_1, \ldots; s)$.

2.2.1 The Discrete Memoryless Channel

One of the simplest types of channels are channels whose output depends solely on the symbol currently entering the input, as follows:

Definition 2.19 (Memoryless Channel) *A discrete channel is* memoryless *if*

1. *The functions $\{p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n; s)\}$ do not depend on s and hence may be written $p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n)$*
2. $p_n(\beta_1, \beta_2, .., \beta_n|\alpha_1, \alpha_2, .., \alpha_n) = p_1(\beta_1|\alpha_1)p_1(\beta_2|\alpha_2) \cdots p_1(\beta_n|\alpha_n)$, *for all possible inputs, outputs and message lengths n.*

Such a channel is fully specified once we provide the *channel matrix*, denoted $a_{\alpha\beta}$ or p where $a_{\alpha\beta} = p(\beta|\alpha) = p_1(\beta|\alpha)$.

Suppose the input symbols are $x_1, .., x_M$ while the output symbols are $y_1, .., y_L$. If we have a probability distribution for the inputs, $p(x_i)$, then we can find the probability distribution of the output and the joint probability distribution:

$$\begin{aligned} P(X = x_i, Y = y_j) &= p(x_i)p(y_j|x_i) \\ P(Y = y_j) &= \sum_{i=1}^{M} p(x_i)p(y_j|x_i) \end{aligned}$$

Hence, we can define the entropy functions of the channel:

Definition 2.20

1. *The input entropy* $H(X)$,
2. *The output entropy* $H(Y)$,
3. *The joint entropy* $H(X, Y)$,
4. *The conditional entropies* $H(X|Y)$ *and* $H(Y|X)$,
5. *The transmission* $H(X : Y)$

Definition 2.21 *We shall call a channel "lossless" if* $H(X|Y) = 0$ *meaning that the output perfectly characterizes the input. Similarly, a channel is called "deterministic" if the input unambiguously predicts the output, i.e.* $H(Y|X) = 0$. *A channel is called noiseless if it is lossless and deterministic. We may also define a "useless" channel by the condition* $H(X : Y) = 0$ *i.e.* $H(X|Y) = H(X)$.

In a lossless channel, we have an injective (one-to-one) function from X to Y and the transmission is just the input entropy. This means that perfect decoding is possible. In contrast, the deterministic channel defines a surjective function from X to Y and sets the

transmission at just the output entropy. Next, the noiseless case defines a bijection and hence the transmission is identical to the input and the output entropies. In the useless channel the input and output are independent random variables. Thus, knowledge of the output tell us nothing of the input and vice versa. Most real channels are noisy because they are either indeterministic or lossy or both.

Definition 2.22 (Channel Capacity) *The channel capacity C is the maximum of the transmission function over all inputs distributions $p(x)$:*

$$C = \max_{p(x)} H(X:Y) = \max_{p(x)} \left(H(Y) - H(Y|X)\right) \tag{2.25}$$

Thus, the capacity is the entropy of the output, $H(Y)$, subtract the noise of the channel, $H(Y|X)$. The second term may be called negative information - the information lost due to the noise. Notice that in a deterministic channel the second term vanishes. An equivalent expression is $C = \max_{p(x)} \left(H(X) - H(X|Y)\right)$. Here the second term, known as "equivocation" describes the entropy lost from the input entropy due to the channel noise.

We shall see that the channel capacity defines an upper bound on the amount of information that could be transmitted reliably through the channel (loosely speaking). As can be seen from the definition, the problem of finding C is the problem of transmitting information such that the output information is greatest, while at the same time incurring the fewest losses. Thus, it is clearly advantageous to decrease the input probability, $p(x)$, of the error-prone symbols. However, there is a tradeoff: If we sent just the least error-prone symbol, we would not be able to transmit any information ($H(X) = 0$). Those two conflicting goals involved in finding C is what makes determining C a hard computational problem. Yet, we have a closed-form expression for the capacity of a particularly simple class of noisy memoryless channels:

Definition 2.23 *The symmetric channel is a memoryless discrete channel with the following property: If for an input symbol x_i, $p(y_j|x_i) = p_j$, then for any other input symbol x_k, $p(y_j|x_k) = p_{j(k)}$ - a permutation of the set $\{p_j\}$.*

A particularly simple type of a symmetric channel is one where for all x_i and y_j,

$$p(y_j|x_i) = \begin{cases} 1-\beta & i = j \\ \frac{\beta}{L-1} & i \neq j \end{cases}$$

The unique property of any symmetric channel is that $H(Y|X)$ depends only on the channel probabilities and not on the input distribution $p(x)$. This is due to the fact that all the input symbols are equally error prone, see Ash [3, p.52-3]. Thus, the task of finding C is reduced to setting the input $p(x)$ to achieve the maximum $H(Y)$. In fact, if we take a uniform distribution of the inputs, we obtain a uniform distribution of the outputs which maximizes $H(Y)$. Thus, $C_{sym} = \log L - H(Y|X) = \log L + \sum_{j=1}^{L} p_j \log p_j$. A special case is the binary symmetric channel (BSC) shown on Fig. 2.9.

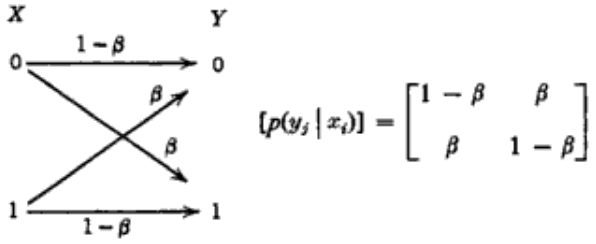

Figure 2.9: **Binary Symmetric Channel**

The channel capacity is $C_{BSC} = \log 2 + \beta \log \beta + (1-\beta)\log(1-\beta)$. It reaches one bit when $\beta = 0, 1$ and vanishes when $\beta = 1/2$. In the case $\beta = 1$, all the bits flip during transmission but the channel is deterministic - all the flips can be perfectly corrected by flipping them back. Image Source: Ash [3, p.53]

It is possible to find a closed-form expression of the channel capacity of a large set of channels, namely, channels having a square $p(y_j|x_i)$ matrix and satisfying a certain criterion (see Ash [3, pp.53-60]). However, the general problem of computing C is a problem in numerical (convex) optimization.

2.2.2 Decoding Schemes

One of the most important problems in communication is to decide which symbol was sent given that a certain symbol was received. More precisely, we define the decoder:

Definition 2.24 (Decoder) *Suppose a noisy channel has an alphabet $x_1, x_2, .., x_M$ an output alphabet $y_1, y_2, .., y_L$ and a channel matrix $p(y_j|x_i)$. A decoder or a "decision*

scheme" is a deterministic channel that takes a value of Y and returns a value of X.

Suppose furthermore that we are given the input distribution $p(x)$. A decoder that minimizes the overall probability of error is called the "ideal observer". More concretely, suppose that the decoder takes y_j and outputs x_j^*. Let $P(c|y_j)$ be the probability of correct decision given that y_j was received. In such a case, the overall probability of correct decision is

$$P(c) = \sum_{j=1}^{L} p(y_j)P(c|y_j) = \sum_{j=1}^{L} p(y_j)P(X = x_j^*|y_j) \tag{2.26}$$

The probability $p(y_j)$ is the same for all decision schemes, and the decoder is free to choose x_j^*. If the decoder maximizes for each y_j the probability, $p(x|y_j)$, that x was sent given that y_j was received then the decoder maximizes the probability $P(c|y_j)$ for each y_j and hence it maximizes $p(c)$. An important special case occurs when all inputs are equally likely ($p(x_i) = 1/M$) and we get

$$p(x_i|y_j) = \frac{p(x_i)p(y|x_i)}{p(y)} = \frac{1}{Mp(y)}p(y_j|x_i) \tag{2.27}$$

Thus, by finding the symbol x_j^* most likely to produce y_j i.e, by maximizing $p(y_j|x_i)$ we automatically maximize $p(x_i|y_j)$ - the probability that x_i was sent. A decoding scheme that maximizes $p(y_j|x_i)$ is known as maximum likelihood decoding (MLD).

To illustrate the different schemes, consider the following example of a code and a decoding scheme. The code, known as a triplication code, encodes $0, 1$ as $000, 111$, respectively for transmission in a binary symmetric channel with error probability β. Suppose $\beta < 1/2$ and $y_j = 001$ is received. The MLD decoder would always correct $= 001$ to $x_i^* = 000$ because it maximizes the conditional probability $p(y_j|x_i^*)$. However, this decision would minimize the overall probability of error only under some input probabilities. For illustration, suppose 111 is sent with probability p significantly exceeding the probability $1-p$ of 000 being sent. For certain range of β, the probability that 111 was sent and experienced a double error is higher than the probability that 000 was sent and experienced a single error, i.e. $p(x_i^* = 111|001) > p(x_i^* = 000|001)$ and explicitly $p\beta^2(1-\beta) > (1-p)\beta(1-\beta)^2$. In such cases, the ideal observer that maximizes $p(x_i^*|y_j)$ would correct to 111 rather than 000 for the MLD.

Decoding done by the genetic code differs in an important way. Whereas in the triplication code only two of the eight three bit strings are codewords, all the codons are codewords of the genetic code. This means that it is impossible to detect errors (a topic we will consider in the next section). Indeed, the genetic code proceeds to decode a codon for what it is without considering the possibility that C may be the product of an error taking $C' \to C$ during transcription, where C' is another codon. Indeed, although many organisms avoid using some of the codons in their genomes[46], these codons are often readily decoded despite the fact that they re-appear in the genome only due to mutations. Thus, the genetic code acts as an MLD rather than as an "ideal observer". Fortunately, such a decoding scheme tends to produce the x_i^* answer most of the time because the probability of error is far smaller than the difference between probabilities of inputs. Quantitatively, whereas the probabilities of error per nucleotide, even in primordial organisms, are in the range of $10^{-5} \ldots 10^{-3}$ (see Fig. 1.15), the probabilities of the amino acids differ by at most an order of magnitude (see Gilis[31]). Thus, the genetic code decodes in a regime where MLD is indistinguishable from the ideal observer.

2.2.3 The Fundamental Theorem

In this subsection, we shall introduce and prove the so-called "Fundamental Theorem of Information Theory". The essence of the fundamental theorem is that in order to achieve arbitrarily high reliability, it is *not* necessary to reduce the transmission rate to zero, but only to bring it below the channel capacity. However, the proof of the theorem does not provide us with a technique for constructing codes that minimize errors while allowing transmission near the channel capacity. Such codes are the subject of the next section.

Theorem 2.25 (Fundamental Theorem) *Let a discrete memoryless channel have capacity C and let a source X produce information at a rate per unit time $H(X) \leq C$. Then there exists a coding system such that the output of the source can be transmitted over the channel $X \to Y$ with an arbitrarily small frequency of errors (or an arbitrarily small equivocation $H(X|Y)$).*

It may also be shown that if $H(X) > C$, then $H(X|Y)$ cannot be less than $C - H(X)$ (see Shannon [76]). We see that the theorem establishes the channel capacity as a fundamental

but achievable barrier in the reliable transmission of information.

The above formulation of the theorem due to Shannon leaves some ambiguity as to the nature of the codes and the method for determining the probability of errors. Both concepts can be defined to state a more precise version of the theorem [3, p.66]. First, let us define the block codes, which we will use in stating and proving the fundamental theorem:

Definition 2.26 (Block code) *Given a discrete memoryless channel, an (s, n) block code is a set of s input n-sequences (sequences of n letters belonging to the alphabet) together with a corresponding decision scheme (a function that assigns one of the s input sequences $x^{(1)}, .., x^{(s)}$ to each output n-sequence). The decoding scheme can be said to divide the output n-sequences into s "decoding sets" $B^{(1)}, .., B^{(s)}$ where, for all i, all the elements in $B^{(i)}$ are decoded as $x^{(i)}$.*

Second, let us introduce a systematic definition of the reliability of a code. Suppose a sequence $x^{(i)}$ is sent. The probability of error is

$$p(e|x^{(i)}) = \sum_{y \notin B^{(i)}} p(y|x^{(i)}) \tag{2.28}$$

The overall probability of error when $x^{(i)}$ are sent with distribution $\mathcal{D}$ (probabilities $p(x^{(i)})$) is

$$p_{\mathcal{D}}(e) = \sum_{y \notin B^{(i)}} p(x^{(i)}) p(e|x^{(i)}) \tag{2.29}$$

The average probability of errors, i.e., the probability of errors when all the inputs have the same probabilities ($p(x^{(i)}) = 1/s$) is

$$\overline{p(e)} = s^{-1} \sum_{y \notin B^{(i)}} p(e|x^{(i)}) \tag{2.30}$$

Finally, the maximum probability of any error is

$$p_m(e) = \max_{1 \leq i \leq s} p(e|x^{(i)}) \tag{2.31}$$

Definition 2.27 (Error bounds) *Suppose (s,n) code has a decoding scheme that guarantees that $p_m(e) \leq \epsilon$. Thus, the probability of error is $\leq \epsilon$ no matter which codeword was sent. We say that we have a uniform error bound ϵ and that the code is an (s, n, ϵ) code.*

Theorem 2.28 (Fundamental Theorem - Precise) *Given a discrete memoryless channel with capacity $C > 0$ and a transmission rate $R < C$, there exists a sequence of codes $\mathcal{A}_1, \mathcal{A}_2, ..$ such that $\mathcal{A}_n$ is a $([2^{nR}], n, \lambda_n)$ code and $\lambda_n \to 0$ as $n \to \infty$. Here $[2^{nR}]$ means the smallest integer greater than 2^{nR}. Moreover, there exist positive constants K and α, depending on the channel and R such that $\lambda_n \leq Ke^{-\alpha n}$. If, in contrast, $R = C + d$ with $d > 0$, the average probability of error $\overline{p(e)}$ must satisfy*

$$\log s \leq \frac{nC + \log 2}{1 - \overline{p(e)}} \tag{2.32}$$

The converse implies that if our code uses $2^{n(C+d)}$ codewords, then the error rate must equal

$$n(C + d) \leq \frac{nC + \log 2}{1 - \overline{p(e)}}$$

Which in the limit of $n \to \infty$ gives

$$\overline{p(e)} \geq 1 - \frac{C}{C + d}$$

Thus the probability of error cannot be reduced to zero.

The proof of the fundamental theorem is dependent on a remarkable property of entropy, known as asymptotic equipartition (AEP). Asymptotic equipartition is closely related to the fact that for sufficiently large n, n-sequences contain symbols in relative frequencies closely agreeing with the probabilities of those symbols (due to the weak law of large numbers). To define AEP, we need to classify n-sequences, as follows. Let X be a random variable taking on the values from the alphabet $\alpha_1, .., \alpha_M$ with probabilities $p_1, .., p_M$, respectively. Suppose $X_1, .., X_n$ are n independent random variables distributed like X. Let $f_i(X_1, .., X_n)$ be the number of times that the symbol X_i appears in $X_1, .., X_n$. As a random variable, f_i has the binomial distribution with mean np_i and standard deviation $\sigma_i = \sqrt{np_i(1 - p_i)}$.

Definition 2.29 (k-Typical Sequences) *Let $x = (x_1, .., x_n)$ be a particular n-sequence. Given $\epsilon > 0$ and $k > 0$ such that $1/k^2 < \epsilon/M$, we say that x is "typical" if for all symbols α_i,*

$$\left| \frac{f_i(x) - np_i}{\sigma_i} \right| < k \tag{2.33}$$

Thus, in any typical sequence, each symbol α_i occurs approximately with its expected frequency np_i. The strictness of the definition depends on the choice of k and hence ϵ. The next theorem essentially shows that the number of typical n-sequences is approximately 2^{nH} each of probability approximately 2^{-nH}. Thus, nontypical n-sequences are exceedingly rare:

Theorem 2.30 (Asymptotic equipartition property)

1. *The set of nontypical sequence of length n has total probability* $< \epsilon$
2. *There is a positive number A such that for each typical n-sequence x,*

$$2^{-nH-A\sqrt{n}} < P\left((X_1, .., X_n) = x\right) < 2^{-nH+A\sqrt{n}} \tag{2.34}$$

3. *The number of typical sequence of length n is* $2^{nH+\delta_n}$ *where* $\lim_{n\to\infty} \delta_n = 0$

For a proof, see Ash [3, p.15]

We are now ready to sketch the proof of the fundamental theorem. The basic idea of the proof is the fact that the volume occupied by the decoding sets of the codewords falls exponentially with n in comparison with the space of possible input n-sequences, making it possible to decode with arbitrary precision.

Proof:

Suppose the distribution $p(X)$ is the distribution giving the channel capacity for one symbol message X. Based on this distribution, we can calculate the entropies $H(X), H(X|Y), H(Y)$ and others. We now consider the n-sequence inputs where each symbol is sampled from $p(X)$. Since the channel is memoryless, the entropy for n-sequences is just n times the entropy for one symbol.

We now construct a code that allows us to transmit at rate $R < C$. For the transmission of a set of symbols, we sample from the probability function $p(x)$ that maximizes the channel capacity to generate high likelihood words. Out of this set of $2^{nH(X)}$ words, we *randomly* chose codewords $\{x^{(i)}\}$.

Because of AEP, there are $2^{nH(X,Y)}$ input-output pairs of typical n-sequences and $2^{nH(Y)}$ typical outputs. Moreover, on average the number of typical input sequences that are

"likely" to produce a given output sequence is $2^{nH(X|Y)} = 2^{nH(X,Y)-H(Y)} = \frac{2^{nH(X,Y)}}{2^{nH(Y)}}$. We also note that when the channel capacity $C = H(X) - H(X|Y)$ is large compared to the rate of transmission, the number of inputs per output ($2^{nH(X|Y)}$) is small compared to the possible number of codewords $2^{nH(X)}$, which makes reliable decoding quite easy. As the channel capacity falls or equivocation $2^{nH(X|Y)}$ rises, it becomes increasingly hard to decode reliably, for the following reason. Because the quantity $2^{nH(X|Y)}$ can also be written as $\frac{2^{nH(X)}}{2^{nC}}$, in the limit of zero C, the number of codewords $2^{nH(X)}$ is approximately the same as $2^{nH(X|Y)}$ - the number of possible input n-sequences producing a given output. In this limit, reliable transmission becomes increasingly difficult.

To find the probability of error, consider how many errors is the random code likely to make on average. Suppose codeword $x^{(i)}$ is transmitted through the channel and y is received. Moreover, the decoding algorithm is programmed to decode y as $x^{(i)}$. Suppose we call S the set of inputs likely to produce y. The probability of incorrect decoding corresponds to the probability that some other codeword $x^{(j)}$ is in S:

$$\begin{aligned} P(error) &= P(\text{at least one other codeword } x^{(j)} \in S) \\ &\leq P(x^{(1)} \in S \text{ or } x^{(2)} \in S, ..) \\ &= \sum_{j=1, j\neq i}^{2^{nR}} P(x^{(j)} \in S) \\ &= 2^{nR} \frac{2^{nH(X|Y)}}{2^{nH(X)}} \\ &= \frac{2^{nR}}{2^{nC}} \end{aligned}$$

When $R < C$, as $n \to \infty$, $P(error) \to 0$. ∎

The above argument does not quite constitute a proof because it does not carefully account for the errors in the approximations. Neither have we shown that there exists a uniform error bound as opposed to an average error bound. However, all the known proofs of the theorem and its more precise statement are based on versions of the argument above (see Ash [3, pp.67-77] for two different proofs and a proof of the converse).

The above argument is interesting in the sense that it is constructive, but the construction can rarely be implemented. First, there are often practical limitations as to how large can n be. In the case of the genetic code, chapter 1 suggested that blocks of three

nucleotides per amino acid may represent a biochemical optimum that could not be altered in the error-minimizing evolution of the genetic code. Moreover, most random codes lead to computationally hard encoding and decoding schemes. Nevertheless, the problem of constructing codes has been thoroughly studied and will be discussed in the next section.

2.2.4 The Channel Cascade Problem

In most applications, information transits a series of channels on the way from a sender to a receiver. Each channel may have its own alphabet, code and noise (Fig. 2.10). This "cascade" is particularly important in biology, because it provides a model for cell division, with the sender being the ancestral organism, and the receiver(s) are all of its descendants. Other processes also have a cascade structure.

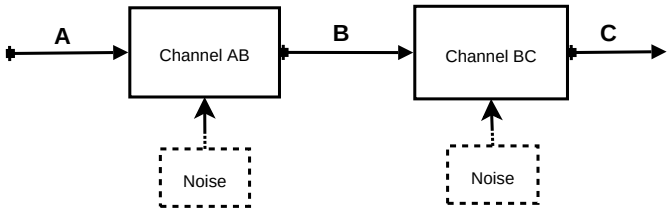

Figure 2.10: **Cascade of two channels**
The communication from A to C goes through two noisy channels, AB and BC, accumulating losses of information, which cannot be reversed.

In the following theorem, we see that information inevitably degrades as it passes through a cascade:

Theorem 2.31 (Data Processing Inequality) *Suppose a random variable A is degraded by noise to give B which in turn experiences noise to give C. Suppose also that $p(C|A,B) = p(C|B)$, i.e. the channels form a Markov chain $A \to B \to C$. Then*

$$H(A) \geq H(A\colon B) \geq H(A\colon C) \tag{2.35}$$

(for proof, see Cover and Thomas [10, p.32]). Genetic information passing through the cascade in Fig. 2.10 indeed forms a Markov chain since there is no direct transfer of information from A to C.

Although cascades are part of standard information theory literature, the interaction between them and natural selection has apparently been ignored. However, as was noted by the biologists Eigen and Schuster, because natural selection is able to eliminate individuals who have lost "the master sequence" - the original message - due to errors, selection is able to correct errors in the population. Nevertheless, its error-correcting capacity is limited:

> "Genetic reproduction is a continuously self-repeating process, and as such differs from a simple transfer of a message through a noisy channel. For each single transfer it requires more than just recovery of the meaning of the message, which, given some redundancy, would always [inevitably] allow a fraction of the symbols to be reproduced incorrectly. It is also necessary to prevent any further accumulation of mistakes in successive reproduction rounds. In other words, a fraction of precisely correct wild-type copies must be able to complete favorably with the total of their error copies. Only in this way can the wild-type be maintained in a stable distribution. Otherwise the information [the number of copies of the master sequence] would slowly seep away until it finally is entirely lost."[19, p.14]. [2]

The problem of accumulation of errors is particularly challenging when multiple noisy channels exist between two selective steps. This is precisely the case in all organisms that have complex life cycles such that reproduction occurs only after many rounds of cell division. How high would the error rate be in such a cascade? Intuitively, we expect that noise would degrade a fixed fraction of the information contained in the input, say $0 < \mu < 1$. Hence after an n-level cascade, $(1-\mu)^n$ of the information would remain - representing a loss exponential in n. More explicitly, we suppose that information is transmitted through a cascade of n channels. For simplicity, we also assume like in theorem 2.31 that the channels form a Markov chain, and that moreover, they are all identical. Thus, any one channel can be described by the transition matrix $p(b_j|a_i)$. In the special case of binary symmetric channels (BSCs), this matrix is characterized by β - the per-symbol probability of error

$$P(A_1|A_0) = \begin{pmatrix} 1-\beta & \beta \\ \beta & 1-\beta \end{pmatrix}$$

[2] More information about this process is found in subsection 1.3.1 and in Appendix A.

The transition matrix for the n-level cascade can be found by diagonalizing this matrix, taking advantage of the fact that the eigenvalues are $\lambda_1 = 1, \lambda_2 = 1 - 2\beta$. Diagonalization allows us to obtain the transition matrix for the cascade connecting A_0 to A_n:

$$P(A_n|A_0) = P(A_1|A_0)^n = \frac{1}{2}\begin{pmatrix} 1+(1-2\beta)^n & 1-(1-2\beta)^n \\ 1-(1-2\beta)^n & 1+(1-2\beta)^n \end{pmatrix}$$

Suppose that initially the two symbols (say $0, 1$) have a uniform distribution. We can then proceed to calculate the mutual information - the information about A_0 at A_n:

$$\begin{aligned} I(A_n : A_0) &= E[-\log \frac{p(Y)}{p(Y|X)}] \\ &= -P(0,0)\log\frac{p(0)}{p(0|0)} - P(0,1)\log\frac{p(1)}{p(1|0)} - P(1,0)\log\frac{p(0)}{p(0|1)} - P(1,1)\log\frac{p(1)}{p(1|1)} \\ &= -2 * .5 * .5\,(1+(1-2\beta)^n)\log\frac{.5}{1+(1-2\beta)^n} \\ &\quad -2 * .5 * .5\,(1-(1-2\beta)^n)\log\frac{.5}{1-(1-2\beta)^n} \\ &= .5\,(1+(1-2\beta)^n)\log(1+(1-2\beta)^n) + .5\,(1-(1-2\beta)^n)\log(1-(1-2\beta)^n) \end{aligned}$$

Now using the Taylor expansion $\log(1+x) = \frac{1}{\ln(2)}\left(x - \frac{1}{2}x^2 + O(x^3)\right)$ we get:

$$\begin{aligned} I(A_n : A_0) &= .5\,(1+(1-2\beta)^n)\left((1-2\beta)^n + O((1-2\beta)^{2n})\right) \\ &\quad +.5\,(1-(1-2\beta)^n)\left(-(1-2\beta)^n + O((1-2\beta)^{2n})\right) \\ &= (1-2\beta)^{2n} + O((1-2\beta)^{3n}) \end{aligned}$$

Thus, we see explicitly that in the limit of large n, the loss of information is exponential in time and a power law in $1-2\beta$. It follows then, that error-tolerant coding is of a considerable advantage. [3]

Here I propose a simple model describing the impact of the genetic code and of selection on fitness. Suppose, as is likely, that the ability of the genetic code to reduce errors is equivalent to experiencing a lower rate of error, i.e. a $\beta' < \beta$. Moreover suppose, as is

[3] An argument that applies to more general types of cascades can be found in Kåhre [48, sec. 6.9]

likely, that the $I(A_n : A_0)$ value is directly related to the fitness of an organism, perhaps proportional to it, $f = L \cdot I(A_n : A_0)$ where f is the "absolute" fitness and $L > 0$ is the proportionality constant.[4] Furthermore, suppose selection acts so as to effectively remove a certain fixed fraction of the errors in each generation, so that while an organism still experiences an error rate per generation of β, the species still experiences an accumulation of errors per generation at an effective rate of $c \cdot \beta$ with $0 < c < 1$. Consequently, the difference in fitness f of the two species at the n-th generation would be

$$\begin{aligned} \Delta f &= f(\beta') - f(\beta) \\ &= L(1 - 2c \cdot \beta')^{2n} - L(1 - 2c \cdot \beta)^{2n} + O((1 - 2c \cdot \beta')^{3n}) \end{aligned}$$

Initially, when $2c \cdot \beta' n \ll 1$, we obtain a difference in fitness which is linear in n:

$$\begin{aligned} \Delta f &= L\left(1 - 2n2c \cdot \beta' + O((2n2c \cdot \beta')^2)\right) - L\left(1 - 2n2c \cdot \beta + O((2n2c \cdot \beta)^2)\right) + O((1 - 2c \cdot \beta')^{3n}) \\ &= 4nLc \cdot (\beta - \beta') + O((4nc \cdot \beta)^2) \end{aligned}$$

However, we can see from Eqn. 2.36 that for large n, any difference between β' and β would grow exponentially with n. This finding again underlines the tremendous selective advantage provided by an error-reducing code like the genetic code. As a side note about the above model, note that the apparent accumulation of errors, $(1 - 2c\beta')^{2n}$, does not mean that the population is doomed. Instead, other mechanisms, like e.g. recombination which is not described by the model, provide a molecular solution that finally halts the degradation.

The ability of evolution to decrease the rate of accumulation of errors has an intriguing connection to the topic of the next section - the construction of codes. Namely, a population of replicating organisms which experiences purifying selection can error-code information in a way that defies the decay due to errors - decay which is otherwise inevitable in channel cascades. Selection is like a strobe light that makes motion appear unmoving - the accumulation of errors in individuals is suspended when we view the population at each generation. Were it possible to encode arbitrary information in such a population,

[4] This is similar to the Eigen model of Appendix A in that we assume that the population starts with a perfected "Master Sequence" and is then degraded by errors.

the population as a whole would represent a system for error-tolerant coding. It is indeed possible, as exemplified by the genetic code. Because any changes in its structure are fatal, its has been preserved for billions of years without error.

2.3 Constructive Error Coding

Error coding typically aims at two objectives - error detection and error correction. Detection - the simpler of the two, is a process for validating the received data. Error-correcting goes further and allows the receiver to recover the original message from the (corrupted) output of the channel. It is useful to quantify the error detecting and correcting capacity of a particularly important type of codes - the block codes.[5]

Definition 2.32 *Suppose we use an [n,M]-code for transmission in noisy channels. The code is said to be an e-error detecting code (e is an integer less than n) if during transmission of any codeword, e or fewer errors occurring anywhere can be detected. Likewise, the code is an e-error correcting code if any e or fewer errors can corrected.*

A good example is the triplication code discussed earlier ($0 \rightarrow 000, 1 \rightarrow 111$). It is a 2-error detecting and 1-error correcting code.

Redundant transmission is the basis of all error detecting and error correcting schemes, as can be seen from the fundamental theorem above. Since the theorem indicates that reliable transmission is possible only at a rate no greater than the channel capacity, reliable transmission is possible only if the source entropy is reduced below the capacity of identical channel but without noise. In other words, the message being sent is compressible i.e. redundant, if only we had a noiseless channel available. Well-designed codes exploit this redundancy to help detect and correct errors.

When designing a code for many applications, one is often given certain desired level of reliability. The task of the designer is then to optimize between three criteria: the maximization of the rate of information transfer, minimization of redundancy required, and minimization of the computational cost of encoding and decoding. The penalty imposed

[5]From here on to the end of the chapter, we shall use $[n,M]$ to designate a block code on n-strings with M codewords.

by noise on the rate of transmission is very high, as the following illustrates. In the binary symmetric channel of Fig. 2.9, if the rate of error β increases to approximately 0.11, the channel capacity falls by 50% (as can be seen from the expression for channel capacity). If we found a block code that saturates the bound of the fundamental theorem, then half of the information in each codeword would be redundant.

In this section we will review schemes for building error-correcting codes. Our goal is to identify the appropriate mathematical framework for studying the genetic code within the theory of error coding. We shall also review other instances of error coding in biological systems. These may provide starting points for an extension of coding theory for application in biological systems.

2.3.1 Linear Block Codes

Linear block codes are a historically important type of coding. In a linear block code, the symbols of all codewords are numbers that satisfy certain linear equations.

First, let us discuss some of the basic ideas of error coding in block codes, of which linear codes are a special case. In a good [n,M] block code, the codewords are chosen in such a way that the codewords are spaced "far apart" in the set of possible words. More precisely, we define a metric d on the set of words, the Hamming distance, where $d(x, y)$ for words x, y equals the number of positions where x and y are different. This leads to the following decoding scheme:

Theorem 2.33 *Suppose a source produces symbols with equal probability. Each symbol is then encoded with a codeword in $\{w_i\}$ and sent down a symmetric channel whose $\beta < 1/2$. Suppose the word v is received. Then $p(v|w_i) > p(v|w_j)$ if and only if $d(v, w_i) < d(v, w_j)$.*

(for a proof, see Ash [pp.87-8][3]). In other words, the ideal observer decoding scheme is identical to this "minimal distance decoder". The set of words is therefore partitioned into subsets (sometimes spheres) around each codeword such that in each subset the words are decoded into the same codeword. There may also be regions that cannot be decoded with this scheme because they are equidistant to two or more codewords.

Definition 2.34 *A code C is said to have distance d if for any pair of distinct codewords,*

(w_i, w_j),

$$d = \min d(w_i, w_j)$$

We can observe now that if d is an odd integer and e is an integer such that $2e + 1 = d$, then C is a $2e$-error detecting and e-error correcting code.

Linear block codes are a special case of block codes. The simplest linear code is a parity check code, as follows. Suppose we wish to transmit n bits of information with the capability of detecting $e = 1$ errors. We can achieve this task by adding an $n + 1$ bit such that the sum of the bits has an even parity. Any flip of a single bit would change the parity, although two flips would remain undetected (any odd number will be detected). If $(r_1, r_2, .., r_n, r_{n+1})$ is any codeword in this code, then the digits always satisfy the linear equation $r_1 + .. + r_{n+1} = 0$ modulo 2. A more sophisticated type of a parity code is found in the ISBN of every book. Here the first nine digits are the "information digits", while the tenth "check digit" is chosen so that $10r_1 + 9r_2 + .. + 2r_9 + 1r_{10} = 1$ modulo 11. ISBN allows the parity digit to have the value 10 - written as "X". It can be shown that such a choice of coefficients guarantees that error in any one of the digits is detected.[6]

Definition 2.35 *A (n,k) linear code is an block code such that the set of codewords is the set of n-strings* $(r_1, r_2, .., r_n)$ *satisfying the m equations*

$$\begin{aligned} a_{11}r_1 + a_{12}r_2 + ..a_{1n}r_n &= 0 \\ a_{21}r_1 + a_{22}r_2 + ..a_{2n}r_n &= 0 \\ .. &= 0 \\ a_{m1}r_1 + a_{m2}r_2 + ..a_{mn}r_n &= 0 \end{aligned}$$

where the symbols and coefficients exist in a finite field (like $GF(2)$*).*

We can place the coefficients a_{ij} in a matrix A known as the parity check matrix. A linear code over field $GF(q)$ must contain q^k codewords where $k = n - rankA$. In such a code, it takes an average of n symbols to transmit k symbols from the source, i.e. the rate of transmission is $R = k/n$. Notice that an (n,k) linear code is a [n, $M = q^k$] block code.

[6]The ISBN is special in two ways: First, any interchange of two adjacent digits would also be detected Second, a valid codeword has the further restriction that 'X" cannot appear except in the parity bit.

The equations above imply that a codeword is any vector x satisfying $Ax = 0$. Suppose now that w was sent and v received. The difference between the two, $z = v - w$ is known as the error pattern vector. We call $c = Az = Av$ the syndrome of v. The decoder would attempt to determine z (and hence $w = v - z$) from c. Unfortunately, there are many possibilities. In the simple parity check code above, any single bit flip would produce the syndrome 1. Fortunately, given sufficient redundancy it is possible to construct $1-$, $2-$ etc. error detecting and $1-$, $2-$ etc. error correcting codes.

Theorem 2.36 *The minimum-distance decoding scheme for a parity check code is to pick, out of all z satisfying $Az = Av$, $z = z'$ having the lowest weight, i.e. the fewest non-zero components.*

The proof can be found in Ash [pp.96-8][3] but it makes an intuitive sense: if z' has the lowest weight of it all its peers, the distance $d(w, w + z')$ would be at least as small as $d(w, w + z)$.

Implementation of a linear decoding scheme is as follows: for every syndrome c, we would store the corresponding minimal weight error pattern z. Since the syndromes are linear combinations of the columns of A and the rank of A is $n - k$, we must store q^{n-k} pieces of data. This is a considerable amount of information. However, the linearity of the code makes it unnecessary to store all the possible words v (of which there are q^n) and the associated minimal distance decodings.

The following theorem gives a lower bound on the error-correcting capability of a linear code for binary strings:

Theorem 2.37 *Consider a linear code based on binary strings which is defined by the matrix A. Then the code can correct e errors if and only if any $2e$ columns of A are linearly independent.*

The proof of the theorem relies on the fact that the syndrome is just a linear combination of the columns. If and only if $l \leq e$ errors occur in a codeword of an e-error correcting code, then it is possible to determine the error pattern vector z from the syndrome. z would have l non-zero components. But z can be found if and only if *after* we find a linear combination of l columns that equals the syndrome, we cannot then find another such a linear combination of l other columns. Hence, any $2l$ columns are linearly independent.

The probability of reliable transmission in a linear code can be written as

$$p(e) = \sum_{i=0} nN_i\beta^i(1-\beta)^{n-i} \tag{2.36}$$

where N_i is the number of i-tuple correctible errors. One may prove that with a linear code it is possible to maintain high reliability while transmitting arbitrarily close to the bound of the fundamental theorem. Namely, Ash [3, pp.110-3] proves the following:

Theorem 2.38 *Given a binary symmetric channel with capacity $C > 0$ and a positive number $R < C$, there is a sequence of linear codes (n_i, k_i) with error bound λ_i, $i = 1, 2, ..$ where $n_1 < n_2 < ..$, $R \leq k_i/n_i < C$ and $\lim_{i\to\infty} \lambda_i = 0$*

Thus, $\frac{k_i}{n_i}$ can exceed any arbitrary R as long as $R < C$. It must be said, though, that the decoding would require resources which are exponential in n.

2.3.2 The Nucleotide Code

The importance of error coding theory in biological situations has been highlighted in a recent paper by Mac Dónaill[52] on the selection of nucleotides. The paper is also significant because it attempts to represent biological semantics in information theory.

Mac Dónaill's work is based on the observation that many other nucleotides are possible beside the four (or five) actually used, as follows. When nucleotides form base pairs, pairing follows two "complementarity rules" (Fig. 2.11). First, in each base pair one of the nucleotides is a pyrimidine (U[or T] or C) and the other a purine (A or G). Since a pyrimidine has a single ring while a purine has a larger double-ring structure, a correct base pair has a certain length. Pyrimidine-pyrimidine and purine-purine pairs are too short or too long, respectively, and would appear as either a narrowing or a bulge in the double helix. Second, the nucleotides have complementary donor/acceptor pattern in the hydrogen bonds. In a perfect base-pairing, a donor hydrogen is situated opposing an acceptor. If two donors or two acceptors are placed in opposition, the nucleotides repel each other. All of the nucleotides have three hydrogen bond forming sites, with each site being a donor or an acceptor. For instance, Guanine is an A-D-D (acceptor, donor, donor) purine. Therefore, within this system it is possible to have a genetic alphabet consisting of up to 16 different

nucleotides or subsets thereof. Indeed, experimenters have successfully synthesized all of the possible alternative nucleotides, known as "nucleotide analogues". Their existence begs the question of why nature chose only four, and why the four we have.[7]

Mac Dónaill proposes that the nucleotides nature chose are the only "nucleotide code" that provides a good tradeoff between replication fidelity and information capacity, as follows. As a first step in his argument, he represents each nucleotide by a bit string of length four. One bit (b_0) is used to indicate whether a nucleotide is a purine (0) or a pyrimidine (1). Furthermore, the three hydrogen bonds of a nucleotide can be represented with three bits ($b_3b_2b_1$), where each acceptor and donor site is labeled 0 and 1, respectively. Thus, Guanine (A-D-D purine) is $(b_3b_2b_1 : b_0) = (011 : 0)$, Cytosine is $(100 : 1)$ and Thymine is $(010 : 1)$. This binary representation makes it apparent that the nucleotides form a code in the coding theory sense.

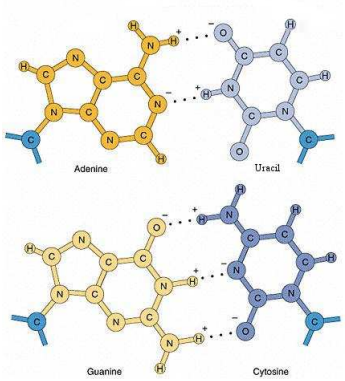

Figure 2.11: **Base pairs in RNA**
Base pairing follows two complementarity rules - the size and the hydrogen donor/acceptor pattern with donors indicated with (+) and acceptors with a (-). Adenine is special, as it forms a double rather than a triple bond although its partner Uracil is capable of forming three bonds. This does not significantly affect the following argument because the other nucleotides fit the pattern very well. Moreover, there exists an idealized form of adenine, the 2-amino-adenine $(101 : 0)$, which may have been used in early evolution[6]. Image adapted from an original by Kenneth G. Wilson.

[7] U and T are identical when we consider just the pyrimidine/purine attribute and the hydrogen bonds.

Next one can introduce a Hamming distance function $\delta(X_1, X_2)$ to measure the replicative similarity between nucleotides, i.e. how likely are two nucleotides to be interchanged during replication. For instance, $G = (011 : 0)$ and $U = (010 : 1)$ have $\delta(G, U) = 2$. Crucially, unlike other copying systems, DNA replication involves a change in how information is stored. Namely, the template sequence $GGCAT$ becomes $CCGTA$ rather than $GGCAT$. Thus, one can introduce a "complementary distance function" $\bar{\delta}(X_1, X_2)$ to measure the complementarity between nucleotides. For instance, G and C as well as A and T are "identical" in this sense ($\bar{\delta}(G, C) = 0 = \bar{\delta}(T, A)$). The smaller the value of $\bar{\delta}$ is, the more like is replication to pair the two nucleotides. If we use $\bar{X}$ to designate the complement of nucleotide X, we have a useful identity linking the two distance functions:

$$\delta(X_1, X_2) = \bar{\delta}(\bar{X}_1, X_2) = \bar{\delta}(X_1, \bar{X}_2) = \delta(\bar{X}_1, \bar{X}_2) \tag{2.37}$$

When designing any block code, an important consideration is the "distance" of the code δ:

$$\delta = \min\{\delta(X_i, X_j) | i \neq j\} \tag{2.38}$$

Thus, it is given by the minimal Hamming distance between any two (non-identical) codewords and provides the upper bound on the error-detecting capability of the code. More precisely, a block code is at most an $e = \delta - 1$ error detecting code. The complementary distance is also important, as follows. We define $\bar{\delta}$ as the minimal pairwise distance between two nucleotides, a purine R_i and pyrimidine Y_j (excluding those that match perfectly):

$$\bar{\delta} = \min\{\bar{\delta}(R_i, Y_j) | i \neq j\} \tag{2.39}$$

This quantity gives the upper bound (less one) to the ability of the code to detect malformed base pairs. Another quantity, the $\hat{\delta}$, is related to δ and measures the ability of a nucleotide code to detect translation errors - errors that preserve the purine/pyrimidine property of the nucleotide:

$$\hat{\delta} = \min\{\delta(Y_i, Y_j) | i \neq j\} \tag{2.40}$$

where Y labels pYrimidines (by complementarity, it equals distance between the purines, $\min\{\delta(R_i, R_j) | i \neq j\}$ where R labels puRines). This quantity is important because translations occur much more frequently than other errors.

For the standard genetic alphabet all three distances equal 2. In words, the nucleotide code has an inherent ability to oppose incorrect replication by at least two forces. Either two hydrogen bonds repel, or one hydrogen bond repels and the pair has the wrong geometry. In fact, the bit b_0 distinguishing a purine from a pyrimidine may be called the "parity bit" for its special compensatory role: The mis-pairings $A = G$ and $T = C$ put in opposition two hydrogen acceptors which repel only weakly (compared to the opposition of two donors). This weakness is compensated by a weighty geometrical check.

The above distance functions can be viewed as evolutionary fitness functions of any nucleotide alphabet-code. Indeed, Mac Dónaill argues that the nucleotide alphabet was likely driven by an evolutionary process - a process which strove in part to maximize those functions as well as information transmission. Given that the earliest organisms had low fidelity replicators, the argument is quite plausible.

Mac Dónaill also proposes that the standard nucleotide code represents an evolutionary optimum, as follows. Consider alternative nucleotides codes where $\hat{\delta}$ is different from 2. As expected, $\hat{\delta} = 1$ codes are inferior to $\hat{\delta} = 2$ codes in their resistance to translation errors. But surprisingly, $\hat{\delta} = 3$ codes are actually worse than $\hat{\delta} = 2$ codes when transversion errors are concerned. For instance, the $\hat{\delta} = 3$ code $\{(011 : 0), (011 : 1), (100 : 1), (100 : 0)\}$ has no hydrogen bonds repelling the association $(011 : 1) = (100 : 1)$. This is a necessary property of all $\hat{\delta} = 3$ codes with four nucleotides, as can be seen from Fig. 2.12. Because we know that without proofreading the rates of transversions are not much greater than the rates of transitions[86], this is an important shortcoming of such codes. Finally, while $\hat{\delta} = 4$ codes do not exist, there are codes with overall distance $\delta = 4$. Such codes offer the highest resistance to all classes of errors, but they can only have up to two codewords (i.e. nucleotides) and hence have a low rate of transmission R. More precisely, if $n = 4$ is the length of a codeword and k is the log of the number of codewords, then $R = k/n$ is not greater than .25 for $\delta = 4$. Codes with $\hat{\delta} = 2$ like the natural code with $k = 2$ are [4,4] block codes and hence have a transmission rate double the above (although in principle a $\hat{\delta} = 2$ can have up to 8 nucleotides i.e. $k = 3$, this leads to chemically unfavorable acceptor/donor patterns). Evolution seems to have found in the natural nucleotide code the optimal engineering tradeoff between error resistance and transmission rate. The nucleotide code does differ from the block codes discussed early because, strictly speaking, it provides

neither error detection nor error correction: while normally detection and correction take place *after* an error occurs, the nucleotide codes helps prevent errors happening in the first place.

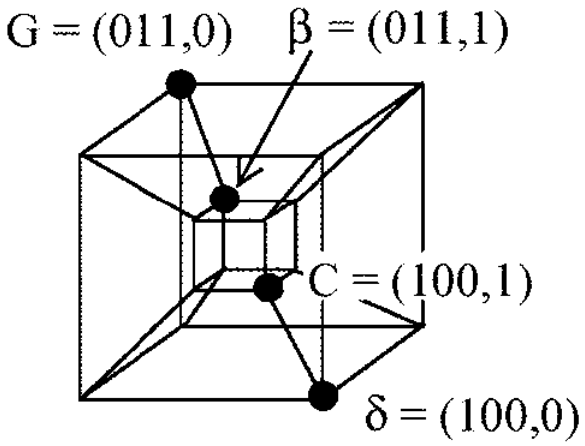

Figure 2.12: **Graphic representation of a sample $\hat{\delta} = 3$ code**
The vertices of the hypercube correspond to words (thick vertices are the chosen codewords) and have coordinates specified by four-bit strings. Purines occupy the outer cube and pyrimidines the inner. Path lengths on the graph give the Hamming distances between them, $\delta(X_i, X_j)$. A code with $\hat{\delta} = 3$ containing two purine-pyrimidine pairs offers little resistance to transversions because the two purines (and pyrimidines) have one identical bit (b_0), and hence to satisfy $\hat{\delta} = 3$ must have three perfectly complementary hydrogen bonds and this is exactly the condition for perfect hydrogen pairs.
Image source: D.A. Mac Dónaill

It should be remarked that Mac Dónaill concedes that the selection of the nucleotide set was also driven by other biochemical forces (like chemical stability and other that I discussed in Chapter 1). For instance, chemical reasoning may explain the presence of Adenine, that lacks a third hydrogen bond. Moreover, he relies on biochemistry to explain why an even parity alphabet was chosen by nature rather than an odd parity one (which can be obtained by swapping the acceptor/donor patterns of the purines with the pyrimidines).

The two alphabets are identical from the point of view of error resistance and information content, but the natural alphabet is preferred on chemical grounds. Overall, Mac Dónaill's provides an interesting case of biological error coding while again emphasizing the importance of error-reducing evolutionary forces in primordial times.

2.3.3 Gray Codes

A type of codes known as Gray codes provides an interesting example of how physical property could be represented symbolically, somewhat in the way amino acids are represented by codons.

Originally designed for analogue electronic systems, Gray codes ensure that a slight change in the property in question leads to only a slight change in the symbols. One application of Gray codes is magnetic discs[36], as follows. The surface of the disc is divided into sectors, where each sector is labeled with a three digit binary string (in practice, it is much longer but three is sufficient for our discussion). There is a natural ordering of the sectors on the disc, $0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, \ldots$ induced by the direction of its rotation. It may seem reasonable to use the usual ordering of binary numbers for the labels: $000, 001, 010, \ldots$. Yet, such a labeling may produce a high rate of errors. For example, consider the labels of sector 3 (011) and of sector 4 (100). If the computer attempts to read the sector label while the magnetic head is between two sectors, error is likely to occur. In fact, if the reading is slow, the output may be *any one* of the eight three-letter strings. To avoid this problem, the labels use a Gray code as shown in Fig. 2.13. Notice that with a Gray code, if the head is between the two sectors, it will reliably return either 010 or 110. This is because two adjacent labels in a Gray code will always differ by exactly one bit. The first and last labels (here 000 and 100) also have this property. In fact, the code can be represented as a graph created by placing each codeword in a vertex of a cube. The code would be a Hamiltonian cycle.

Gray codes are not error-correcting, but they do minimize the effect of physical errors on their representation. This is achieved by building a cyclical ordering of the physical property and a parallel cyclical ordering of the symbolic representation - the codewords. In an interesting paper, Swanson[80] proposed that the genetic code is a Gray code. The paper proposes that the codons were linked in a cycle due to translation errors in the third

Sector	Label
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Figure 2.13: **Eight sector disc where sectors are labeled with a Gray code**
The disc has a circular topology, so sector 7 is followed by sector 0. Notice that two adjacent sectors are always labeled by codewords which differ by exactly one bit. Example taken from Hamming [36]

and first position. Moreover, the paper used data on the physico-chemical similarity of amino acids and created a cycle of the chemical properties of amino acids. The two cycles run in parallel, such that errors in the codons lead to motion along the cycle of chemical properties. Because nearby amino acids in the physical cycle also lie nearby in the codon cycle, such a code is error-minimizing.

In my view, the Swanson paper is open to several criticisms. First, the data the paper uses for finding the most similar amino acids was taken from comparing the rates of non-synonymous substitutions (i.e. substitutions that change the amino acids) in two related conserved genes (like in Grantham [33]). The argument being that the more likely a substitution is to occur, the more closely related two amino acids are. Admittedly, no better data was available at the time. The problem in such an argument is that the patterns of such substitutions are strongly "contaminated"[8] by the structure of the genetic code. Namely, the substitution that are most likely to be observed are those that represent Hamming-distance one changes in the genetic code, with chemical similarity playing a secondary role. With this method of determining chemical similarity, one is liable to mistakenly see amino acids placed nearby in the codon space as chemically most similar.

In fact, a recent study that obtained a genuine chemical distance between amino acids[90]. The new data puts the Swanson paper in question, since it finds that amino acids that are most similar do not always lie near to each in the codon space, unlike what Gray code would have. For instance, Histidine is chemically much closer to Alanine (which is Hamming-distance 2 away) than to Glutamine (which is just a single third-position error way from it).

Secondly, the error-minimizing properties of a Gray code are subtly different from what the paper suggests. The structure of a Gray code is such that errors in the physical domain cause small changes in the symbolic domain. However, it has only a limited capability to minimize the impact of errors in the symbolic domain on the physical domain. In the Hamming example, the Gray code used for labeling sectors on a magnetic disc does not minimize the effect of bit flips. Of the 3 possible bit flips, only 2 will produce the label of a nearby sector. In the case of the genetic code, if we assume that the genetic code is a Gray code, then the physical domain corresponds to amino acids and the symbolic domain are the codons. Yet, such a code would do even worse than the disc label code - only minimizing the impact of 2 out of the 9 possible Hamming-distance one changes.

Third, it is not clear that the codons can be ordered in a cycle based on the probabilities of the different errors. The paper suggests that such an ordering is possible because some errors (like transition errors in the third position) are much more likely than others (transversion errors in the second position). However, careful examination of the data does not support this ordering. For instance, Leucine is coded by six codons - the four codon block CUU, CUC, CUA, CUG and the double block UUA, UUG. Thus, Leucine is within close distance of not two but five amino acids. While Phenylalanine is closest to Leucine (requiring a transversion in the third position or a transition in the first), the next most likely errors from Leucine (transversions in the first position) lead to Isoleucine, Methionine or Valine! Presumably, the genetic code is adapted to all of those errors. Since a Gray code minimizes the effect of exactly two possible errors (forward and backward in the cycle), the genetic code is not a Gray code.

Nevertheless, the Gray code idea contains an essential element of a correct mathematical theory of the genetic code. Unlike the linear codes discussed earlier, the symbolic representation in a Gray code is designed in view of some scale of physical properties.

Thus, the code is built with consideration of differences in the physical impact of errors. In contrast, linear codes do not distinguish between the impact of different errors, instead distinguishing only between errors and no errors. The evolution of the genetic code must have considered the physical impact as well.

Chapter 3

Analysis of the Genetic Code

The goal of this chapter is to combine the biology of Chapter 1 and the information theory of Chapter 2 to study the structure of the genetic code. In Chapter 1 is was argued that the genetic code evolved very early in the history of life and then froze. It was also argued that error reduction has been the dominant evolutionary force shaping the code. Since the code is a product of evolution, its structure is an "imprint" of the evolutionary forces that acted upon it. It would seem possible then to determine the strength of those ancient evolutionary forces by studying the code. The ultimate goal is to use the structure of the genetic code to shed light on the nature of very early organisms. Especially interesting is the question of temperature in which they lived. Previously, DiGiulio [15] tried to address the temperature question by studying the biochemical properties of the code while Ardell [2] proposed a heuristic method to guess the relative rates of errors that acted on the code. However, to the best of my knowledge, this is the first time a quantitative information-theoretic approach has been used to study the genetic code and address both the temperature and the error rate questions.

I describe below some of the methods I have developed for the task and their results. Most of them give a small part of the information the genetic code should ultimately yield and thus it is my hope that they can be improved or supplanted in the future. Although I have also implemented and computed many other methods, I report here only those methods that have either produced results or that I believe may be of a particular interest in the future. Although all of the methods are information-theoretic, they are

quite different from one another. While one class of methods builds upon certain relations between the genetic code and the strength of error-minimizing evolutionary forces, the other class studies the genetic code using evolutionary fitness functions.

This project has faced obstacles of both a biological and an information-theoretic nature. On the biology side, today we have but an incomplete understanding on the evolution of the genetic code. For instance, one is forced to speculate on whether the code evolved in response to mutations or mistranslations, and on how this evolution may have taken place. An issue which is perhaps even more important is the lack of a theory for building codes like the genetic code (see subsection 3.1.1). Possession of such a theory would make it far easier to devise and test new methods for solving the reconstruction problem. Consequently, it is impossible to provide a proof for the validity of the methods proposed below, or even to test them from first principles. Nevertheless, whenever I proposed a method, I have provided an argument for its validity. Although most methods below are based on heuristic arguments, the transmission function method of subsection 3.4.1 directly uses the ideas developed in Chapter 2.
The outline of this chapter is as follows:

- Preliminaries
- Relational Reconstruction Methods
- The Tolerance-Safety Method
- Optimization-based Reconstruction Methods

3.1 Preliminaries

In this section we will address mathematical questions that pertain to all of the different methods. First, we will discuss the nature of the genetic code and how its evolution could be modeled and reverse engineered. Second, we will introduce a mathematical model for the problem that would be used throughout the chapter.

3.1.1 What kind of code is the genetic code?

The genetic machinery can be represented in different ways within the framework of information theory. One view is to consider the nucleotides as symbols being sent through a noisy channel. Here we will instead consider the nucleotides as encoded representation of the actual chemical message - a sequence of N chemical instructions generated from a source alphabet of 21 symbols (one of 20 amino acids or the stop codons). In this "genetic channel", the "sender" is the ancestral organism that evolved a protein, and the "recipients" are either its protein-synthesizing machinery or its descendant organisms.

Like many other codes studied by information theorists (e.g. Hamming, Golay, Cyclic, Turbo and others) the genetic code is a block code, where the block size is 3 and the alphabet consists of 4 nucleotides-symbols. Since there are 21 possible source symbols, we get $k = \log_4 21 \approx 2.196$. Therefore, the maximal transmission rate is $R = \frac{k}{n} = \frac{2.196}{3} \approx 0.732$. However, proteins do not include all 21 symbols-amino acids with equal probability, leading to a much lower transmission rate. Also, because any one of the words could be found in the encoded sequence, i.e. all the words are codewords, the genetic code has no error detecting or error correcting ability unlike most block codes and instead has an error-reducing ability. [1]

There are at least two other very substantial differences between the genetic code and most other block codes. First of all, the design of the genetic code minimizes both the probability of error *and* the damage done to a protein when an error does occur [2]. This can be seen from the fact that codons connected by a single error especially in the first or third position lead to chemically related amino acids (Fig. 1.12). Unfortunately, such a "semantic code" is not described in any of the references on coding theory I found.

Indeed Shannon restricted his theory to studying the manipulation of symbols, without reference to what he called "semantic aspects of communication" [76]. Consequently, one is forced to rely on heuristics when trying to understand how such a code may be designed and reverse engineered. Second, whereas the block codes discussed in Chapter 2 are optimized

[1] Synonymous codon blocks might be called "error correction", but in any case, because of the low redundancy the code "corrects" only some of the Hamming distance 1 errors.

[2] Here and below, when referring to amino acids we include in the discussion the chain termination symbol encoded by the stop codons, unless noted otherwise.

to the symmetric channel, the genetic code performs better when the errors have a more complex distribution. It has the highest tolerance to transition errors in the third codon position, slightly lesser tolerance to transversion errors in the third codon position, even less tolerance to errors in the first codon position, and very low tolerance to errors in the second codon position.Third, the error-reducing evolution of the genetic code was not free to chose an optimal block size or the nucleotide alphabet. Rather, it was dictated by the structure of the existing genetic machinery (section 1.2).

We proceed to analyze the structure of the genetic code by inspiration from the case of simpler codes, like the linear codes of Chapter 2. First, recall that codewords of a block code could be represented as a graph where vertices are the codewords, and edges are the possible errors between them (cf. the graph for the nucleotide code in Fig. 2.12). However, the codons of the genetic code are linked with errors occurring with different probabilities. Thus, rather than using a simple graph, we use a directed graph in which every vertex is connected to every other with two edges, one going in each direction. Moreover, each directed edge $e_{ij} = v_i v_j$ has a weight w_{ij}, where $w_{ij} = P(v_i, v_j)$ - the probability of a mutation taking codon-word v_i to v_j. Each of the vertices-codewords must be assigned one amino acid. No amino acid can be without associations, or mathematically, the map from the set of words to the set of amino acids-source symbols must be surjective. Next, for every pair of amino acids, (A_p, A_q), we can write a distance function, D such that $D(A_p, A_q) = 0$ whenever $A_p = A_q$. We require that $0 < D(A_p, A_q) < m$ for all pairs (A_p, A_q) but unlike in metric spaces, D need not be symmetric or satisfy the triangle inequality. In one definition of distance of amino acids, the function is nearly symmetric. The D function can also be written as a complete digraph with weights representing distances in each direction.

When we considered simpler codes, one of the design goals was to minimize the probability of an irrecoverable error. Many authors (e.g. Gilis [31]) have suggested that perhaps the genetic code is the solution to a related minimization problem, involving probabilities of error and chemical distances. Namely, the design of the genetic code could minimize the expected damage $\langle D \rangle$:

$$\langle D \rangle = \sum_{i,j} D(A_i, A_j) w_{ij} \tag{3.1}$$

$$= \sum_{i,j} D(A_i, A_j) P(v_i, v_j) \tag{3.2}$$

where A_i, A_j are the amino acids at vertices (codons) i and j, respectively. Since the probability of a double error is very low, it is likely that such errors can be excluded from the sum. One notable shortcoming of this fitness function that it does not consider the evolutionary tradeoff between increasing catalytic versatility by increasing the number of encoded amino acids and reducing the fidelity of the translation machinery.

I performed simulations of code evolution in which Eqn. 3.1 was the fitness function. I found that the simulated codes had a degeneracy structure quite similar to the genetic code, consisting of blocks of amino acids with the third position partially degenerate. This result does not by itself show that evolution of the genetic code involved the minimization of $\langle D \rangle$ or any related fitness function. Yet, it does show one mechanism by which the code may have evolved. If so, then the model may explain some features of the genetic code.

Perhaps more importantly, calculations using the model confirmed the existence of a certain relation between damage and error rate, namely that a high error rate leads to greater error reduction. Indeed, the frequent third position errors are very often synonymous. The model was less conclusive about errors with low probability, sometimes showing that they lead to high D (like the rare but damaging second position errors in the genetic code), and sometimes giving moderate D. This somewhat inconclusive finding may well be the result of the flaw in the fitness function or the finite time available for simulated evolution.

Even with this simulation many questions remain unanswered about the natural genetic code:

- Which was the actual evolutionary fitness function?
- Did evolution reach a global optimum, a local optimum, or did it stop before any optimality was reached?
- How far is the genetic code from the global optimum?
- Is the position bias of errors an ancient feature of the genetic machinery to which the code adapted, or a recent adaptation of the machinery to the code?

3.1.2 The Mathematical Model

Recall that our goal is to extract information from the structure of the genetic code, and specifically information about the temperature of the environment in which the code evolved. We have noted earlier that under the conditions of an RNA world, there exists a correlation between the temperature and the nucleotide frequencies, namely, the G+C content of the genome. Also, by studying the genetic code it should be possible to determine the relative frequencies of the different errors to which the code adapted.

We now introduce a mathematical model that describes the targets of our reconstruction. This model represents an application and extension of a standard model used by molecular biologists (cf. [50, 51]). First of all, we wish to represent the rates of different errors. Since the genetic code is primarily adapted to reduce the rates of point errors and not frame shift errors (see Chapter 1), we shall only model the former type of errors. Moreover, we shall assume that errors occur to each nucleotide independently of other nucleotides like in a memoryless channel. Also, recall that it is not known whether the genetic code is adapted to mutations or mistranslations or some combination of them, though because there was no DNA genome during much of code evolution, the genetic code is unlikely to be adapted to transcription errors. Thus, in our discussion, the terms "errors" or "substitutions" are intentionally ambiguous, referring to either mutations or mistranslations, unless specified otherwise. However, unlike in phylogenetic literature, we shall not make any distinction between an error and a substitution and they are equivalent.

There are 12 possible errors, and their probabilities shall be designated M_{XY}, where X is the original nucleotide and Y is the new nucleotide. M_{XY} is the conditional probability of $X \to Y$ error given that X was found. These can be placed into the matrix M:

$$M = \begin{pmatrix} M_{UU} & M_{CU} & M_{AU} & M_{GU} \\ M_{UC} & M_{CC} & M_{AC} & M_{GC} \\ M_{UA} & M_{CA} & M_{AA} & M_{GA} \\ M_{UG} & M_{CG} & M_{AG} & M_{GG} \end{pmatrix} \tag{3.3}$$

This is a stochastic matrix, i.e. a matrix whose columns sum to unity. Hence the diagonal elements M_{XX} for all X - the probabilities that X does not experience an error - are fixed

by the off-diagonal elements. Namely,

$$M_{XX} = 1 - (M_{XY_1} + M_{XY_2} + M_{XY_3}) \tag{3.4}$$

where Y_1, Y_2, Y_3 are the three other nucleotides. Also, because DNA and (especially) RNA are much more stable in double-stranded form, primordial organisms likely used double-stranded rather than single-stranded genomes for much of their history. This means, as was discussed in subsection 1.1.2, that the rate of mutations might be approximately equal to the rates of the complementary mutations. For example, if the nucleotides are equally likely, then $M_{CU} \approx M_{GA}$.

In addition to the error rates, let π_X denote the probability (frequency) of each nucleotide in the genome where X takes the four values U, C, A, G (in RNA). The nucleotide probabilities π_X can be written as a state vector of the genome:

$$\pi = \begin{pmatrix} \pi_U \\ \pi_C \\ \pi_A \\ \pi_G \end{pmatrix} \tag{3.5}$$

Notice that the first component was labeled π_U as would be appropriate for RNA rather than π_T as would be appropriate for DNA. Indeed, biological evidence suggests that DNA was not used during most of the evolution of the genetic code [24], but whatever the case may be, it does not affect the computational methods we shall introduce, only the interpretation of the results.

Taking those four variables and the M_{XY} above, we can express the absolute (non-conditional) probability of errors: the probability of finding nucleotide X and seeing it experience the error $X \to Y$ is $\pi_X M_{XY}$. This probability, rather than just M_{XY} is the source of the evolutionary force acting on the genetic code, because one expects more evolutionary adaptation to errors in the more frequent nucleotides, and less to errors in the rarer ones, all else being equal. It is for this reason that the error reduction hypothesis makes it possible, in theory, to gauge the nucleotide frequencies in the primordial genome and not just the error rates. As was mentioned subsection 1.1.1, if the primordial organisms were thermophiles with RNA genomes, their $\pi_G + \pi_C$ value would have been elevated (to at least 60%). Also, the genetic code responds only to errors in the coding parts of the

genome. Thus, if for example, the primordial organisms had large introns (non-coding sequences) enriched with Guanine, π_G would not account for them. However, the low fidelity of the primordial genetic machinery discussed in Chapter 1 weighs strongly against the existence of such large non-coding sequences. Thus, the values in the vector π are without reservation the composition of the genome.

One of the advantages of this model is the ability to calculate the frequencies of the codons in the genome. In the first order approximation, we would take the statistical mean - the product of the probabilities of its nucleotides. Thus, for example, the probability of finding the codon CAG would be $\pi_C\pi_A\pi_G$. This simple approximation is supported by data for modern organisms [46]. Furthermore, the probability of a given amino acid being found in a protein would be calculated by taking the sum of the probabilities of its codons. This approximation agrees well with the data in modern organisms. For example, Gilis [31] notes that degeneracy correlates well with frequency, as expected when π is uniform.

If indeed evolution was driven mainly by mutations, then M could be either one of two quantities - instantaneous error rate $I + Q$ (where I is the $4x4$ identity matrix) or the cumulative error rate over time, designated $P(t)$, where t is time. The matrices P and Q are related as $P(t + dt) = P \cdot (Id + dtQ)$. Thus,

$$P(t) = \exp(tQ)$$

Moreover, if we assume that M is a time-homogeneous Markov process, we obtain a Markov chain:

$$\pi \rightarrow M\pi \rightarrow MM\pi \rightarrow \ldots \tag{3.6}$$

Moreover, π may be stationary with respect to M, i.e. $\pi = M\pi$. In fact, it can be shown that because all the components of Q are strictly positive probabilities, the stationary state is attractive and unique (Ash [3, p. 181]). Naturally, no Markov chain exists for translation errors since they do not act on the genome composition.

Next, it is convenient to introduce an overall rate of error ϵ - the probability that a randomly chosen nucleotide would experience an error:

$$\epsilon = \sum_{Y \neq X} \pi_X M_{XY} \tag{3.7}$$

The discussion in Chapter 1 (subsection 1.3.1) suggests that $\epsilon \ll 1\%$, and perhaps as low as 10^{-5} or lower.

Finally, recall that errors may have a position bias. We quantify this effect by three weight factors, p_t, with $t \in \{1, 2, 3\}$ and $p_1 + p_2 + p_3 = 1$. Namely, the absolute probability of $X \to Y$ error occurring at position t is $\pi_X M_{XY} p_t$. When we measure the translation errors in modern organism, we find some bias towards errors in the first and third positions [63]. Haig and Hurst [35] suggest $(p_1, p_2, p_3) = (\frac{5}{16}, \frac{1}{16}, \frac{10}{16})$, while other authors quantify the bias in a slightly different way (cf. Gilis [31]). The case of no position bias (as expected if the code is adapted to mutations) corresponds to $(p_1, p_2, p_3) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

3.2 Relational Reconstruction Methods

As discussed in subsection 1.3.2, the evolution of the genetic code likely took place stochastically. As the code evolved, the tolerance to each of the 12 substitution errors likely grew in step. However, the rate of this growth was likely dependent on the probability of each error. The greater the probability, the greater the growth in error reduction to it. That suggests that a promising method of calculating the pair π, M from the genetic code g would be to relate the evolutionary forces represented by π, M with how well the genetic code reduces each of the errors. Indeed, I have developed several methods that use this idea with considerable success. One large advantage of the relational approach to the reconstruction problem is that it is valid whether the natural code is "optimally error-reducing" or less so. Indeed, while some authors argued that the code is the optimal code out of some 10^{84} alternatives (e.g. [31]) it is not likely to be, for several reasons discussed in subsection 1.3.3.

It should be noted though, that because the number of codes is large but finite, while the number of π, M values is $\aleph_1$ it is impossible to determine π, M from the genetic code g, i.e. there is no surjection from the set of genetic codes to π, M. As we will see shortly, when the structure of the code was related to the values of π, M, we found a unique solution. This solution is just one of an uncountably many other values of π, M that may have propelled the evolution of the given genetic code. However, this non-uniqueness need not be biologically significant: the biological problem is solved as long as all of those alternative

π, M values are close to the calculated one, and they are likely to be.

In this section, I will describe the following reconstruction methods:

1. The Degeneracy Counting Method
2. The Fidelity Attenuation Method
3. The Stationary Genome Composition Method

3.2.1 The Degeneracy Counting Method

The Degeneracy Counting Method relates the probabilities of error M with the pattern of degenerate amino acids in the genetic code. It is interesting not for its results, but rather because it is a simplistic version of other far more successful methods, that are at the same time better-developed and more complex.

Recall that the adaptation of the genetic code to an error reflects the probability of that error. The higher the probability of error, the greater the adaptation. The basic idea here is to use the degeneracies of the amino acids as a proxy for the adaptation of the genetic code. However, we shall look at the "degeneracies of an error" rather than at the degeneracy of a single amino acid, as follows. Take any substitution $X \to Y$ and count the fraction of instances, t_{XY}, in the genetic code in which this substitution leads to a synonymous codon (making it a "silent" error). The genetic code (Fig. 1.7) contains 64 codons with three nucleotides each implying that each nucleotide appears in exactly $48 = 3/4 \cdot 64$ instances, sometimes twice or thrice in the same codon. The substitution $X \to Y$ can occur in any one of those places, but it is silent at only a fraction t_{XY} of them. It would be more appropriate to think of the substitutions as occurring in the genome, but if all codons are approximately equally likely it is sufficient to look at the genetic code: the outcome of a substitution (whether the error is silent or not) is determined by the code alone. To give a concrete example of a calculation, consider the error $U \to C$. When it occurs at the first, second and third position in the codon it is silent in 2, 0 and 16 instances, respectively, making $t_{UC} = \frac{2+0+16}{48}$. The tally for all the errors is displayed in Eqn. 3.8. It is apparent that the genetic code has a markedly higher tolerance (error reduction) to the four transition errors over tolerance to the eight transversions. Notably,

the matrix is symmetric because whenever $X \to Y$ is silent, the reverse error $Y \to X$ is silent as well.

$$t = \frac{1}{48}\left(\begin{array}{c|cccc} To\backslash From & U & C & A & G \\ \hline U & * & 18 & 9 & 8 \\ C & 18 & * & 11 & 8 \\ A & 9 & 11 & * & 15 \\ G & 8 & 8 & 15 & * \end{array}\right) \tag{3.8}$$

The t_{XY} value quantifies how much the genetic code evolved to tolerate the error $X \to Y$. This reflects the probability of the error because evolution in the genetic code is assumed to be in proportion to the probability. More precisely, in this method we propose that the t_{XY} values are directly related to the rates of errors, i.e.

$$t_{XY} = f(\pi_X M_{XY}) \tag{3.9}$$

for all X, Y with $X \neq Y$ where f must be a strictly increasing function. Monotonicity of f reflects the fact that increasing the error rate must lead to a higher evolutionary pressure and subsequently improved error reduction in the genetic code. Thus, the right hand side of the Eqn. 3.9 is the evolutionary pressure while the left hand side gauges the response.

One can "reverse engineer" the code by giving an explicit equation for f and calculating the t_{XY}, as we did above. For the function f, in the simplest case we take the linear relation $f(x) = cx$ with c a constant. At this point we do not have enough equations to solve for both π and M. Specifically, we have 16 variables (plus c) and 12 equations of the form 3.9. We also have the relations

$$\begin{aligned} \epsilon &= \sum_{Y \neq X} \pi_X M_{XY} \\ 1 &= \sum_X \pi_X \end{aligned}$$

where the overall error rate ϵ is a parameter. Thus, we may be able to solve for all the variables if we find just three additional equations, but that is a challenging task. Instead, as a first order approximation we assume that all the nucleotides are equally likely ($\pi_X = 0.25$), and hence all codons are equally likely as well. In this case, the conditional

probabilities M_{XY} are proportional to elements of t:

$$M_{XY} = \frac{1}{0.25 \cdot c} t_{XY}$$

Results

The M_{XY} values we find are displayed in (Fig. 3.1). These are quite close to what is expected and resemble what is observed in modern organisms (cf. Fig. 1.9). Like in modern organisms the rate of transitions is several times greater than the rate of transversions and $M_{AG} > M_{CU}$. The reconstructed error rates are also consistent with rates of replication errors that are to be expected in a double-stranded genome, where the rates of errors should approximately equal the rates of their complements.

$$M = \left(\begin{array}{c|cccc} To\backslash From & U & C & A & G \\ \hline U & * & 0.0052 & 0.0026 & 0.0023 \\ C & 0.0052 & * & 0.0032 & 0.0023 \\ A & 0.0026 & 0.0032 & * & 0.0043 \\ G & 0.0023 & 0.0023 & 0.0043 & * \end{array}\right)$$

Figure 3.1: **Degeneracy Counting Method: Reconstructed** M

The reconstructed conditional probabilities of error are based on setting the parameter $\epsilon = 1\%$, but the error rates scale linearly with the value of ϵ. This means that the relative magnitudes of the error rates remain unchanged. Asterisks are placeholders for the diagonal elements in the matrix, which can be easily found from Eqn. 3.4.

Although the data is interesting, the assumption that all nucleotides are equally likely defeats the main purpose of the project - to find the nucleotide frequencies. The method has other shortcomings: First, it assumes that all codons are equally likely in the genetic code, despite the fact that their frequencies can vary by about an order of magnitude. Second, it considers only silent errors - errors between synonymous codons, ignoring the fact that the structure of the genetic code has adapted to reduce the impact of even non-synonymous errors, as discussed in Chapter 1. Third, it is unable to distinguish forward and reverse errors. Four, it assumes that an error is equally likely to occur in all three codon positions.

A slightly improved version of the method takes into account non-synonymous substitutions. For this we need to introduce a measure of chemical relatedness of amino acids. One such measure, already mentioned in chapter 2 is the EX - "Experimental Exchangeability" (EX) value [90]. It is an experimentally obtained quantitative measure of how efficient a resulting protein is, on average, when a substitution error replaces an amino acid A_i with an amino acid A_j. For synonymous substitutions $EX = 1$, while non-synonymous ones get fractional values: $0 < EX < 1$. We set $EX = 0$ in cases when an amino acid codon is replaced by a stop codon or vice versa, because the resulting protein likely loses all catalytic ability (interchanges of two stop codons are $EX = 1$). Finally, if we assume the errors have no position bias, we get the matrix T ("T" stands for Tolerance):

$$T(\pi \text{ is uniform}) = \frac{1}{48}\left(\begin{array}{c|cccc} To\backslash From & U & C & A & G \\ \hline U & * & 25.6 & 20.0 & 18.0 \\ C & 24.4 & * & 22.0 & 19.1 \\ A & 17.5 & 20.6 & * & 25.0 \\ G & 16.9 & 19.9 & 25.5 & * \end{array}\right) \tag{3.10}$$

The quantities T_{XY} are greater than the t_{XY} before, since T contains all the synonymous terms of t with additional non-synonymous terms that are not included in t. T_{XY} has a biological interpretation - it is the mean similarity in chemical properties resulting from the error $X \to Y$. The values of T_{XY}, like of EX, fall between 0 and 1.

Comparison of t and T shows that the terms approximately maintain their relative magnitudes: if $t_{XY} > t_{IJ}$, then typically (but not always) $T_{XY} > T_{IJ}$. This general lack of change is not surprising, because adaptation to increasing error rates causes an increase in both the number of synonymous substitutions and increase in the EX for non-synonymous ones, leading to a correlation between the contribution of both mechanisms to tolerance T_{XY}. Another cause for the lack of change is this: Although there are many non-synonymous substitutions, amino acids tend to differ considerably from each other i.e. $EX << 1$, and thus the total contribution is modest.

Moving forward, we again introduce a linear function

$$T_{XY} = c \cdot \pi_X M_{XY} \tag{3.11}$$

with $c > 0$. The resulting error rates (not shown) show little change compared to the

previous values. If $M_{XY} > M_{IJ}$ based on the old matrix, then typically $M_{XY} > M_{IJ}$ based on the new matrix. This is due to the monotonicity of the relation in Eqn. 3.11. Because of monotonicity, the results are not significantly affected by the details of Eqn. 3.11, even if the equation is non-linear. This will continue being a valuable property when subsequently we introduce more sophisticated relations to replace Eqn. 3.11.

3.2.2 The Fidelity Attenuation Method

The evolution of the genetic code tended to reduce the probability that a substitution of nucleotides leads to an error in the amino acids. This process gradually increased the average probability of correct transmission through the "genetic channel". Let us call the average probability of correct transmission - the "fidelity" function Φ of the code. If $p(x, x)$ is the probability that amino acid x is placed as input and received in the output, then the fidelity is just:

$$\Phi(\pi, M) = \sum_x p(x, x) \tag{3.12}$$

As an illustration, here is the $p(x, x)$ term of Histidine:

$$\begin{aligned} p(His, His) &= \pi_C \pi_A \pi_U M_{CC} M_{AA} M_{UU} + \pi_C \pi_A \pi_U p_3 M_{CC} M_{AA} M_{UC} \\ &\quad + \pi_C \pi_A \pi_C M_{CC} M_{AA} M_{CC} + \pi_C \pi_A \pi_C p_3 M_{CC} M_{AA} M_{CU} \end{aligned}$$

The genetic code is reflected in Φ through the terms corresponding to silent substitutions of the nucleotides. For example, in the expression above, those are the second and the fourth terms. For computational simplicity, we ignored double errors which make a negligible contribution when $\epsilon \approx 0.01$ or lower.

In this method we link the *derivatives* of the fidelity function to the rates of the different errors, as follows. Suppose π and M each have a certain value and we proceed to increase the probability of an error $X \to Y$ while leaving all else unchanged. How fast would Φ fall? Of course, the answer depends on how well the genetic code is adapted to reducing the impact of $X \to Y$. If it is one of the more frequent errors, then the fall would be relatively slow because the genetic code is highly adapted to reducing it. This means that the derivative of Φ with respect to each error rate is a measure of the probability of that error. Unfortunately, a similar argument cannot be made for nucleotides: unlike with error

rates, an increase in the probability of a nucleotide need not lead necessarily to a decrease in Φ.

To quantify this idea, it is necessary to estimate the value of Φ at a certain point π, M. Since we have at this stage no means to find π, we assume that π is uniform. As to M, we suppose that the error rate is zero to begin with (M equal to the identity matrix). Now, for each error $X \to Y$, we measure the value $d_{XY} = \frac{\partial \Phi}{\partial M_{XY}}$ evaluated at the chosen point. Next, we make the ansatz that M_{XY} is related to d_{XY} through a decreasing function h - the more negative the rate of change, the lower the tolerance, and hence, the lower the M_{XY}:

$$d_{XY} = h(\pi_X M_{XY}) \tag{3.13}$$

The relationship is not quite linear - we expect that as $d_{XY} \to 0$, the error rate should grow unbounded. One simple model is $d_{XY} = -\frac{c}{\pi_X M_{XY}}$ with $c > 0$.

Results

The calculated conditional probabilities of error are shown in Fig. 3.2. Since this method is sensitive to the degeneracy only, it is unable to distinguish the rate of an error and the rate of the error in the reverse direction - we have seen this problem before. The gap between the rates of transition errors and that of transversion errors is slightly less than what we have found with the degeneracy counting method. However, it is dependent on h. Setting $h(x) = -\frac{c}{x^3}$ yields a larger gap.

Overall, although this method is conceptually interesting, it does not provide a real improvement over the degeneracy counting method.

3.2.3 The Stationary Genome Composition Method

So far we have talked about "errors" or "substitutions" in general without making assumptions about their nature i.e. whether the genetic code has adapted to translation errors or mutations. Here we shall consider only the latter case. We have noted in subsection 3.1.2 that mutations act on the genome composition thus creating a Markov chain $\pi \to M\pi \to MM\pi \to \ldots$. After a sufficiently long time the composition of the genome

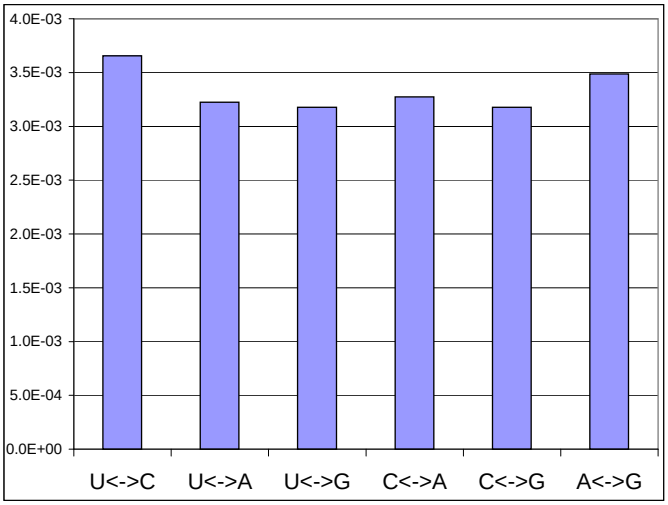

Figure 3.2: **Fidelity Attenuation Method: Reconstructed** M

The reconstructed conditional probabilities of errors at the time the genetic code stopped evolving. Because we assume that the π_X values are always 25%, those bars are exactly 4 times greater than the absolute rates of error. The rates of forward and backward substitutions are found to be identical, as before. The scale of the bars reflects an assumed overall error rate of $\epsilon = 1\%$ but ϵ has no effect on the relative magnitudes of the errors.

reaches a stationary genome composition

$$\pi = M\pi$$

These are four independent equations, one for each nucleotide X, which we can explicitly write as:

$$\pi_X M_{XX} + \pi_{Y_1} M_{Y_1 X} + \pi_{Y_2} M_{Y_2 X} + \pi_{Y_3} M_{Y_3 X} = \pi_X \tag{3.14}$$

where Y_1, Y_2, Y_3 are the three nucleotides not equal to X. In words, at the stationary point the frequency π_X of any nucleotide represents an equilibrium between its tendency to experience an error (the outflow $\pi_X(M_{XX} - 1)$) and the inflow of nucleotides mutating into X (the three other terms). The determination of the composition of the genome π by the error rates does not necessarily conflict with the notion that evolution for heat tolerance drove the genome composition to a high value of $\pi_G + \pi_C$, because organisms are

able to change the rates of error adaptively. For example, by making the nucleotides U and A slightly more error prone, an organism can decrease its $\pi_U + \pi_A$ and thus increase the $\pi_G + \pi_C$ of its genome.

Beside these four equations, we already have 12 relations (Eqns. 3.9) linking the rate of each substitution $X \to Y$ ($\pi_X M_{XY}$) with the tolerance T_{XY} to that substitution:

$$T_{XY} = F(\pi_X M_{XY})$$

These can be rewritten in a more convenient form

$$\pi_X M_{XY} = F^{-1}(T_{XY}) \tag{3.15}$$

(the inverse exists if we restrict F to be strictly increasing). Together with the normalization conditions $\sum \pi_X = 1$ and $\sum \pi_X M_{XY} = \epsilon$ we seem to have enough equations to obtain both π, M simultaneously (or so it seems). Unfortunately, because the T_{XY} is nearly symmetric (i.e. $T_{XY} \approx T_{YX}$), no progress can be made, as the following theorem shows:

Theorem 3.1 *If T_{XY} is symmetric, then for any choice of state vector π, one can find a stochastic matrix M such that the following equations hold:*

$$\pi_X M_{XY} = F^{-1}(T_{XY})$$
$$M\pi = \pi$$

Proof:
Suppose $\pi_X M_{XY} = F^{-1}(T_{XY})$ holds for any $X, Y (Y \neq X)$ and that $T_{XY} = T_{YX}$. Thus, by Eqn. 3.15 we have $\pi_X M_{XY_i} = \pi_{Y_i} M_{Y_i X}$ for $i = 1, 2, 3$.

Let us show that for all X the stationarity condition (Eqn. 3.14) holds. Recall that by Eqn. 3.4, have $M_{XX} = 1 - (M_{XY_1} + M_{XY_2} + M_{XY_3})$. Consequently,

$$\begin{aligned}
\pi_X M_{XX} \quad + \quad & \pi_{Y_1} M_{Y_1 X} + \pi_{Y_2} M_{Y_2 X} + \pi_{Y_3} M_{Y_3 X} \\
= \quad & \pi_X \left(1 - (M_{XY_1} + M_{XY_2} + M_{XY_3})\right) + \pi_{Y_1} M_{XY_1} + \pi_{Y_2} M_{XY_2} + \pi_{Y_3} M_{XY_3} \\
= \quad & \pi_X + (-\pi_X M_{XY_1} + \pi_{Y_1} M_{XY_1}) + (-\pi_X M_{XY_2} + \pi_{Y_2} M_{XY_2}) + (-\pi_X M_{XY_3} + \pi_{Y_3} M_{XY_3}) \\
= \quad & \pi_X
\end{aligned}$$

The implications for this method are dire even though T_{XY} is not perfectly symmetric, since numerical attempts to find π, M would inevitably suffer from instability. This symmetry is an inevitable consequence of the way the genetic code evolves to reduce errors, as follows. Suppose evolutionary pressure due to the error $X \to Y$ causes reassignments in the genetic code. In the modified code, when the error takes codon C_1 into codon C_2, the chemical distance between their amino acids, $EX(A_1, A_2)$, is reduced. However, this process inadvertently reduces the distance $EX(A_2, A_1)$, thus improving the error reduction for the error $Y \to X$ often beyond what is justified by the evolutionary force this error induced. The more frequent of the two errors can be assumed to have had the most influence in shaping the genetic code, thus "shadowing" the lesser error, and the data likely reflects this more influential force. Shadowing is a fundamental barrier to any method discussed in this work. Fortunately, while shadowing is catastrophic for the current method, other methods partially overcome it.

3.3 The Tolerance-Safety Method

This Tolerance-Safety Method is a relational reconstruction method, with the distinctions of being highly successful and resolving many of the limitations of its predecessors.[3] It builds upon the idea of other relational methods, more specifically, the notion of tolerance T_{XY} for a given error. However, it also introduces a new set of equations that allow one to calculate π. Thus, we are finally able to address the question of temperature. Moreover, the values of M and π are calculated simultaneously without making any simplifying assumptions about either one.

Recall that while discussing the structure of the genetic code, we noted that the code is expected to evolve so that its structure would reflect the error rates: the code would adapt more to the relatively frequent errors and reduce the damage they cause. In fact, we made this relation explicit by linking the tolerance, expressed as the degeneracy with the error

[3] I am currently submitting a paper that introduces the reconstruction idea and applies the tolerance-safety method to obtain the G+C values [34].

rate. Here we would like to develop this idea to include non-synonymous substitutions and develop a related concept for nucleotides rather than errors, a concept we call "safety".

3.3.1 Measuring Error Reduction

The "tolerance" T_{XY} we saw before and the "safety" σ_X we shall see shortly are both measures of the fitness of the genetic code. Indeed, there is considerable literature on measuring the error-reducing fitness of a genetic code, see e.g. Gilis *et al.* [31] but it always considers its *overall* fitness. Here, we instead wish to quantify in more detail the fitness of the code with respect to *individual* types of error and all errors in a given nucleotide.

Errors and Tolerance

When we discussed T_{XY} we considered it an extension of the degeneracy counting method. Thus, we ignored the fact that the nucleotides have different probabilities, and errors have a codon position bias - factors that we would like to include now. Recall that the number of occurrences of each nucleotide X in the genetic code is $64 \cdot 3/4 = 48$. We label these occurrences by $k = 1, .., 48$ and the corresponding codons by C_k (a codon that contains multiple X's is labeled multiple times). The tolerance value, T_{XY}, is defined as the mean change in chemical properties of the amino acids (the EX value), conditional on the error $X \to Y$ occurring. Thus,

$$T_{XY} = \frac{1}{\text{Probability of } X \to Y} \cdot \text{Mean EX} \tag{3.16}$$

$$= \frac{1}{\pi_X \cdot M_{XY}} \cdot \sum_{k=1}^{48} r(k) \cdot M_{XY} \cdot p_{t(k)} \cdot EX\left[A(C_k), A(\widehat{C_k})\right] \tag{3.17}$$

Eqn. 3.17 contains a sum over all k in order to take into account that nucleotide X can be in any one of the 48 codons C_k. As before, $EX\left[A(C_k), A(\widehat{C_k})\right]$ measures how efficient a resulting protein is, on average, if a substitution error replaces a codon, C_k, by another codon, $\widehat{C_k}$, i.e., it measures the chemical proximity of the amino acids $A(C_k)$ and $A(\widehat{C_k})$. By definition, $EX\left[A(C_k), A(\widehat{C_k})\right] = 0$ when the amino acids are most dissimilar and the value is 1 if the amino acids are identical, i.e., if the codons C_k and $\widehat{C_k}$ are synonymous. Note

that due to the large number of terms in Eqn. 3.17, our results will be only very weakly dependent on the details of the amino acid proximity measure. Also, we set $EX = 1$ for all errors between two stop codons and we set $EX = 0$ whenever an amino acid is replaced by a stop codon or vice versa. The latter may underestimate the effect of those errors on proteins but the resulting inaccuracy in T_{XY} is small because, again, each T_{XY} consists of a large number of terms.

The r_k term in Eqn. 3.17 is the probability r_k of each codon C_k in which the error $X \to Y$ could occur. Suppose in codon C_k the remaining two nucleotides beside X are $a(k)$ and $b(k)$. Assuming that on average the probability of a codon is the product of the probabilities of its nucleotides, we get that $r(k) = \pi_X \pi_{a(k)} \pi_{b(k)}$.

Finally, Eqn. 3.17 unlike the earlier methods, contains weight factors, $p_{t(k)}$, which take into account the fact that the probability of a substitution may also depend on the position of X in the codon. Namely, let $t(k) = 1, 2, 3$ denote the position of X in the codon C_k. If a substitution possesses unequal probabilities, p_t, of occurring at codon positions $t = 1, 2, 3$, then the contribution of a substitution $X \to Y$ to T_{XY} needs to be weighted by the probability, $p_{t(k)}$, for its position, t, in the codon. Since we saw in chapter 1 that the genetic code is better adapted to a distribution that is skewed towards first and third position errors, we take $(p_1, p_2, p_3) = (\frac{5}{16}, \frac{1}{16}, \frac{10}{16})$.

The T_{XY} given by Eqn. 3.17 is a generalization of the tolerance value we used earlier. It may be verified that the old value is recovered if, first, the probabilities of the nucleotides are all $\pi_X = 25\%$ and second, the errors occur evenly in all three codon positions ($p_t = \frac{1}{3}$). The T_{XY} given by Eqn. 3.17 can be simplified by canceling the $\pi_X M_{XY}$ from the numerator and the denominator, giving a polynomial in π and M containing exactly 48 terms. I wrote a Matlab program that took the structure of the genetic code as input and produced the formula for T_{XY} as output (see Appendix C).

In past methods, we introduced a linear function to relate the error rate $\pi_X M_{XY}$ and the tolerance T_{XY}. However, a more careful consideration suggests that the relation is at the very least affine. The trouble with a linear relation is that it incorrectly describes the behavior as $\pi_X M_{XY} \to 0$. In actuality, even if the rate of error was very small, T_{XY} does not go to zero because average$EX \geq \min_{i,j} EX(A_i, A_j) > 0$. Thus, a more correct relation

is

$$T_{XY} - k_0 = c\pi_X M_{XY}$$

where c, k_0 are constants with k_0 representing the tolerance level corresponding to no evolutionary pressure ($\pi_X M_{XY} = 0$). Yet, if we used an affine relation, there would be no good way of estimating k_0. A better way to proceed is to use difference relations, as follows. Consider any two substitutions $X \to Y$ and $I \to J$, with absolute probabilities $\pi_X M_{XY}$ and $\pi_I M_{IJ}$, respectively. For instance, if $X \to Y$ is the more frequent substitution, i.e., if $\pi_X M_{XY} > \pi_I M_{IJ}$, then the genetic code should have preferentially adapted to the substitutions $X \to Y$, i.e., the tolerance values should obey $T_{XY} > T_{IJ}$. We now postulate that differences in the error rates led to the evolution of proportional differences in the tolerances:

$$(T_{XY} - T_{IJ}) = F_m \cdot (\pi_X M_{XY} - \pi_I M_{IJ}) \tag{3.18}$$

This is a linear approximation and as such will be valid as long as the differences among the tolerance values are small. In Eqn. 3.18, the right hand side quantifies the evolutionary pressure, while the left hand side quantifies the error-reducing adaptation of the genetic code. The proportionality parameter, F_m, expresses the strength of the coupling between the evolutionary pressure due to errors and the adaptive response of the genetic code. For example, a large F_m would mean that the coupling is strong, namely that the code's adaptation to the various types of errors depends very sensitively on the relative prevalence of these errors.

Nucleotides and Safety

Having derived a better way of calculating the error rates, we return to the problem of finding the nucleotide frequencies. The basic observation is that varying frequencies should have exerted correspondingly varying evolutionary pressures. To see this, assume, for instance, that a nucleotide X was more abundant in the genome than a nucleotide Y. In this case, even if X and Y were equally error-prone, there would have been more evolutionary pressure for the genetic code to evolve so as to tolerate errors in X than pressure for the code to evolve to tolerate errors occurring in Y, because the absolute probability of error is higher for X. The outcome of the evolution of the genetic code

should, therefore, have been such that when a nucleotide X is substituted by some other nucleotide then the resulting amino acid is, on average more similar to the original, than what would be observed if Y was substituted. To measure this effect, it is necessary to introduce a new measure of fitness of the genetic code. This measure, called "safety", σ_X of nucleotide X in the genetic code is derived from T_{XY}, but takes into account the three possible substitution of that nucleotide, rather than a particular one type. The safety value is found by taking the weighted average of the tolerance values for the three possible substitutions of X. The weights are given by the probabilities of each substitution, i.e.,

$$\sigma_X = \frac{M_{XY_1}}{N_X} T_{XY_1} + \frac{M_{XY_2}}{N_X} T_{XY_2} + \frac{M_{XY_3}}{N_X} T_{XY_3} \tag{3.19}$$

where Y_1, Y_2, Y_3 are the other three nucleotides and $N_X = M_{XY_1} + M_{XY_2} + M_{XY_3}$ is the normalization constant. The normalization ensures that the new measure σ_X is sensitive only to differences in π_X, not the varying error proneness of nucleotides. Biologically, the safety, σ_X is the conditional mean chemical proximity of pairs of amino acids that are created when X experiences any error.

It is straightforward to relate the π values to the safety values σ. If $\pi_X > \pi_Y$ was true for primordial organisms, error reduction in the genetic code would have ensured that $\sigma_X > \sigma_Y$. This leads us to postulate that:

$$(\sigma_X - \sigma_Y) = F_\pi \cdot (\pi_X - \pi_Y) \tag{3.20}$$

where F_π is a coupling constant. The precise relationship between the two differences may be nonlinear but this linear approximation should be very good since the values of $\pi_X - \pi_Y$ are likely to have been small.

We now finalize the setup of this method. Recall that π_X and M_{XY} satisfy two further equations, namely the probabilistic consistency condition $\sum \pi_X = 1$ and the definition of an overall error rate $\epsilon = \sum_{Y \neq X} \pi_X M_{XY}$. We set $\epsilon = 1\%$ but the precise value of ϵ is immaterial because multiplying it by a constant c causes no change except multiplying all the M_{XY} values by c. We show in subsection 3.3.3 that the values of F_π and F_m can be chosen in a wide range around the best estimate without significantly affecting the conclusions.

We also note that among the $12 \cdot 11$ equations of the form Eqn. 3.18 generically 11 are linearly independent. Similarly, of the $4 \cdot 3$ equations of the form Eqn. 3.20, generically

3 are independent. Therefore, Eqns. 3.18 and 3.20, together with $\sum \pi_X = 1$ and $\epsilon = \sum_{Y \neq X} \pi_X M_{XY}$ are solvable for the 16 variables π and M after we estimate the two values of F_m and F_π. To this end, we note that for typical values of π_X and M_{XY} we obtain values of $T_{XY} - T_{IJ} \leq 250$ and $\sigma_X - \sigma_Y \leq 200$. On average, we crudely expect $\pi_X M_{XY} = \epsilon/12$ and $\pi_X = 25\%$, which indicates that corresponding to the two ranges above are $\pi_X M_{XY} - \pi_I M_{IJ} \approx \epsilon/24$ and $\pi_X - \pi_Y \approx 12.5\%$. Therefore, for the computation (using AMPL [22]) we set $F_m = 250/(\epsilon/24)$ and $F_\pi = 200/0.125$. As will be shown in the following section, our results are robust against variations in the estimated values of F_m and F_π.

All of the computations were performed using a combination of Matlab and AMPL. The former, a well-known mathematical programming environment, was used to generate the cumbersome values of T_{XY} and σ_X based on the genetic code. The latter was used as a non-linear solver. AMPL is a programming language that was developed for solving large systems of equations and optimization problems. To solve a problem in AMPL, the programmer typically writes two files: first, a "model" file that describes the functions and variables and the relations between them, and second, a script (a batch file) that calls one of the "solvers" included with AMPL and processes the data. The purpose of the script in this method was to catch numerical problems and to produce a file for processing in Excel (see Appendix C).

3.3.2 Results

The results are displayed in Fig. 3.3. In particular, Fig. 3.3A shows the reconstructed values of the primordial nucleotide frequencies. We find that $\pi_G + \pi_C = 50.1\%$, i.e., the $G + C$ content is not elevated. Such a moderate level of $G + C$ is inconsistent with heat-adapted RNA, as we saw in Fig. 1.2. Such a result has intriguing implications to our understanding of primordial evolution, and we will discuss them in the concluding chapter.

The finding that $\pi_G + \pi_C \approx 50\%$ is fundamentally caused by elevated safety to errors in A, compared to safety to errors in G, *i.e.* $\sigma_A > \sigma_G$, and with a different genetic code the results would have been different. Indeed, I have verified that the method is able to infer $\pi_G + \pi_C$ much greater than 50% given alternative genetic codes. Namely, I constructed codes in which $\sigma_A < \sigma_G$ while most features of the genetic code, including the third position degeneracy structure that provides tolerance to transition errors, were

retained. As required, given the elevated safety for G, the method inferred an elevated $\pi_G + \pi_C$. Moreover, while in the standard code $\pi_G + \pi_C$ stays close to 50% regardless of the choice of F_π and F_m, with those codes it can reach 59% or more. It is likely that the algorithm would infer an even higher $\pi_G + \pi_C$ value if the constructed codes were allowed to have fewer of those $U - C$ and $A - G$ degeneracies in the third position because they force $\sigma_U \approx \sigma_C$ and $\sigma_A \approx \sigma_G$. Likely, had the standard code genuinely evolved in response to a genome rich in $G + C$, these degeneracies would have been much rarer.

Also, for the standard code, we find that the rates of transition errors were consistently high compared to the transversions (Fig. 3.3B), in line with earlier reconstruction methods as well as nucleotide substitution rates of modern organisms (cf. Ota and Penny [61]). We note that like in other methods, because of the way the genetic code minimizes errors, we are prevented from clearly reconstructing the differences between the rates of forward and backward substitutions. As a result, the reconstructed values $\pi_X M_{XY}$ and $\pi_Y M_{YX}$ tend to be close even if these rates were not as close when the genetic code was fixed. A detailed analysis of this shadowing effect and of other sources of uncertainty is described in the subsection 3.3.3 below. It is shown there that even under extreme assumptions our value for $\pi_G + \pi_C$ increases to merely about 57%.

The foregoing method may appear to have a suspicious mathematical property. Namely, it appears to obtain the values of π, M continuously from just the values of T. Even though T is approximately continuous, it lies in a space which has fewer dimensions than the space of π, M values. More precisely, while the former are 20 variables with 6 constraints (the four equations for determining the diagonal elements of M, the normalization of π and the rate ϵ), the latter are 12 evolutionarily-determined values of tolerance, T_{XY} (the safety σ_X are found from them). Hence, the method appears to take the set T_{XY} - a point in $\Re^{12}$ - and yields a unique point in $\Re^k$ where $k = 20 - 6 = 14$. Standard topological arguments suggest that such a mapping cannot be continuous and bijective. In actuality, because of the two parameters F_m and F_π, whose values are a point in $\Re^2$, the reconstruction takes a point in $\Re^{14}$ and not from $\Re^{12}$. Thus, the mapping can be continuous and bijective. The two parameters, like the 12 T_{XY} values, are determined by evolution and represent how effective was evolution in achieving error reduction. What is unusual about them is that they need to be estimated rather than determined from the genetic code (which is why we

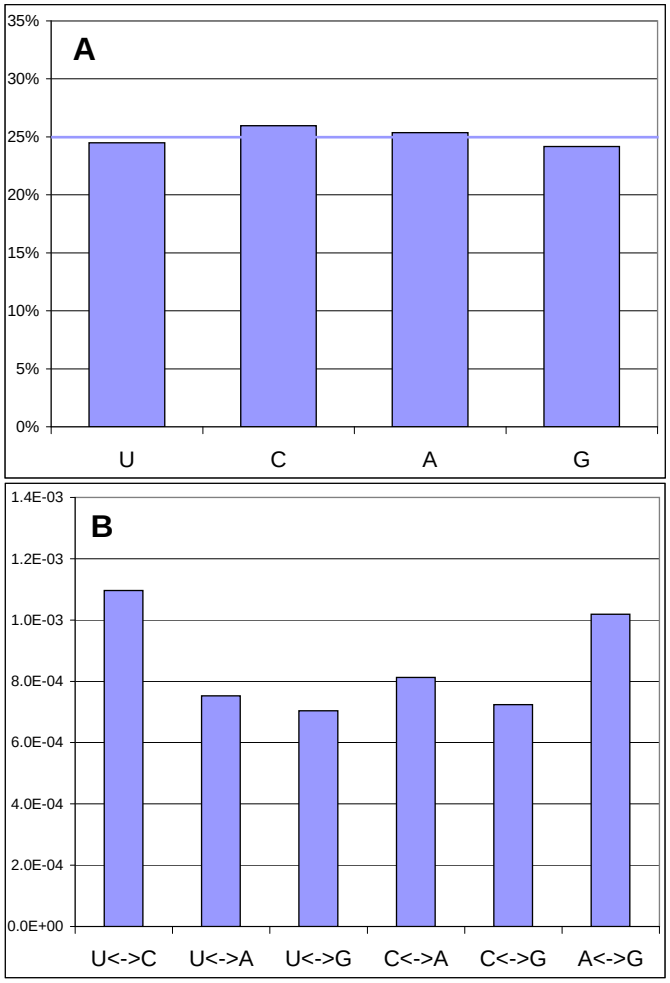

Figure 3.3: **Tolerance-Safety Method: Reconstructed π and $\pi_X M_{XY}$**
(A) shows the reconstructed nucleotide frequencies while (B) gives the reconstructed absolute probabilities of errors at the time the genetic code stopped evolving. The rates of forward and backward substitutions are found to be similar and each column in (B) displays the larger of the two rates, as discussed in the text. The scale of the bars in (B) reflects an assumed overall error rate of $\epsilon = 1\%$. The value of ϵ has no effect on the nucleotide frequencies.

call them parameters). Fortunately, in subsection 3.3.3 below we show that because the two parameters have a special geometric property, our conclusions are insensitive to any error in such an estimate.

3.3.3 Robustness Analysis

In the following we show that our results are robust in various respects. In points 1-3 of this robustness analysis, F_m and F_π are set to the estimated values described in the Methods section. In point 4 we address the robustness of the results with respect to changes in F_m and F_π.

1) Regarding the "shadowing" effect, we expect that strong shadowing of the rates of transitions would have the largest impact on our data because transitions make the largest contributions to the calculation of the safety values, σ_X. In particular, if the rates of the transitions $U \rightarrow C$ and $A \rightarrow G$ are shadowed, then the value of $\pi_U + \pi_A$ would be systematically elevated, and the value $\pi_G + \pi_C$ would be systematically suppressed. This is unlikely to have been the case because σ_G is found to be quite low (ultimately because amino acids where G is in the second position have a relatively low degeneracy) while σ_U is quite high (because of a relatively high degeneracy). We can put an upper bound on the impact of shadowing on the nucleotide frequencies by considering the case of a larger than possible shadowing. To this end, we set the values of T_{UC} and T_{AG} at 50% of normal. We found that even in this extreme case $\pi_G + \pi_C$ increases to merely 56.7%. Such a small elevation of $\pi_G + \pi_C$ still indicates a non-thermophile organism (Fig. 1.2).

2) Eqns. 3.18 and 3.20 are linear approximations to what are likely non-linear equations $\pi_X M_{XY} - \pi_I M_{IJ} = f(T_{XY} - T_{IJ})$ and $\pi_X - \pi_Y = g(\sigma_X - \sigma_Y)$. Fortunately, the linear approximation can be assumed to be very good because the non-linear f and g must be both continuous and monotonously increasing. Specifically, $M_{XY} > \pi_I M_{IJ}$ must give $T_{XY} > T_{IJ}$, and likewise, $\pi_X > \pi_Y$ must imply $\sigma_X > \sigma_Y$. One test for the impact of non-linearity is to replace Eqn. 3.20 with $(\sigma_C - \sigma_Y)^2 = \widehat{F_\pi}(\pi_C - \pi_Y)$ where C is Cytosine and Y stands for any of the three other nucleotides. This choice satisfies the considerations above because after calculating we find that indeed $\sigma_C > \sigma_Y$ and $\pi_C > \pi_Y$ for all Y. Here, $\widehat{F_\pi}$ is the value of the proportionality constant adjusted for the non-linearity ($\widehat{F_\pi} = 200^2/0.125$). Solving this modified equation (simultaneously with the unchanged Eqn. 3.18) leads to less

than 1% change in nucleotide frequencies and less than 0.2% change in $\pi_G + \pi_C$.

This result is part of a more general phenomenon that can be stated as follows: The ordering of the set of nucleotide frequencies $\{\pi_U, \pi_C, \pi_A, \pi_G\}$ does not depend on the full nonlinear generalization of Eqns. 3.18 and 3.20 as long as the functions f and g are slowly-varying and increasing. This means that not only the solutions to the linear approximations but also the solutions to the full equations imply that π_G was the least abundant nucleotide and therefore that $\pi_G + \pi_C$ was not significantly elevated above 50%. Note that Eqns. 3.18 and 3.20 do in fact already incorporate some non-linearity in the sense that $r(k)$ in T_{XY} depends non-linearly on the nucleotide frequencies, while σ_X also depends non-linearly on M_{XY}.

3) The effect of the position bias of errors on our results is very small. Recall that we model this bias by the triplet p_t as described earlier. When we choose p_t to have no position bias instead of the non-uniform bias, we find a change of at most 1% in the nucleotide frequencies. This insensitivity is plausible: a frequently-occurring nucleotide would have caused the genetic code to structure in such a way that errors in this nucleotide are less harmful no matter in which position in a codon the error occurs. There is, though, a stronger dependence on the position bias in the reconstructed error rates. In particular, one finds a greater gap between the rates of transitions and the rates of transversions in the biased case. This difference is especially noticeable when F_m is small. The effect is a natural consequence of the structure of the genetic code: the code is particularly adapted to transition errors in the third position, as we saw in Fig. 1.13.

4) The dependence of our results on the values of F_π and F_m is plotted in Fig. 3.4. Notice that over the entire range of possible values of the parameters F_π and F_m the sum $\pi_G + \pi_C$ ranges merely between 50% and 52%. Thus, our conclusion that the $G + C$ content is not significantly elevated is robust with respect to varying the values of F_π and F_m. Regarding the determination of the values of F_π and F_m, we notice on the one hand that, for large values of F_π and F_m, the curves in Fig. 3.4 converge. This is because if in Eqn. 3.20, for instance, $\sigma_X - \sigma_Y$ is known and F_π is doubled, then the difference $\pi_X - \pi_Y$ is halved. The value of the parameter F_m affects the solution in a similar way. Regardless of the value of F_m, transitions are always more frequent than transversions. This is nicely consistent with the fact that the difference between the rates of these errors

stems from chemical differences between pyrimidines and purines [86] - a difference that should have impacted the primordial genetic machinery in essentially the same way that it affects modern organisms. Exactly how different the two types of rates are depends of course on the value of F_m. For instance, when $F_m = 250/(\epsilon/10)$ we find that the ratio between the rates of transitions and transversions is between 3 and 4 (cf. Fig. 1.9).

On the other hand, the parameters F_π and F_m also cannot be arbitrarily small or else the equations have no solutions. For instance, a very small value of F_π for a fixed $\sigma_X - \sigma_Y$ would require the difference $\pi_X - \pi_Y$ to be so large that the constraint that the sum of probabilities is one ($\sum \pi_X = 1$) no longer holds. F_m cannot be very small for similar reasons, as this would be inconsistent with the overall error rate being equal to ϵ. An estimate of the lower bound on F_m follows from the size of the gap between the rates of transitions and the rates of transversions. With the exception of mitochondria, the average rate of transitions in modern organisms is rarely more than four times the average rate of transversions [51, 63] with similar expectations for errors in primordial organisms [86, 77]. This suggests the lower bound is $F_m \geq 250/(\epsilon/10) = 2.5 \cdot 10^5$ which is shown in Fig. 3.4B.

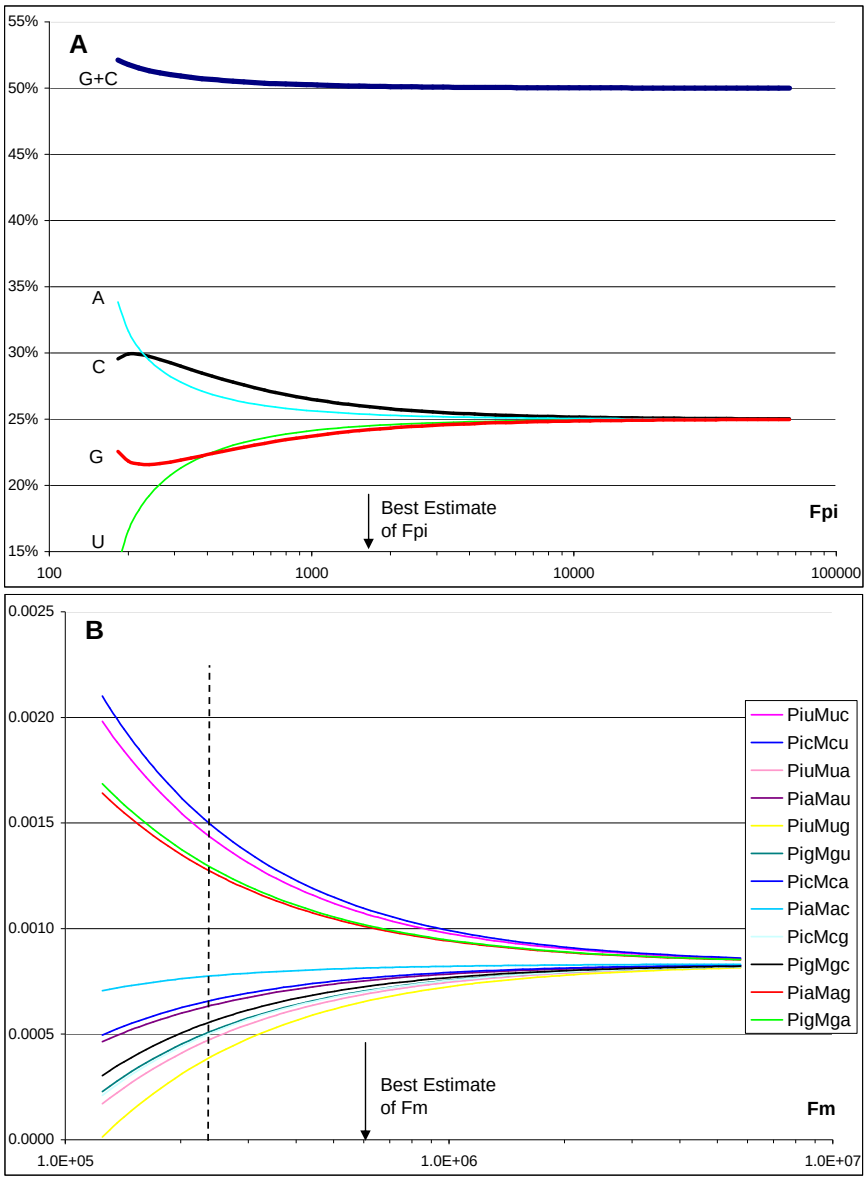

Figure 3.4: **Tolerance-Safety Method: Effect of the parameters F_π and F_m**
The reconstructed nucleotide frequencies, (A), and absolute probabilities of errors, (B), are shown as functions of F_π and F_m, respectively, down to the minimum possible values of F_π and F_m. (A) shows that the reconstructed value of $\pi_G + \pi_C$ in the primordial genome is very close to 50% regardless of the value of F_π. Thus, our finding that $\pi_G + \pi_C$ was not elevated is robust. In (B), the rates of transition errors are the top four curves while the rates of transversion errors are the bottom eight curves. The vertical dashed line indicates estimated lower bound on F_m. Away from its extreme low, F_m has a negligible effect on π_X values

3.4 Optimization-based Reconstruction Methods

There is an entirely different way of approaching the reconstruction problem, as follows. Recall, one of the goals of communication engineering is to maximize the transmission rate and to bring it as close as possible to the channel capacity. Because many error-prone genetic processes are of an information-theoretic nature, it is likely that natural selection had a similar engineering goal when it designed the genetic code. In other words, the structure of the genetic code may represent an optimal solution to the problem of optimizing a certain information-theoretic function. Even if the genetic code is not the optimal solution, it could represent a local optimum in the fitness landscape of genetic codes. More generally, the fitness function may also include biological criteria, rather than just information-theoretic ones.

In this framework, evolution is a search after a genetic code g that lies close to the optimal value of a fitness function $F(g, \pi, M)$. The value of the function F and hence of the optimal g is dependent on the composition of the genome π and of the rates of error acting on the genome M. Of course, our goal here is not to find the optimal g which we know, but rather to determine the π and M given g.

In the subsections below I will introduce some of the methods that apply this idea:

1. The Transmission Function Method
2. The Fixed Risk Method
3. Other Methods

3.4.1 The Transmission Function Method

The error reduction hypothesis discussed in previous chapters means that the error-free transmission of information is an important evolutionary criterion. Perhaps the most natural way of quantifying this is by assuming that evolution of the genetic code optimized T - the information theoretic transmission function.[4] In other words, the genetic code

[4]The transmission function has already been found useful in solving problems involving estimation and data analysis [85], which are somewhat similar to our reconstruction problem.

evolved so as to increase the rate of transmission through the genetic channel. Recall that such a channel takes an amino acid as input and produces an amino acid as output. Thus, if during transmission the codon is changed but the symbol remains the same, no actual error occurs in the biological sense. Increasing the rate of transmission requires evolution of a genetic code that minimizes the probability of one of the 21 symbols changing into another. An error-minimizing genetic code would reflect factors like whether a given nucleotide is frequent and whether it is prone to errors, and what are their rates. In this light, many features of the genetic code are seen as optimizations that maximize the transmission rate. For instance, there is no better way to reduce the number of interchanges in the amino acids (the equivocation) than to ensure that the most frequent errors - transition errors in the third position are silent, and this is nearly always the case.

Optimization of the genetic code was likely a process that was highly dependent on the nucleotide composition π and the error rates M. If the π, M were changed significantly, a different code would have been optimal. This suggests that these exist a close relationship between π, M and the code g. Because that relationship occurs at the optimum of T, we propose to find π, M by optimizing $T(g, \pi, M)$ at a fixed g. This is the reverse of the evolutionary process in which π, M were held fix while g evolved to maximize T.

According to subsection 2.2.1, the transmission T is:

$$\begin{aligned} T &= I(X:Y) \\ &= H(X) - H(X|Y) \\ &= -\sum p(x)\log p(x) + \sum p(x,y)\log\frac{p(x,y)}{p(y)} \end{aligned}$$

where all of the probabilities reflect, in part, the structure of the genetic code. As well, there is a close relation between T and entropy $H(X)$: when the equivocation goes to zero, $T = H(X)$.

To write down T, we need to express various probabilities in terms of our model. These are first, $p(x)$ - the probability of finding amino acid x as the input, second, $p(x, y)$ - the probability of x being the input and y being the output and finally $p(y)$ - the probability of amino acid y as the output. I have written a Matlab program that finds the symbolic expression of all of those quantities (Appendix C). The expressions are often quite long,

but here is one simple case:

$$\begin{aligned} p(x = Histidine) &= \pi_C\pi_A\pi_U + \pi_C\pi_A\pi_C \\ p(x = Histidine, y = Glutamine) &= \pi_C\pi_A\pi_U p_3 M_{CC} M_{AA} M_{UA} + \pi_C\pi_A\pi_U p_3 M_{CC} M_{AA} M_{UG} \\ &\quad + \pi_C\pi_A\pi_C p_3 M_{CC} M_{AA} M_{CA} + \pi_C\pi_A\pi_C p_3 M_{CC} M_{AA} M_{CG} \end{aligned}$$

The $p(y)$ terms are too long to be listed here. Each $p(y)$ is a sum of: 1. Terms expressing error-free transmission of any codon of y, and 2. Terms expressing errors (silent or of Hamming distance one) that lead to a codon of y (higher order errors are negligible). It is necessary to use software like Matlab to write down the expression of the transmission function.

With the transmission function ready, we meed a numerical solver that can find the optimal value of π and M. Although Matlab has some functionality for optimization (the Optimization Toolbox), in my experience another tool is needed for solving harder problems. Instead, it is better to use a less familiar but much more powerful AMPL programming language - the same language as the one we used for solving the non-linear equations of the other methods. Beside writing an optimization program in AMPL, I also wrote a script for catching numerical problems and producing a report. To ensure that the computed solution is indeed the global optimum, the script would recompute it about a hundred times, each time with a different starting point.

In summary, here is the optimization problem we will be solving:

$$Maximize \quad T(\pi, M) \quad where \tag{3.21a}$$

$$\pi, M \quad \geq \quad 0 \tag{3.21b}$$

$$\text{Subject to} \tag{3.21c}$$

$$\forall Y, M_{YY} \quad = \quad 1 - \sum_{X \neq Y} M_{YX} \tag{3.21d}$$

$$\sum \pi_X \quad = \quad 1 \tag{3.21e}$$

$$\sum_{X \neq Y} \pi_X M_{YX} \quad = \quad \epsilon = 0.01 \tag{3.21f}$$

$$p_t \quad = \quad \left(\frac{5}{16}, \frac{1}{16}, \frac{10}{16}\right) \tag{3.21g}$$

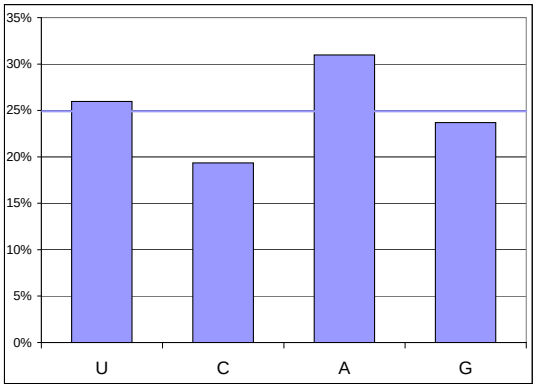

Figure 3.5: **Transmission Function Method: Reconstructed π**

Here the parameter ϵ was set to 1%. The solution does not significantly depend on ϵ unless the parameter is increased by an order of magnitude, but fortunately $\epsilon = 1\%$ is already on the limit of the error rate any species could sustain.

Results

There is no unique solution to the mathematical problem. Instead, for every starting point, the optimization routine finds a locally optimal solution which is different from the solution obtained when starting at a different point. Some regularities do exist, however. For $\epsilon \leq 0.01$, the optimal genome composition is $(\pi_U, \pi_C, \pi_A, \pi_G) = (0.26, 0.19, 0.31, 0.24) \pm (0.01, 0.01, 0.01, 0.01)$ which gives $T(\pi, M) = 4.20$ (Fig. 3.4.1). For larger ϵ, we find optimal solutions that are increasingly dispersed in the set of possible π. As to the error matrix M, we again find no unique solution. Moreover, in all of the solutions, all of the M_{XY} terms are found equal to 0, except for a single term. This special M_{IJ} is always one of the four translation errors, but it is a different one every time! Thus, $\epsilon \approx \pi_I M_{IJ}$. The solution is robust with respect to changes in the position bias of error, p_t. When we change it to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, the values of π, M see less than 5% relative change.

The value of π gives a $\pi_G + \pi_C$ of just 43%, which is lower than that found with the Tolerance-Transmission method. Overall, the two methods differ by up to 7% in their π values. I think that this difference is caused by a tendency of the transmission function method to find the π value that makes all amino acids have the same probability, as

much as possible. Recall that the transmission function can be written as the difference between the input entropy $H(X)$ and the equivocation $H(Y|X)$. Hence, when ϵ is small, $H(X) \gg H(Y|X)$ and maximum transmission coincides with maximum input entropy. Maximum entropy (of the amino acids) has a complex relation with π: when all π_X values are approximately 25%, the probability of an amino acid is determined by its degeneracies - the number of codons that code for it. Because the degeneracies vary widely (from 1 to 6), a uniform π_X gives a very non-uniform distribution of amino acids. In fact, it is not possible to find π that gives the maximal entropy because it is impossible to find a π that gives the uniform distribution. The best possible solution involves increasing the probability π_X for those nucleotides that code for amino acids with comparatively low degeneracy because these are the amino acids that tend to have a relatively small frequency.

This interpretation of the π value is supported by the fact that the current method gives π which is within 1% of the π that maximizes entropy. Also, the reason why in Fig. 3.4.1 π_A is elevated but π_C is suppressed is that the former codes for amino acids with comparatively low degeneracy, while the latter is the opposite. There are some reasons to believe that in the actual genome, π was not specifically set so as to maximize entropy and give a uniform distribution of amino acids. Indeed, it seems like amino acids are actually needed in different quantities, that could correlate with degeneracy. For example, the the single-codon amino acids Tryptophan and Methionine are both difficult to synthesize and are required in special structures only. Empirically too, we find that amino acids usage tends to vary widely (see e.g. Gilis [31]), at least in modern organisms.

In regards to the disappointing result for M, it is likely due to the fact that transmission is not sensitive to differences in the error-reducing capability of the code for the four different transition errors, although it is able to detect the difference between transitions and transversions. The large number of different optimal solutions indicates that the problem is not sufficiently determined. This may have been anticipated by noting that the problem of finding π, M has many more variables than the standard problem of subsection 2.2.1 of finding π that maximizes the transmission rate. Whereas in the standard problem, we are given a fixed M that characterizes the channel and the task is to find π, in the current setup both π and M are free.

I have considered several possible ways forward, but have not yet developed a method

that would resolve this issue. Since information is sent through the genetic channel, it is likely that this mode of communication can be viewed as information-theoretic optimization, although its exact nature is unknown. Another interesting question is whether the transmission function could be modified to include some consideration of non-synonymous substitutions. With the transmission function, a non-synonymous error is sometimes incorrectly considered as just noise that adds to the $H(X|Y)$ term, despite the fact that the two amino acids could be closely related. Perhaps one cannot properly describe communication in biological systems without referring to some "semantic aspects" of the amino acids - their chemical relatedness. This issue calls for a reconsideration of the way we define entropies through probabilities $p(x, y)$ alone rather than dimensional or physical quantities. Progress is not going to be easy since this is a fundamental feature of information theory: it is because of this that information can be studied independently of its carrier.

3.4.2 The Fixed Risk Method

According to the error reduction hypothesis, the natural code has a well-developed assignment of codons that evolved towards reducing the impact of the different types of errors. Suppose those errors occur with probabilities $\{\rho_i\}$, and each causes a certain amount of damage on average to the efficiency of proteins. If error i causes the organism damage D_i, then the average damage from these errors is

$$\langle D \rangle = \sum_{i=1}^{N} \rho_i D_i \tag{3.22}$$

Like in other optimization-based methods, we suppose evolution minimized a fitness function, in this case it is the function $\langle D \rangle$. Even if the genetic code is not strictly optimal, optimization-based ideas could be used as an idealization. Fortunately, evidence discussed in Chapter 1 suggests that the code has undergone extensive error-reducing adaptation, and the approximation is likely a good one.

We now proceed to study the problem of designing the genetic code using ideas from economics.[5] Although the genetic code is an information-theoretic device, it has a property

[5]Information theory and economics have many other interesting connections. See e.g. Cover and Thomas [10].

shared by many economic systems - that of scarce resources: Any well-designed error-reducing code represents a compromise between the need to reduce the impact of twelve different types of errors. For instance, it is possible to increase the value of tolerance T_{XY} for any chosen error $X \rightarrow Y$. However, this would require changing the codon assignment and will likely come at the cost of decreasing the T_{IJ} for some other error $I \rightarrow J$. Thus, we now view this adaptation as the problem of allocating resources $\{r_1, r_2, ...r_N\}$. In the adaptation of the genetic code to minimize damage, resources represent targeted rearrangements of the codon assignments in the genetic code. Whenever r_i is increased, D_i is reduced. However, there is a finite amount of resources. Resources spent on one error become generally unavailable for reducing another error:

$$\sum_{i=1}^{N} r_i = 100\% \tag{3.23}$$

This is of course, an idealization. Although it may be valid for most pairs of errors it breaks down in the case of pairs of forward and backward substitution errors ($X \rightarrow Y$ and $Y \rightarrow X$). For these pairs, increasing r_i reduces not only D_i but also D_j - the damage due to another error j ($Y \rightarrow X$).

The precise relation between r_i and D_i must be formulated while considering the law of diminishing returns, as follows. It is initially easy to adapt the code to an error by forming blocks of synonymous codons. Reducing damage further becomes more and more difficult because it requires large rearrangements of the amino acids in the code, and in any case the payoff is lower. Therefore, it is maladaptive to spend all of the resources on just the few most likely errors. Instead, some of the resources should be spent on the rarer errors.

Mathematically, the law of diminishing returns may be modeled by requiring that the smaller D_i becomes, the slower it decreases:

$$\frac{dD_i}{dr_i} = -\phi D_i \tag{3.24}$$

where $\phi > 0$ is a constant that quantifies the fall in the rate of return. We set $D_i(0) = 1$ - the damage is undiminished when $r_i = 0$. The solution for each i is

$$D_i(r_i) = \exp(-\phi r_i) \tag{3.25}$$

Using the method of Lagrange multipliers, one can show that the optimal resource allocation is to set r_i for all i such that $\rho_i = \frac{constant}{\exp(-\phi r_i)} = \frac{constant}{D_i}$ or

$$\rho_i D_i = constant \tag{3.26}$$

The advantage of the model in Eqn. 3.24 over alternative formulations is that the resulting optimality condition Eqn. 3.26 does not contain any r_i - quantities which we cannot easily quantify.

In Engineering and Economics, the product of probability and damage of an event is known as the risk associated with that event. Therefore, our model implies that the genetic code satisfies a "fixed risk relation". Indeed, the fixed risk relation is a type of an inverse relation between frequency and distance discussed in subsection 3.1.1. Relation 3.26 allows us to derive equations linking the error rates and the nucleotide frequencies to this damage, D_i, as follows. Let us associate the set of events $\{i\}$ with the 12 substitution errors. The probability of an error $X \to Y$ is $\pi_X M_{XY}$ and it causes on average damage which we may label D_{XY}. Hence Eqn. 3.26 gives

$$\pi_X M_{XY} D_{XY} = K_m \tag{3.27}$$

K_m is the risk constant (m in K_m refers to the M matrix).

We now propose to measure the damage due to error $X \to Y$ by calculating the average chemical distance, d, between amino acids when such an error occurs. Recall that the tolerance T_{XY} to such a substitution represents the average value of chemical proximity, EX, between amino acids when the error occurs. Since $0 \leq EX \leq 1$, we can use $d = 1 - EX$ for the chemical distance and interpret it as the average lost enzymatic efficiency. Thus, we define D_{XY} by a slight modification of Eqn. 3.17 (of the Tolerance-Safety Method)

$$D_{XY} = \frac{1}{\pi_X M_{XY}} \cdot \sum_{k=1}^{48} r(k) \cdot M_{XY} \cdot p_{t(k)} \cdot \left(1 - EX\left[A(C_k), A(\widehat{C_k})\right]\right) \tag{3.28}$$

Eqns. 3.27 and 3.28 are insufficient to determine the values of π, M. However, we can determine the M matrix if we assume that π has a uniform distribution. We also set other

conditions in analogy with the previous methods:

$$\begin{aligned} \forall Y\, M_{YY} &= 1 - \sum_{X \neq Y} M_{YX} \\ \sum_{X \neq Y} \pi_X M_{XY} &= \epsilon = 0.01 \\ p_t &= \left(\frac{5}{16}, \frac{1}{16}, \frac{10}{16}\right) \end{aligned}$$

Results

The M_{XY} values we find are displayed in (Fig. 3.6). Again, the values closely resemble the results of other methods. If the positions error bias is set to uniform, we obtain a $10-30\%$ fall in the calculated rates of the transition errors and a simultaneous increase of about 15% in the rates of transversions.

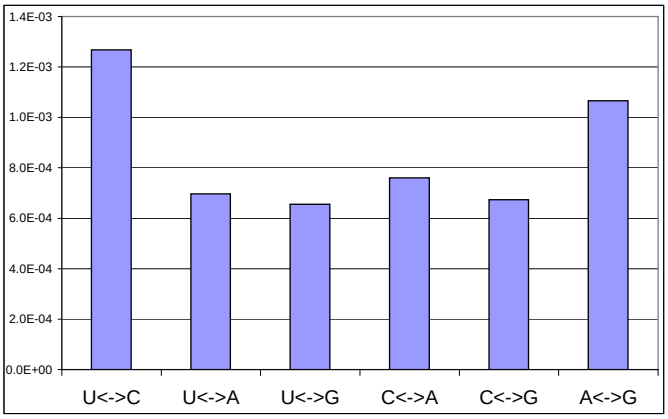

Figure 3.6: **Fixed Risk Method: Reconstructed** M

The values are based on an overall error rate of $\epsilon = 1\%$, but the error rates scale linearly with the value of ϵ.

We did not proceed to investigate this method further because it is not as well justified as the Tolerance-Safety Method. However, one could potentially develop equations like the safety equation, Eqn. 3.20 and determine the nucleotide frequencies.

3.4.3 Other Optimization Methods

I have also investigated alternative reconstruction methods that involve the optimization of a certain function. Firstly, I have tried to identify an alternative fitness function to replace the transmission function, but without success. One idea is to maximize the fidelity function (Eqn. 3.12), that is, to find π and M that maximize the probability of correct transmission. The advantage of this function is that its optimum does not necessarily lie close to the π value that maximizes the input entropy. Unfortunately, it was found that no unique solution exists and that it is insufficiently sensitive to M (results not shown).

Second and more radical was the idea of "parametric optimization", as follows. Hitherto, we attempted to identify the function $F(g, \pi, M)$ that evolution optimized. We then proceeded to find π, M by finding the values of π, M that optimize this very function. In the parametric optimization method, we also identify the fitness function $F(g, \pi, M)$. However, we find the values of π, M such that $F(g)$ is an optimal choice of the genetic code g. Although g is not a continuous variable, the optimal genetic code g^* may have had the property that any small changes in g^* decrease fitness F. Indeed, in Chapter 1 we argued that the genetic code is highly evolved, and is perhaps even locally or globally optimal. We hope to find π, M with this method because the fitness F depends on π, M and therefore, only for some values of π, M the optimality condition would hold. Since g is a parameter of $F(g, \pi, M)$, we call this method the "parametric optimization method".

We implement this method as follows. Small changes in the code are reassignments of one of the 64 codons into a new amino acid. We label each of those $64 \cdot 20$ slightly modified codes $g_{i,j}$, where i labels the codon and j each of its 20 possible reassignments. We then determine the changes of fitness, $\Delta_{i,j} = F(g_{i,j}, \pi, M) - F(g^*, \pi, M)$, due to this reassignment. If the genetic code is optimal for a certain π, M pair, then those $\Delta_{i,j}$ are expected to be negative. If the genetic is merely highly optimized (but not perfect), then $\Delta_{i,j}$ values should be mostly negative (slight sub-optimality is of little concern, as we shall see). We then write a global sum $Z = \sum_{i,j} \Delta_{i,j}$ and find the values of π, M that make Z as large and as negative as possible. An alternative approach to defining Z is to find the minimum of the maximum change, namely to find minimum of $\bar{Z} = \max_{i,j} \Delta_{i,j}$. The advantage of $\bar{Z}$ over Z is that it does not allow a very negative $\Delta_{i,j}$ to compensate for a positive $\Delta_{i',j'}$. Unfortunately, $\bar{Z}$ is unsuitable for numerical computation because it is

not smooth and it was not used. A special problem arises with the single-codon amino acids Tryptophan and Methionine. From the point of error reduction, it is advantageous to remove them from the genetic code, thus increasing the error-reducing degeneracy of their neighbors. This must not be allowed to happen, and these changes $\Delta_{i,j}$ were explicitly excised from Z.

Given that we do not clearly known the fitness function, we are free to make an arbitrary choice and we take the fitness function that makes computation easier. This choice is not insignificant. For instance, were we to choose the transmission function as F, the expressions for $\Delta_{i,j}$ would have become unwieldy. Indeed, I have found that this choice makes just the statement of the Z fitness function about 10 Megabytes long! This is far too long for the number of computations required in an optimization algorithm, even if a high-speed workstation is used. A far better choice for the fitness function is the fidelity function because it consists of a sum. Terms that correspond to amino acids not effected by the reassignment $g_{i,j}$ cancel out when we calculate $\Delta_{i,j}$. The resulting $\Delta_{i,j}$ is only a few terms long, giving a relatively compact expression for Z. Beside this change in the fitness function, in this method we may use the same setup as for the transmission function (Eqns. 3.21).

Despite trials with slightly different setups, the results of this method were disappointing, and no unique pair of π, M was found. There could be at least four causes of this problem: the fidelity function is unsuitable for optimization, the optimization landscape is too "rugged" which prevents the algorithm from finding the optimum, or finally, that there is a basic but unidentified flaw in this parametric optimization technique. Nevertheless, I think that the technique is conceptually sound, and may bear fruit in the future.

Chapter 4

Conclusions

The different methods I have developed, and in particular the Tolerance-Safety Method have successfully reconstructed some details about the primordial conditions from the genetic code. They reveal a primordial genetic machinery that is surprisingly modern both in the error rates and in the nucleotide frequencies. Like in modern organisms, the rates of transition errors are found to be elevated above the rate of the transversions. Unfortunately, the detailed pattern of errors may be too noisy especially due to the shadowing effect, to be studied further. More significant than the error rate data, is the finding that there was no extreme $G + C$ nucleotide content in the primordial genome. It is clear from Fig. 4.1 below that the $G + C$ value of 50% as found by the Tolerance-Safety method (Fig. 3.3A) is inconsistent with thermophile organisms having RNA genomes, because we expect to see in such organisms $G + C$ of 60% or more. Nevertheless, the data need not contradict phylogenetic studies which purportedly show that LUCA was a thermophile, because the genetic code likely evolved before LUCA. As was shown in Fig. 1.15, the most likely scenario for the evolution of life before LUCA involves an RNA world, followed by an RNA-protein world (that includes the genetic code) followed by the emergence of DNA and finally the arrival LUCA. Notably, the genetic code is seen as evolving in RNA-based organisms in which $G+C$ correlates with temperatures, unlike in DNA-based organisms, in which it does not, but which came only later. The newfound $G+C$ evidence indicates, therefore, that the genetic code phase of pre-LUCA evolution occurred in mesophile organisms rather than in thermophile ones. If other evidence supports our conclusions about pre-LUCA mesophiles

while LUCA and its immediate relations are confirmed as thermophiles, then we find the intriguing possibility that for some reason our primordial ancestors have transited from a mesophile to an exclusively thermophile mode of existence.[1] Further evidence for this hypothesis may come from several RNA-based systems (like rRNA) which are "relics" from the same epoch of the RNA world as the genetic code. Unfortunately, none of them is as well-preserved as the genetic code.

A challenge to our finding comes from the investigation of the genetic code by Di Giulio [16] using biochemical arguments, which has arrived at the opposite conclusions from our own, and argued that the genetic code evolved in a thermophile. That investigation considered the chemical nature of the amino acids coded in the genetic code, and used the degeneracies as a measure of amino acid preference. Because thermophiles use some amino acids in greater quantities than other amino acids, one can infer the optimal growth temperature based on amino acid frequencies. According to Di Giulio, the genetic code was fixed in thermophile or hyper-thermophile organisms. Therefore, he believes that his finding and our own are irreconcilable (personal communication). Yet, in my view, the situation is different. One possibility is that the structuring of the genetic code continued until the arrival of DNA and then LUCA. Thus, it is possible that while most of the structure of the code took place in a mesophile, subsequent evolution saw a "crawl" toward certain amino acids. This would be consistent with the expected slowing of the rate of code evolution as the emergence of DNA and proofreading allowed for longer genomes.

In Chapter 3, we began an investigation of a type of error-tolerant coding that takes into consideration the semantics of information. That investigation has produced some tools for studying codes like the genetic code. I believe that this investigation may be fruitfully pursued in the future not only because of its mathematical interest, but also because of its future applications. The growth in genetic engineering and especially progress in biochemical synthesis of macromolecules implies that we may soon require techniques for programming complex biochemical instructions. Because errors pervade biochemical systems, such programming would require semantic error coding.

[1] Unfortunately, little can be said about the primordial biosphere as a whole: it is possible that other lineages, which became extinct later, may had continued mesophile or thermophile existence throughout that time.

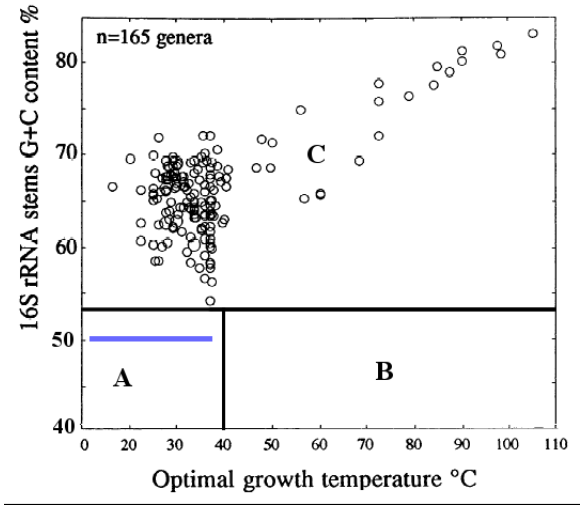

Figure 4.1: **Our G+C findings**
The organism that evolved the genetic code must have been a mesophile, whose possible optimal growth temperatures are indicated with the blue line in zone A. The growth temperature cannot be in the thermophile zone B because of the observed correlation. Notice that if we were to find elevated G+C in zone C, conclusions would have been harder to draw because zone C includes both mesophiles and thermophiles. In modern organisms there appears to be a compensatory effect (Wang [87]) between the relatively elevated G+C of the double-stranded rRNA shown above and the rest of rRNA which tends to be poor in C and very rich in A. Since we do not know if such an effect was present in primordial organisms, there is no contradiction between our results and the absence of any points with $G + C \approx 50\%$. The requirements of heat stability would have forced a hypothetical thermophile with a double-stranded RNA genome to have a much higher $G + C$ content in its genome. Image adapted by adding the low G+C region to an original image by Galtier *et al.* [27].

Appendix A

The Eigen Model

The error-minimization pressures facing primordial organisms are highlighted by the Eigen model [19] - an elementary mathematical model of Darwinian selection. In the following presentation, I will detail an analytic version developed by Bagnoli and Bezzi [4], which I have slightly modified.

Suppose we have a "quasi-species" - a population of primitive organisms closely related to each other and that reproduce asexually. Moreover, suppose that their genotypes and phenotypes are fully specified by strings of letters, where each string has length L. For simplicity, the genotypes are assumed to be haploid, *i.e.* containing one copy of each gene.

This simplistic type of organisms is not unlike the earliest ribo-organisms, who had an RNA genome and produced catalysts directly out of folded RNA. The number of chemical letters in RNA is 4, but for our purposes we could just consider the distinction between purines A, G and pyrimidines U, C. In other words, the organisms can be fully represented by a string of length L with two distinguishable letters (say 0 and 1).

Furthermore, we define a "fitness landscape", i.e., a real-valued function defined on the set of all strings. The greatest fitness, α, is assigned to one "Master Sequence", which could be the strings of all zeros, without a loss of generality. A reduced fitness, β, is assigned to all other sequences. Crucially, fitness is identified with reproductive success. Individuals who have the Master Sequence (X individuals) are more successful in making a copy of themselves than the rest of the population. The mutants, i.e., organisms whose genome does not contain the master sequence, are further subdivided into two subgroups:

those with one mutation away from the Master sequence (Y individuals), and the rest (Z individuals). We suppose for simplicity that reproduction occurs in non-overlapping generations in proportion to the existing population. During reproduction, errors may occur with probability μ at each of the L sites in the genome. Therefore, the probability of correctly copying the genome is $q = (1-\mu)^L$. Under such conditions, the transition from current population ($N = X+Y+Z$) to population at the next time step ($\hat{N} = \hat{X}+\hat{Y}+\hat{Z}$) may be described by the following dynamical equations:

$$\hat{X} = q\alpha X \tag{A.1a}$$

$$\hat{Y} = q\beta Y + (1-q)\alpha X \tag{A.1b}$$

$$\hat{Z} = \beta Z + (1-q)\beta Y \tag{A.1c}$$

$$\hat{N} = \hat{X}+\hat{Y}+\hat{Z} = \alpha X + \beta(Y+Z) \tag{A.1d}$$

Here N is the total population. We neglected the flow into population X because we are interested in the case where the genome is long (large L) and mutations are rare (small μ). Thus, our case is such that when a site experiences a mutation, it is unlikely to mutate back. We also neglected the flow from X into Z since we will be interested in the regime where the mean number of mutations per generation μL is much less than 2. Notice that if the population described by those equations is small, then it may experience an accumulation of mutations know as "Muller's Ratchet", as follows. Because back mutations that restore the master sequence are very rare, a population that starts at the master sequence would after several generations contain only individuals with one mutation or more. Several more generations pass and only individuals with two mutations or more exist. Muller's Ratchet is counteracted by evolution, because the non-mutants reproduce faster. Notice that the ratchet process is unique to asexual reproduction, because in sexual reproduction (involving recombination) the offspring may contain fewer mutations than the parents - a major advantage of sexual of reproduction [38, 69].

Our main interest is in the effect of the mutation rate μ and the fitness landscape (defined through α and β) on the population. Suppose that we start at a population that is dominated by the fittest individuals, i.e., $N \approx X$. Three types of the transitions could occur. The first transition, known as "Error Threshold" occurs when the Master Sequence loses its dominance, and is no longer the most common ($X = Y+Z$). The second transition,

("Muller's Ratchet") corresponds to the loss of the Master Sequence ($X = 0$). Finally, a transition known as the "Mutation Meltdown" occurs when the population becomes extinct ($N = 0$).

All three transitions are biologically significant. The first transition, the error threshold, is important because it relates to how we characterize a species. In any species, the genome of each individual should be very close to the genome corresponding to the average genome of the population. Past the error threshold transition, the average genome in the Eigen model becomes distinguishable from the master sequence and thus the population no longer has the structure of a species. The second transition, Muller's Ratchet, is also biologically significant because it corresponds to the complete disintegration of the former species. The population is now spread over the fitness landscape instead of being localized about the Master Sequence. Muller's Ratchet rarely occurs in large populations because enough individuals always retain the master sequence by chance. When the transition does occur, the species often faces transition to total extinction as well. Namely, the ability of the mutants to reproduce (β) is often lower than necessary for sustainable population. Studies showed that Muller's Ratchet transition and subsequent collapse in the population is the principal ingredient of successful treatment strategies of mutation-prone pathogens like HIV [18].

We will study the above transition by, first, introducing fractional populations $x = X/N, y = Y/N, z = Z/N$ (assuming $N \neq 0$), and second, by finding the equilibrium solutions. In the fractional variables, the dynamical equations, Eqns. A.1, become:

$$\hat{x} = \frac{q\alpha x}{\alpha x + \beta(y+z)} \tag{A.2a}$$

$$\hat{y} = \frac{q\beta y + (1-q)\alpha x}{\alpha x + \beta(y+z)} \tag{A.2b}$$

$$\hat{z} = \frac{\beta z + (1-q)\beta y}{\alpha x + \beta(y+z)} \tag{A.2c}$$

The conditions for equilibrium are $\hat{x} = x, \hat{y} = y, \hat{z} = z$. Bagnoli and Bezzi [4] found two

solutions. The boundary solution $x = 0 = y, z = 1$ and the interior solution:

$$x = \frac{q\alpha - \beta}{\alpha - \beta} \tag{A.3a}$$

$$y = \frac{(1-q)}{q}\frac{\alpha(q\alpha - \beta)}{(\alpha - \beta)^2} \tag{A.3b}$$

$$z = \frac{(1-q)^2}{q}\frac{\alpha\beta}{(\alpha - \beta)^2} \tag{A.3c}$$

It may be seen that the boundary equilibrium solution is unstable, while the interior solution is stable for some choices of the parameters. Stability is guaranteed if we have a contraction mapping from one generation to the next. In particular, the derivatives at a stable equilibrium point p must be less than 1 (e.g. $\frac{d\hat{x}}{dx}|_p = \lim_{x\to p}\frac{f(x,y,z)-f(p,y,z)}{x-p} < 1$).

At the boundary equilibrium point, $\frac{d\hat{x}}{dx}| = \cdots = \frac{q\alpha}{\beta}$ meaning that for α slightly larger than β, even a slight x would grow rapidly (derivative is greater than 1). In contrast, the interior solution is stable because those derivatives, found from Eqns. A.2 at the equilibrium point, are less than 1. To see that, we take the derivatives one at a time. From Eqn. A.2(a):

$$\begin{aligned}
\frac{d\hat{x}}{dx} &= \frac{q\alpha D - q\alpha x\alpha}{D^2} \\
&= \frac{q\alpha(D - x\alpha)}{D^2} \\
&= \frac{q\alpha}{(q\alpha)^2}\left(q\alpha - \alpha\frac{q\alpha - \beta}{\alpha - \beta}\right) \\
&= \frac{1}{q}\frac{q\alpha - q\beta - q\alpha + \beta}{\alpha - \beta} \\
&= \frac{\beta(1-q)}{q(\alpha - \beta)} < 1
\end{aligned}$$

since $(1 - q)$ is much smaller than the other terms. Here D is the denominator, $D = \alpha x + \beta(y + z)$ which equals $q\alpha$ at the equilibrium point. Similarly, from Eqn. A.2(b) we have

$$\begin{aligned}
\frac{d\hat{y}}{dy} &= \frac{q\beta D - (q\beta y + (1-q)\alpha x)\beta}{D^2} \\
&= \cdots \\
&= \frac{\beta}{\alpha} - \frac{\beta}{q^2}\frac{1-q}{(\alpha - \beta)^2}(q\alpha - \beta) < 1
\end{aligned}$$

for $q\alpha > \beta$, which corresponds approximately to $\alpha > 1$. Finally, from Eqn. A.2(c) we have

$$\begin{aligned} \frac{d\hat{z}}{dz} &= \frac{\beta D - (\beta z + (1-q)\beta y}{D^2} \\ &= \dots \\ &= \frac{\beta}{\alpha q} - \frac{\beta^2}{q^2}\frac{(1-q)^2}{(\alpha-\beta)^2} < 1 \end{aligned}$$

for $q\alpha > \beta$.

This equilibrium point provides a convenient locale for investigating the values of the parameters that correspond to each of the transitions. Notice that we needed to introduce the fractional variables since they allowed us to find an equilibrium point even if the size of the overall population is changing. The first transition, the error threshold, occurs when the master sequence becomes non-dominant, and it is, to a good approximation, the case $x = y + z \approx y$. From Eqns. A.3, $x = y$ corresponds to

$$q_e = \frac{\alpha}{2\alpha - \beta} \quad \text{Error Threshold} \tag{A.4}$$

The approximation we have made means that the actual transition occurs at a slightly higher x and hence a higher q. Recall that $q = (1-\mu)^L$ is the probability of correctly copying the genome. We see that in the extreme case of $1 < \beta \ll \alpha \to \infty$, selection can preserve the master sequence even if $q \to \frac{1}{2}$.

Onset of Muller's Ratchet, *i.e.*, the disappearance of the master sequence, corresponds to the condition $\frac{\hat{x}}{x} < 1$ in the limit of small x. We obtain that the critical q is

$$q_r = \frac{\beta}{\alpha} \quad \text{Muller's Ratchet} \tag{A.5}$$

Thus, the master sequence is maintained as long as the probability of correct copying, q, is kept above β/α. Let us find the relationship between genome length and the probability of error that must hold if the master sequence is to be maintained. To that end, it is convenient to introduce σ - labeling the relative selective advantage of the master sequence

over the mutants, $\sigma = \alpha/\beta$. We must have that $q = (1-\mu)^L > \frac{\beta}{\alpha} = 1/\sigma$, hence:

$$\begin{aligned} \frac{1}{\sigma} &< q \\ &= (1-\mu)^L \\ &= \exp\left(L\ln(1-\mu)\right) \\ &\approx \exp\left(L(-\mu)\right) \end{aligned}$$

Thus, $-\ln(\sigma) < -L\mu$. Hence, the maximum genome length that selection can maintain in the face of errors, L_{max}, satisfies:

$$L_{max} < \frac{\ln\sigma}{\mu} \tag{A.6}$$

(see Fig. A.1). Because L_{max} depends not on σ but on $\ln\sigma$, increasing the selective pressure pays rapidly diminishing returns. In contrast, increasing the replicative fidelity by decreasing μ leads to direct gains in L_{max}.

Because the mutants are often inviable, the loss of the master sequence is often the harbinger of extinction. Recall that $\hat{N} = \alpha X + \beta(Y+Z)$. Extinction occurs if $\frac{\hat{N}}{N} = \alpha x + \beta(y+z) = q\alpha < 1$. Therefore, the critical transition is at

$$q_d = \frac{1}{\alpha} \quad \text{Extinction through Mutation Meltdown} \tag{A.7}$$

In other words, if the probability of correctly replicating falls to $1/\alpha$, the species becomes extinct.

In summary, we saw how selection, expressed through σ counteracts the mutation pressure expressed through μ. This simplistic model of Eigen's illustrates the crucial importance of error-minimization in early evolution. We also saw how three disadvantageous transitions occur at certain fixed values of q, the replication fidelity. There is an inverse relationship between the error rate experienced by an organism and the length of its genome. Lowering the error rate in the replication machinery allows L to rise, and hence for the organism to grow in complexity.

Eigen, who introduced these relations, noted that evolution may have been able to sustain longer genomes than allowed by Eqn. A.6. He argued that the functionality contained in the genome may have been distributed among self-replicating but interdependent

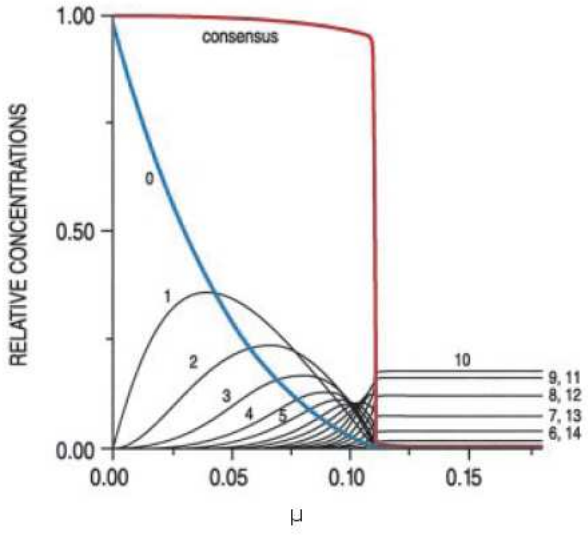

Figure A.1: **Relative populations of binary sequences as functions of error rate μ** $L = 20$ and the selective advantage is $\sigma = 10$. The resulting quasispecies distribution is centered at the master sequence ("0" errors). The numbers 1, 2, . . . 20 label the populations of 1-, 2-, . . . 20-error mutants, respectively. The red curve indicates the "consensus" or average sequence of the population. There is a sharp first-order phase transition at the predicted threshold of $\mu = \ln \sigma / L \approx 0.12$. Image adapted from an original by M. Eigen [18]

organisms, in what he called a "Hypercycle". Selection would then act on each one of these organisms, and thus maintain the information contained in the genome. Considered collectively, they could maintain a metabolic function that was equivalent to the function of a large complex organism. An example of a hypercycle of mutual assistance between replicators (for purposes other than maintaining long genomes) is viral infection. During the initial stages of an HIV infection, for example, the growth of the viral population is not in proportion to population, but faster - in proportion to its square. This is due to mutual assistance the virions provide to each in suppressing the immune system [18]. Unfortunately, hypercycles are relatively rare because they require too many pre-conditions (e.g. selection must act individually on all the components) and thus they do not provide

a convincing resolution to the error rate problem.

There exists another method for increasing the complexity of an organism, without invoking additional selective forces. This is of course, for the organism to evolve a genetic code that decreases the phenotypical impact of those errors at a fixed q. While the precise effect of the error-minimization in the genetic code on fitness is unknown, it must have been significant since, as we saw in discussion of channel cascades (subsection 2.2.4), error-minimization produces a power law decrease in the accumulation of errors. In the near future I intend to develop a more advanced version of the Eigen model to examine the above argument quantitatively. The idea is to simulate and compare two populations of "Eigen organisms" which, unlike in the model above, possess a genetic code. Consequently, there will be a distinction between the genotype and the phenotype, unlike in the standard model. Like in the Eigen model, mutations act on the genome but selection acts on the phenotype, which is a sequence of amino acids. The crux of the simulation would be to find the difference in fitness between the two populations. One population would have a genetic code not designed to reduce errors, while the other would have a the standard code - a highly-efficient error-reducing code.

There are several questions such a simulation could address. First of all, it would be possible to find the selective advantage of an error-reducing genetic code, thus possibly, obtain further support for the error-minimization theory. In particular, it would be possible to find how much larger a genome can a quasi-species sustain given that it has an error-reducing code. I will be looking at the points where the three key transitions occur in those populations - error catastrophe, mutational meltdown and Muller's ratchet.

Second, it may be possible to find another reason why there is a factor of $64/21 \approx 3$ redundancy in the genetic code. It could be that the number of amino acids corresponds to an optimal tradeoff between catalytic efficiency and redundant coding. Recall, that it has been proposed that the number of amino acids represents a tradeoff between catalytic efficiency and translation fidelity but not coding fidelity, although they are related. The simulation proposed above may provide the first quantitative argument to support this conclusion.

Appendix B

Biological Glossary

Definition B.1 (Codon) - *aka. triplet, are three nucleotides of either RNA of DNA which encode exactly one amino acid. There are also several codons which encode a termination signal.*

Definition B.2 (Deletion error) - *a genetic error in which a nucleotide is not copied or otherwise eliminated. It is highly damaging because it causes a shift in the reading frame and corrupts the subsequent protein. Deletion errors are much rarer than substitutions* i.e. *"point errors".*

Definition B.3 (DNA) - *or "Deoxyribonucleic Acid" is a family of molecules which carry genetic information. DNA often coils into a double helix of two chemically bound "strands". In organisms that use DNA, it is found in several pieces known as units. Each unit has either a circular or a linear topology. Double-standed DNA molecule is a polymer consisting of a stack of pairs of nucleotides.*

Definition B.4 (Enzyme) - *a protein performing a catalytic function.*

Definition B.5 (mRNA) - *aka. "Messenger RNA" is an intermediate carrier of genetic material. mRNA is created during transcription from DNA. mRNA is then attached to the rybozome, where its information is used to build a protein.*

Definition B.6 (Nucleotide) - *a single unit of RNA or DNA. The RNA nucleotides are Uracil, Cytosine, Adenine and Guanine (U, C, A, G). In DNA these are Thymine, Cytosine, Adenine and Guanine (T, C, A, G). DNA nucleotides form pairs and chains (or helixes). C pairs with G and T pairs with A. In RNA, the pairs are C with G and U with A, although other pairs are found as well. U, T and C all belong to the pyrimidine family (abbreviated Y), characterized by a single ring. They all pair with A and G - members of the purine family, characterized by a double-ring.*

Definition B.7 (RNA) - *"Ribonucleic Acid" is a family of molecules used to carry genetic information. RNA molecules are chains (polymers) formed mostly from four RNA nucleotides. RNA can coil, but is often found in other geometries. RNA is more chemically active than DNA, and can be used as a catalyst, not only as a carrier of information.*

Definition B.8 (RNA World) - *The dominant hypothesis about an early stage in the evolution of life. It not known exactly when the RNA World existed, but probably not later than 3.5 GY ago. "Riboorganisms" that purportedly existed in the RNA world had an RNA-based genome and used RNA chains in the building of ribozymes. The latter would have performed catalytic functions. The RNA World hypothesis has two basic advantages. Firstly, it explains an important chicken-and-egg problem in the evolution of cells. Cells need a genetic molecule (e.g. DNA) to build catalysts (e.g. proteins), yet a genetic molecule is useless unless there exists a catalytical machinery that can read it and build proteins. Thus, proteins need DNA and DNA needs proteins, and their complexity makes co-evolution implausible. RNA was shown to be capable of both tasks. It is believed that it was RNA-based organisms who evolved the ability to encode proteins and to use DNA. Secondly, the RNA world hypothesis explains the presence of RNA in modern organisms, most notably in the role of genetic intermediates (mRNA and tRNA). These are believed to be relics from the RNA world.*

Definition B.9 (Protein) - *a chain of amino acids with a specific shape, typically performing a catalytic function in the cell. Often, a protein corresponds to a single gene.*

Definition B.10 (Substitution error) - *aka. a point error, is a type of genetic error in which one nucleotide is replaced by another.*

Definition B.11 (Taxon) *(plural: taxa) - a species, genus, family or other biological class of organisms. The following are examples of taxons:* canis familiaris*, fruit flies, ciliates.*

Definition B.12 (tRNA) *- aka. "Transfer RNA", a class of molecules performing a key role in translation. Its most important structures are first, an anti-codon and secondly, an acceptor stem. The anticodon binds to a codon in mRNA while the acceptor stem binds to an amino acid. Each tRNA molecule consists of about 80 RNA nucleotides.*

Definition B.13 (Transitions) *- the most common type of substitutions, involving misreading or miscopying within one chemical family. For example, the replacement of the purine C with a purine U is such an error. There are four types of transition errors.*

Definition B.14 (Transversions) *- a relatively rare type of substitution in which a nucleotide is replaced with a nucleotide from another chemical family (e.g. purine C with pyrimidine A).*

Definition B.15 (Transcription) *- one of the steps in the reading of genetic material. During transcription, a sequence of DNA is copied (with some modification) into a sequence of mRNA.*

Definition B.16 (Translation) *- one of the steps in the reading of genetic material. During translation, the information in the mRNA "transcript" becomes a protein. A single mRNA could be used in the synthesis of many identical proteins.*

Definition B.17 (Triplet) *- see Codon.*

Appendix C

Matlab and AMPL programs

The following programs were used in the production of the data:

1. **getAdvantage.m** Matlab program for estimating the evolutionary advantage of the genetic code compared to codes of the same class
2. **tolerance.m** Matlab program for calculating the tolerance and safety expressions based on the structure of the genetic code and the EX values
3. **transmission.m** Matlab program for calculating the transmission function based on the structure of the genetic code
4. **ts.method.mod** AMPL model expressing the Tolerance-Safety Method
5. **ts.method.oneRun.run** AMPL script for executing **ts.method.mod**

C.1 getAdvantage.m

```
function dataSum = getAdvantage(arg_biases)
%find how much of an error tolerance advantage does the std code get over
%alternative codes by looking at the fall in EX of a long seqeunce

global Pi;
global testSeq;
global biases;
global aminoEXTable;
%global stopsIncluded;
```

```
if nargin == 0
    biases = [0.3333 0.3333 0.3333];
else
    biases = arg_biases;
end


%rand('state',0);
rand('state',sum(100*clock));

aminoEXTable = setupAminoTable();

seqLen = 10000;

[Pi, M, testSeq]  = setupPiMSeq(seqLen); %M must be normalized to 1

epsilon1    = 0.001;
epsilon2    = 1;
numTrials   = 200;

global origCode;
origCode    = STDCode();
origDatas   = zeros(numTrials, 1);

numDeranged = 100;
derangCodes = makeDerangedCodes(numDeranged);
derangDatas = zeros(numTrials, numDeranged);

numRandom   = 100;
randomCodes = makeRandomCodes(numRandom);
randomDatas = zeros(numTrials, numRandom);

epsilons    = zeros(numTrials, 1);
for i=1:numTrials
    epsilon     = epsilon1 * 10^((i-1)/numTrials*log10(epsilon2/epsilon1));
    %epsilon     = epsilon1 + (i-1)/numTrials*(epsilon2 - epsilon1);
    epsilons(i) = epsilon;
    curM        = epsilon * M;
    newSeq      = mutateSeq(curM);

    origDatas(i)   = fitness(newSeq, origCode);

    for j=1:numDeranged
        derangDatas(i,j)   = fitness(newSeq, derangCodes{j});
    end

    for j=1:numRandom
        randomDatas(i,j)   = fitness(newSeq, randomCodes{j});
    end
end

derangMean = mean(derangDatas, 2);
derangSTD  = std(derangDatas, 0, 2);

randomMean = mean(randomDatas, 2);
randomSTD  = std(randomDatas, 0, 2);

%semilogx(epsilons, origDatas, epsilons, derangMean, epsilons, randomMean);
%blue, green, red...

%semilogx(epsilons, derangMean);
%semilogx(epsilons, derangSTD);
dataSum{1} = origDatas;
dataSum{2} = derangDatas;
```

```
dataSum{3} = randomDatas;
dataSum{4} = epsilons;

%semilogx(dataSum{4}, mean(dataSum{1}), 'b', dataSum{4}, mean(dataSum{2}), 'g', dataSum{4}, mean(dataSum{3}));
%blue, green, red...
1


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Pi, M, testSeq] = setupPiMSeq(len)
%

Pi = [0.25, 0.25, 0.25, 0.25];

%for computational reasons, we shall use a 4x3 table, so each column skips
%the no-mutation case
%M(source, destination)
M  = ones(4,3);

M(1,1) = 4;
M(2,1) = 4;
M(3,3) = 4;
M(4,3) = 4;

M = M/sum(sum(M));

tmpPi(1) = Pi(1);
tmpPi(2) = tmpPi(1) + Pi(2);
tmpPi(3) = tmpPi(2) + Pi(3);
tmpPi(4) = tmpPi(3) + Pi(4); %redundant

%too crude?:  testSeq = ceil(4*rand(100,3));
testSeq(len,3) = 0;
tmpRnd = rand(len,3);
for i=1:len
    for pos=1:3
        x = tmpRnd(i,pos);
        if x > tmpPi(3)
            testSeq(i,pos) = 4;
        elseif x > tmpPi(2)
            testSeq(i,pos) = 3;
        elseif x > tmpPi(1)
            testSeq(i,pos) = 2;
        else
            testSeq(i,pos) = 1;
        end
    end
end

%test: this should give rought the same numbers
%x(1) = length(find(~(1*ones(100,3)-testSeq)));
%x(2) = length(find(~(2*ones(100,3)-testSeq)));
%x(3) = length(find(~(3*ones(100,3)-testSeq)));
%x(4) = length(find(~(4*ones(100,3)-testSeq)))
%x = x/sum(x);


function code = STDCode()
%
code(4,4,4) = 0;

code(1,:,:) = [18 18 16 16 ; 16 16 16 16; 15 15 15 14; 17 17 17 17];
code(2,:,:) = [2 2 2 2 ; 4 4 4 4; 3 3 3 3; 5 5 5 5];
code(3,:,:) = [19 19 21 21 ; 11 11 10 10; 7 7 13 13; 8 8 9 9];
```

```
code(4,:,:) = [1 1 21 20 ; 12 12 12 12; 2 2 12 12; 6 6 6 6];

code = permute(code, [2 1 3]);


function codes = makeDerangedCodes(numCodes)

global origCode;

codes = cell(21,1);

for c=1:numCodes
    newPositions  = zeros(21,1);
    taken         = zeros(21,1);
    for i=1:21
        tmpPos = ceil(21*rand());
        while taken(tmpPos) == 1
            tmpPos = mod(tmpPos, 21) + 1;
        end
        newPositions(i) = tmpPos;
        taken(tmpPos)   = 1;
    end

    code = zeros(4,4,4);
    for i=1:64
        code(i) = newPositions(origCode(i));
    end

    codes{c} = code;
end


function codes = makeRandomCodes(numCodes)

codes = cell(21,1);

for c=1:numCodes
    %first ensure all aminos are there, then fill in the rest
    code = zeros(4,4,4);
    for i=1:21
        tmpCodon = ceil(ceil(64*(rand()+0.0000000001))-0.0000000001);
        while code(tmpCodon) ~= 0
            tmpCodon = mod(tmpCodon, 64) + 1;
        end
        code(tmpCodon) = i;
    end

    for i=1:64
        if code(i) == 0
            code(i) = ceil(21*rand());
        end
    end

    codes{c} = code;
end


function aminoEXTable = setupAminoTable();
%
%aminoEXTable(source, destination)
%C,S,T,P,A,G,N,D,E,Q,H,R,K,M,I,L,V,F,Y,W,EXsrc
aminoEXTable(1,:)  = [1000,258,121,201,334,288,109,109,270,383,258,306,252,169,109,347,89,349,349,139, 0];
aminoEXTable(2,:)  = [373,1000,481,249,490,418,390,314,343,352,353,363,275,321,270,295,358,334,294,160, 0];
aminoEXTable(3,:)  = [325,408,1000,164,402,332,240,190,212,308,246,299,256,152,198,271,362,273,260,66, 0];
```

```
aminoEXTable(4,:)  = [345,392,286,1000,454,404,352,254,346,384,369,254,231,257,204,258,421,339,298,305, 0];
aminoEXTable(5,:)  = [393,384,312,243,1000,387,430,193,275,320,301,295,225,549,245,313,319,305,286,165, 0];
aminoEXTable(6,:)  = [267,304,187,140,369,1000,210,188,206,272,235,178,219,197,110,193,208,168,188,173, 0];
aminoEXTable(7,:)  = [234,355,329,275,400,391,1000,208,257,298,248,252,183,236,184,233,233,210,251,120, 0];
aminoEXTable(8,:)  = [285,275,245,220,293,264,201,1000,344,263,298,252,208,245,299,236,175,233,227,103, 0];
aminoEXTable(9,:)  = [332,355,292,216,520,407,258,533,1000,341,380,279,323,219,450,321,351,342,348,145, 0];
aminoEXTable(10,:) = [383,443,361,212,499,406,338,68,439,1000,396,366,354,504,467,391,603,383,361,159, 0];
aminoEXTable(11,:) = [331,365,205,220,462,370,225,141,319,301,1000,275,332,315,205,364,255,328,260,72, 0];
aminoEXTable(12,:) = [225,270,199,145,459,251,67,124,250,288,263,1000,306,68,139,242,189,213,272,63, 0];
aminoEXTable(13,:) = [331,376,476,252,600,492,457,465,272,441,362,440,1000,414,491,301,487,360,343,218, 0];
aminoEXTable(14,:) = [347,353,261,85,357,218,544,392,287,394,278,112,135,1000,612,513,354,330,308,633, 0];
aminoEXTable(15,:) = [362,196,193,145,326,160,172,27,197,191,221,124,121,279,1000,417,494,331,323,73, 0];
aminoEXTable(16,:) = [366,212,165,146,343,201,162,112,199,250,288,185,171,367,301,1000,275,336,295,152, 0];
aminoEXTable(17,:) = [382,326,398,201,389,269,108,228,192,280,253,190,197,562,537,333,1000,207,209,286, 0];
aminoEXTable(18,:) = [176,152,257,112,236,94 ,136,90 ,62 ,216,237,122,85,255,181,296,291,1000,332,232, 0];
aminoEXTable(19,:) = [142,173,  276  ,194,402,357,129,87,176,369,197,340,171,392,  276  ,362,  282  ,360,1000,303, 0];
aminoEXTable(20,:) = [137,92,17,66,63,162,  200  ,  184  ,65,61,239,103,54,110,  218  ,177,110,364,281,1000, 0];
%values separated by spaces were calculated as suggested in Yampolsky et al.

%errors to and from stop codons have similarity = 0, and between two stops = 1
%destination(notice the zeros) on last column
aminoEXTable(21,:) = zeros(21,1);
aminoEXTable(21,21)= 1000;

%first one is the source, and the second one is the destination AA
%similarityAA = aminoEXTable(amino1, amino2);


function newSeq = mutateSeq(curM)
%

global testSeq;
global biases; %ZZZ: do biases effectively degrease epsilon by 1/3!

len = length(testSeq);
%the nuc freq. are already built in into the seq frequencies
% Q   = zeros(4,3);
% Q(1,:) = Pi(1)*M(1,:);
% Q(2,:) = Pi(2)*M(2,:);
% Q(3,:) = Pi(3)*M(3,:);
% Q(4,:) = Pi(4)*M(4,:);

tmpM(:,1) = curM(:,1);
tmpM(:,2) = tmpM(:,1) +  curM(:,2);
tmpM(:,3) = tmpM(:,2) +  curM(:,3);

rnd = rand(len, 3);
newSeq = testSeq;

mutLookup(4,3) = 0;
mutLookup(1,:) = [2 3 4];
mutLookup(2,:) = [1 3 4];
mutLookup(3,:) = [1 2 4];
mutLookup(4,:) = [1 2 3];

for i=1:len
    for pos=1:3
        x = rnd(i,pos);
        nuc = testSeq(i,pos);
        if x > biases(pos)*tmpM(nuc,3)
        elseif x > biases(pos)*tmpM(nuc,2)
            newSeq(i, pos) = mutLookup(nuc,3);
        elseif x > biases(pos)*tmpM(nuc,1)
            newSeq(i, pos) = mutLookup(nuc,2);
```

```
        else
            newSeq(i, pos) = mutLookup(nuc,1);
        end
    end
end


function fitness = fitness(newSeq, code)
%
global testSeq;
global aminoEXTable;

len = length(testSeq);
netEX = 0;
for i=1:len
    amino1 = code(testSeq(i,1), testSeq(i,2), testSeq(i,3));
    amino2 = code(newSeq(i,1),  newSeq(i,2),  newSeq(i,3));

    netEX = netEX + aminoEXTable(amino1, amino2);
end

fitness = netEX/len;
```

C.2 tolerance.m

```
function tolerance()
% Copyright 2005, Alexander Gutfraind.  Free academic use.
%
% Finds the expressions for the 12 Tij expressions

global M muu muc mua mug  mcu mcc mca mcg   mau mac maa mag   mgu mgc mga mgg;
syms M muu muc mua mug  mcu mcc mca mcg   mau mac maa mag   mgu mgc mga mgg;
global q qu qc qa qg
syms q qu qc qa qg;

M = [muu muc mua mug;  mcu mcc mca mcg;   mau mac maa mag;   mgu mgc mga mgg];
%M here is the transpose of the traditional translation matrix.  this way is more convenient

q = [qu, qc, qa, qg];
%these are the Pi values

global codeType;
global lookuptable
if nargin > 0
    codeType = 'user_specified';
    lookuptable = code;
else
    lookuptable = setupEncoding();
end

global aminoEXTable;
aminoEXTable = setupAminoTable();
global biases pos1 pos2 pos3;
syms biases pos1 pos2 pos3;
global stopsIncluded;
syms stopsIncluded;
biases = [pos1 pos2 pos3];

totalTable(4,4) = sym(0);
for i = 1:4
    for j = 1:4
```

```
        for k = 1:4
    codon = [i, j, k];
    totalTable = totalTable + substitutionTable(codon);
        end
    end
end

translation = totalTable;

fileName = strcat('data\translation_eqns_', datestr(now,'mmm.dd.yyyy.HH.MM.SS'), '.txt');
fid = fopen(fileName,'w');

fprintf(fid,'\r\nparam codeType := "%s" symbolic;\r\n\r\n', codeType);

%note: ordering: table(strtNuc, endNeuc)
fprintf(fid,'var mucTolerance = %s;\r\n', char(translation(1,2)));
fprintf(fid,'var mcuTolerance = %s;\r\n', char(translation(2,1)));
fprintf(fid,'var muaTolerance = %s;\r\n', char(translation(1,3)));
fprintf(fid,'var mauTolerance = %s;\r\n', char(translation(3,1)));
fprintf(fid,'var mugTolerance = %s;\r\n', char(translation(1,4)));
fprintf(fid,'var mguTolerance = %s;\r\n', char(translation(4,1)));

fprintf(fid,'var mcaTolerance = %s;\r\n', char(translation(2,3)));
fprintf(fid,'var macTolerance = %s;\r\n', char(translation(3,2)));
fprintf(fid,'var mcgTolerance = %s;\r\n', char(translation(2,4)));
fprintf(fid,'var mgcTolerance = %s;\r\n', char(translation(4,2)));
fprintf(fid,'var magTolerance = %s;\r\n', char(translation(3,4)));
fprintf(fid,'var mgaTolerance = %s;\r\n', char(translation(4,3)));

fclose(fid);
1;

function table = substitutionTable(strtCodon)
%a 4x4 table corresponding to avg Tolerance of substitutions from the given codon
global lookuptable;
global q;
global M;
global biases;
global stopsIncluded;
global aminoEXTable;

strtAcid = lookuptable(strtCodon(1), strtCodon(2), strtCodon(3));

tmpTable(4,4) = sym(0);

%we are gonna only consider Hamming-distance one mutations
for endNeuc = 1:4
    for posit = 1:3
        endCodon = strtCodon;
        endCodon(posit) = endNeuc;

        strtNeuc = strtCodon(posit);

        endAcid  = lookuptable(endCodon(1), endCodon(2), endCodon(3));
        if strtNeuc == endNeuc
            continue;
        elseif endAcid ~= 21
            prefix = 1;
        else
            prefix = stopsIncluded;
        end
        otherNeucs = q(strtCodon(1))*q(strtCodon(2))*q(strtCodon(3))/q(strtCodon(posit));
        newData = prefix*biases(posit)*aminoEXTable(strtAcid, endAcid)*otherNeucs;
        tmpTable(strtNeuc, endNeuc) = tmpTable(strtNeuc, endNeuc) + newData;
```

```
    end
end

if strtAcid ~= ' '
    table = tmpTable;
else
    table = stopsIncluded*tmpTable;
end

function lookuptable = setupEncoding()
%
global codeType;

codeType = 'standard_code';
lookuptable(1,:,:) = [18 18 16 16   ; 16 16 16 16   ; 15 15 15 14   ; 17 17 17 17];
lookuptable(2,:,:) = [2 2 2 2       ; 4 4 4 4       ; 3 3 3 3       ; 5 5 5 5   ];
lookuptable(3,:,:) = [19 19 21 21   ; 11 11 10 10   ; 7 7 13 13     ; 8 8 9 9   ];
lookuptable(4,:,:) = [1 1 21 20     ; 12 12 12 12   ; 2 2 12 12     ; 6 6 6 6   ];
lookuptable = permute(lookuptable, [2 1 3]);


function aminoEXTable = setupAminoTable();
%
%aminoEXTable(source, destination)
%C,S,T,P,A,G,N,D,E,Q,H,R,K,M,I,L,V,F,Y,W,EXsrc
aminoEXTable(1,:)  = [1000,258,121,201,334,288,109,109,270,383,258,306,252,169,109,347,89,349,349,139, 0];
aminoEXTable(2,:)  = [373,1000,481,249,490,418,390,314,343,352,353,363,275,321,270,295,358,334,294,160, 0];
aminoEXTable(3,:)  = [325,408,1000,164,402,332,240,190,212,308,246,299,256,152,198,271,362,273,260,66, 0];
aminoEXTable(4,:)  = [345,392,286,1000,454,404,352,254,346,384,369,254,231,257,204,258,421,339,298,305, 0];
aminoEXTable(5,:)  = [393,384,312,243,1000,387,430,193,275,320,301,295,225,549,245,313,319,305,286,165, 0];
aminoEXTable(6,:)  = [267,304,187,140,369,1000,210,188,206,272,235,178,219,197,110,193,208,168,188,173, 0];
aminoEXTable(7,:)  = [234,355,329,275,400,391,1000,208,257,298,248,252,183,236,184,233,233,210,251,120, 0];
aminoEXTable(8,:)  = [285,275,245,220,293,264,201,1000,344,263,298,252,208,245,299,236,175,233,227,103, 0];
aminoEXTable(9,:)  = [332,355,292,216,520,407,258,533,1000,341,380,279,323,219,450,321,351,342,348,145, 0];
aminoEXTable(10,:) = [383,443,361,212,499,406,338,68,439,1000,396,366,354,504,467,391,603,383,361,159, 0];
aminoEXTable(11,:) = [331,365,205,220,462,370,225,141,319,301,1000,275,332,315,205,364,255,328,260,72, 0];
aminoEXTable(12,:) = [225,270,199,145,459,251,67,124,250,288,263,1000,306,68,139,242,189,213,272,63, 0];
aminoEXTable(13,:) = [331,376,476,252,600,492,457,465,272,441,362,440,1000,414,491,301,487,360,343,218, 0];
aminoEXTable(14,:) = [347,353,261,85,357,218,544,392,287,394,278,112,135,1000,612,513,354,330,308,633, 0];
aminoEXTable(15,:) = [362,196,193,145,326,160,172,27,197,191,221,124,121,279,1000,417,494,331,323,73, 0];
aminoEXTable(16,:) = [366,212,165,146,343,201,162,112,199,250,288,185,171,367,301,1000,275,336,295,152, 0];
aminoEXTable(17,:) = [382,326,398,201,389,269,108,228,192,280,253,190,197,562,537,333,1000,207,209,286, 0];
aminoEXTable(18,:) = [176,152,257,112,236,94 ,136,90 ,62 ,216,237,122,85,255,181,296,291,1000,332,232, 0];
aminoEXTable(19,:) = [142,173,  276  ,194,402,357,129,87,176,369,197,340,171,392,  276  ,362,  282  ,360,1000,303, 0];
aminoEXTable(20,:) = [137,92,17,66,63,162,  200  ,  184  ,65,61,239,103,54,110,  218  ,177,110,364,281,1000, 0];
%values separated by spaces were calculated as suggested in Yampolsky et al.
%EX values are the above values / 1000

%errors to and from stop codons have 0 proximity, between two stops 1000
%destination(notice the zeros) on last column
aminoEXTable(21,:) = zeros(21,1);
aminoEXTable(21,21)= 1000;

%debug
%aminoEXTable = diag(ones(21,1));
```

C.3 transmission.m

```
function transmission = transmission()
% find out what is the transmission function of the genetic code
```

```
sym transmission;
global codeType;
global M muu muc mua mug  mcu mcc mca mcg   mau mac maa mag   mgu mgc mga mgg;
syms M muu muc mua mug  mcu mcc mca mcg   mau mac maa mag   mgu mgc mga mgg;
global q qu qc qa qg
syms q qu qc qa qg;

global biases pos1 pos2 pos3;
syms biases pos1 pos2 pos3;
global stopsIncluded;
syms stopsIncluded;
biases = [pos1 pos2 pos3];

M = [muu muc mua mug;  mcu mcc mca mcg;   mau mac maa mag;   mgu mgc mga mgg];
%M here is the transpose of the traditional translation matrix.  this way is more convenient
q = [qu, qc, qa, qg];

global lookuptable
lookuptable = setupEncoding();

%transmission = muu + muu;
%transmission = probAX('W')
transmission = probAY('Y');
%transmission = probJointA('F', 'H');
%transmission = probAX('S');

Hx = sym(0);
for i = 1:21
    acid = aminoAcidFromNumber(i);
    p = probAX(acid);
    if i ~= 21
        Hx = Hx - p*log(p);
    else
        Hx = Hx - stopsIncluded*p*log(p);
    end
end

Hxy = sym(0);
for i = 1:21
    acidX = aminoAcidFromNumber(i);
    sym terms(21);
    terms(21) = sym(0);
    for j = 1:21
        acidY = aminoAcidFromNumber(j);
        pxy = probJointA(acidX, acidY);
        %pxgy = pxy/probAY(acidY);
        %Hxy = Hxy - pxy*log(pxgy);
        %notice that we don't add Hxy but rather subtract it, so it's really log(py/pxy), as it should be
        terms(j) = - pxy*log(pxy/probAY(acidY));
    end
    Hxy = Hxy  + terms(1) + terms(2) + terms(3) + terms(4) + terms(5) + terms(6)+ terms(7) + terms(8) + terms(9)+ terms(10)...
         +terms(11) + terms(12) + terms(13) + terms(14) + terms(15) + terms(16)+ terms(17) + terms(18) + terms(19) + terms(20) + terms(21);
    clear terms;
    done = i
end
transmission = Hx - Hxy;
%transmission = Hx;
fileName = strcat('transmission_optfn_', datestr(now,'mmm.dd.yyyy.HH.MM.SS'), '.txt');
fid = fopen(fileName,'w');
fprintf(fid,'\r\nparam codeType := "%s" symbolic;\r\n\r\n', codeType);
fprintf(fid,'%s',char(transmission));
fclose(fid);
disp('Output File Written!')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function lookuptable = setupEncoding();
global codeType;
%note: this encoding here is different from the previous two functions,
%because they use the EX table indices, not alphabetical indices
codeType = 'standard_code';
lookuptable(1,:,:) = ['F' 'F' 'L' 'L' ; 'L' 'L' 'L' 'L'; 'I' 'I' 'I' 'M'; 'V' 'V' 'V' 'V'];
lookuptable(2,:,:) = ['S' 'S' 'S' 'S' ; 'P' 'P' 'P' 'P'; 'T' 'T' 'T' 'T'; 'A' 'A' 'A' 'A'];
lookuptable(3,:,:) = ['Y' 'Y' ' ' ' ' ; 'H' 'H' 'Q' 'Q'; 'N' 'N' 'K' 'K'; 'D' 'D' 'E' 'E'];
lookuptable(4,:,:) = ['C' 'C' ' ' 'W' ; 'R' 'R' 'R' 'R'; 'S' 'S' 'R' 'R'; 'G' 'G' 'G' 'G'];

lookuptable = permute(lookuptable, [2 1 3]);



function name = aminoAcidFromNumber(number)
% simple conversion from numbers to names for easy iterations
switch number
    case 1
        name = 'A';
    case 2
        name = 'C';
    case 3
        name = 'D';
    case 4
        name = 'E';
    case 5
        name = 'F';
    case 6
        name = 'G';
    case 7
        name = 'H';
    case 8
        name = 'I';
    case 9
        name = 'K';
    case 10
        name = 'L';
    case 11
        name = 'M';
    case 12
        name = 'N';
    case 13
        name = 'P';
    case 14
        name = 'Q';
    case 15
        name = 'R';
    case 16
        name = 'S';
    case 17
        name = 'T';
    case 18
        name = 'V';
    case 19
        name = 'W';
    case 20
        name = 'Y';
    case 21
%       error('Search for stop codon!');
        name = ' ';
```

```
    otherwise
        error('Acid number out of bounds!');
end


function listCodons = listCodons(A)
%returns an array of the codon of amino acid A
global lookuptable;
global q;
global M;

degen = 0;
for i = 1:4
   for j = 1:4
      for k = 1:4
          if lookuptable(i,j,k) == A
             degen = degen + 1;
             codons{degen} = [i, j, k];
          end
      end
    end
end
if degen == 0
    error('Amino acid unknown!')
    return
end;

listCodons = codons;


function probAX = probAX(X)
%return the probability of
%input amino acid X, for example X = L
%p(a) = sum_(all ijk meaning a) p(ijk)
global lookuptable;
global q;

tmp = sym(0);
codonsX = listCodons(X);
degenX  = length(codonsX);
for i = 1:degenX
    codon = codonsX{i};
    tmp = tmp + q(codon(1))*q(codon(2))*q(codon(3));
end
if isequal(tmp, sym(0))
    error('X amino acid unknown!')
    return
end;
probAX = tmp;

function probAY = probAY(Y)
%return the probability of
%output amino acid Y, for example Y = L
%p(a) = sum_(all ijk meaning a) p(ijk)
global lookuptable;
global q;
global M;
global biases;
global stopsIncluded;

codonsY = listCodons(Y);
degenY  = length(codonsY);
tmp = sym(0);
%we are gonna only consider X values with Hamming-distance zero or one
```

```
for i = 1:degenY
    codon = codonsY{i};

    tmp = tmp + q(codon(1))*q(codon(2))*q(codon(3))*M(codon(1), codon(1))*M(codon(2), codon(2))*M(codon(3), codon(3));

    for basis = 1:3
        for j = 1:4
            initCn        = codon;
            initCn(basis) = j;
            if isempty(find(initCn - codon))
                continue;
            end

            newTerm = q(initCn(1))*q(initCn(2))*q(initCn(3))...
                      *biases(basis)*M(initCn(1), codon(1))*M(initCn(2), codon(2))*M(initCn(3), codon(3));
            if lookuptable(initCn(1), initCn(2), initCn(3)) ~= ' '
                tmp = tmp + newTerm;
            else
                tmp = tmp + stopsIncluded*newTerm;
            end
        end
    end
end

if isequal(tmp, sym(0))
    error('Y amino acid unknown!')
    return
end;
probAY = tmp;



function probJointA = probJointA(X, Y)
%return the joint probability of
%amino acids, for example X = 'L', Y = 'F'
%The probability that the pair of amino acids a,a' is found is:
%p(a',a) = sum_(all rst meaning a' and all ijk meaning a) p(rst,ijk)
global lookuptable;
global q;
global M;
global biases;
global stopsIncluded;

tmp = sym(0);
%codonsOfX = sym(0);
%codonsOfY = sym(0);

codonsX = listCodons(X);
degenX  = length(codonsX);
codonsY = listCodons(Y);
degenY  = length(codonsY);

for i = 1:degenX
   codonX = codonsX{i};
   for j = 1:degenY
       codonY   = codonsY{j};
       diffCdns = codonX-codonY;
       basisBiasFactor = prod(biases(find(diffCdns)));
       if isempty(find(diffCdns))
           basisBiasFactor = 1;
       end
       %  this is an alternative and unequivalent method:
       tmp = tmp + q(codonX(1))*q(codonX(2))*q(codonX(3))...
                   *basisBiasFactor...
```

```
                *M(codonX(1), codonY(1))...
                *M(codonX(2), codonY(2))...
                *M(codonX(3), codonY(3));

      %tmp = tmp +  M(codonsX{i}(1), codonsY{j}(1))...
      %             *M(codonsX{i}(2), codonsY{j}(2))...
      %             *M(codonsX{i}(3), codonsY{j}(3));
   end
end
%there seems to be a wierd order flipping that is mathematically neutral but unexplained

%probJointA = tmp*probAX(X);
probJointA = tmp;
```

C.4 ts.method.mod

```
# ------------------------------------------------------------
# Copyright 2005, Alexander Gutfraind.  Free academic use.
#
# the TS model
# ------------------------------------------------------------

var qu >=0, <=1;
var qc >=0, <=1;
var qa >=0, <=1;
var qg >=0, <=1;

var muu >= 0.00001, <= 1;
var mcu >= 0.00001, <= 1;
var mau >= 0.00001, <= 1;
var mgu >= 0.00001, <= 1;

var muc >= 0.00001, <= 1;
var mcc >= 0.00001, <= 1;
var mac >= 0.00001, <= 1;
var mgc >= 0.00001, <= 1;

var mua >= 0.00001, <= 1;
var mca >= 0.00001, <= 1;
var maa >= 0.00001, <= 1;
var mga >= 0.00001, <= 1;

var mug >= 0.00001, <= 1;
var mcg >= 0.00001, <= 1;
var mag >= 0.00001, <= 1;
var mgg >= 0.00001, <= 1;

var sumQ = qu + qc + qa + qg;
var sumP = muu + mcu + mau + mgu +  muc + mcc + mac + mgc +  mua + mca + maa + mga +  mug + mcg + mag + mgg;
#var sumErrors = mcu + mau + mgu +  muc +       mac + mgc +  mua + mca +       mga +  mug + mcg + mag      ;  param epsilonQ = 0;
var sumErrors = qu*(    muc+mua+mug) +qc*(mcu    +mca+mcg)+ qa*(mau+mac    +mag) + qg*(mgu+mgc+mga    );  param epsilonQ = 1;

var u_normal = muu+muc+mua+mug;
var c_normal = mcu+mcc+mca+mcg;
var a_normal = mau+mac+maa+mag;
var g_normal = mgu+mgc+mga+mgg;



param codeType := "standard_code" symbolic;
```

```
var mucTolerance = 339*pos2*qc*qa+535*pos2*qc*qg+1000*pos3*qc^2+389*pos2*qg^2+296*pos1*qu^2+152*pos2*qu^2+1000*pos3*qu^2+2000*pos3*qc*qa+545*pos1*qu*qc+298*pos2*qu*qc+
var mcuTolerance = 456*pos2*qc*qa+577*pos2*qc*qg+1000*pos3*qc^2+319*pos2*qg^2+336*pos1*qu^2+334*pos2*qu^2+1000*pos3*qu^2+2000*pos3*qc*qa+728*pos1*qu*qc+592*pos2*qu*qc+
var muaTolerance = 422*pos2*qc*qa+478*pos2*qc*qg+1000*pos3*qc^2+192*pos2*qg^2+181*pos1*qu^2+332*pos2*qu^2+296*pos3*qu^2+1301*pos3*qc*qa+662*pos1*qu*qc+620*pos2*qu*qc+1
var mauTolerance = 575*pos2*qc*qa+566*pos2*qc*qg+1000*pos3*qc^2+351*pos2*qg^2+331*pos1*qu^2+360*pos2*qu^2+336*pos3*qu^2+1396*pos3*qc*qa+739*pos1*qu*qc+724*pos2*qu*qc+1
var mugTolerance = 381*pos2*qc*qa+454*pos2*qc*qg+1000*pos3*qc^2+269*pos2*qg^2+291*pos1*qu^2+176*pos2*qu^2+296*pos3*qu^2+1301*pos3*qc*qa+781*pos1*qu*qc+361*pos2*qu*qc+1
var mguTolerance = 512*pos2*qc*qa+450*pos2*qc*qg+1000*pos3*qc^2+208*pos2*qg^2+207*pos1*qu^2+349*pos2*qu^2+336*pos3*qu^2+1396*pos3*qc*qa+591*pos1*qu*qc+591*pos2*qu*qc+1
var mcaTolerance = 624*pos2*qc*qa+577*pos2*qc*qg+1000*pos3*qc^2+275*pos2*qg^2+301*pos1*qu^2+294*pos2*qu^2+296*pos3*qu^2+1301*pos3*qc*qa+587*pos1*qu*qc+663*pos2*qu*qc+1
var macTolerance = 541*pos2*qc*qa+505*pos2*qc*qg+1000*pos3*qc^2+520*pos2*qg^2+417*pos1*qu^2+173*pos2*qu^2+336*pos3*qu^2+1396*pos3*qc*qa+581*pos1*qu*qc+393*pos2*qu*qc+1
var mcgTolerance = 662*pos2*qc*qa+641*pos2*qc*qg+1000*pos3*qc^2+387*pos2*qg^2+275*pos1*qu^2+373*pos2*qu^2+296*pos3*qu^2+1301*pos3*qc*qa+729*pos1*qu*qc+627*pos2*qu*qc+1
var mgcTolerance = 626*pos2*qc*qa+514*pos2*qc*qg+1000*pos3*qc^2+369*pos2*qg^2+333*pos1*qu^2+258*pos2*qu^2+336*pos3*qu^2+1396*pos3*qc*qa+576*pos1*qu*qc+403*pos2*qu*qc+1
var magTolerance = 721*pos2*qc*qa+630*pos2*qc*qg+1000*pos3*qc^2+407*pos2*qg^2+494*pos1*qu^2+142*pos2*qu^2+1000*pos3*qu^2+2000*pos3*qc*qa+896*pos1*qu*qc+417*pos2*qu*qc+
var mgaTolerance = 678*pos2*qc*qa+476*pos2*qc*qg+1000*pos3*qc^2+206*pos2*qg^2+537*pos1*qu^2+349*pos2*qu^2+1000*pos3*qu^2+2000*pos3*qc*qa+849*pos1*qu*qc+612*pos2*qu*qc+

var QuSafety = (              0 + muc*mucTolerance + mua*muaTolerance + mug*mugTolerance)/(0   + muc + mua + mug);
var QcSafety = (mcu*mcuTolerance +               0 + mca*mcaTolerance + mcg*mcgTolerance)/(mcu + 0   + mca + mcg);
var QaSafety = (mau*mauTolerance + mac*macTolerance +               0 + mag*magTolerance)/(mau + mac + 0   + mag);
var QgSafety = (mgu*mguTolerance + mgc*mgcTolerance + mga*mgaTolerance +               0)/(mgu + mgc + mga + 0  );

#product function
maximize dummyFn: 1;

#subject to eqnQu:
subject to eqnQc: Fq*(qu - qc) = QuSafety-QcSafety;
subject to eqnQa: Fq*(qu - qa) = QuSafety-QaSafety;
subject to eqnQg: Fq*(qu - qg) = QuSafety-QgSafety;


#
#subject to eqnMcu:
subject to eqnMau: Fm*(qa*mau - qc*mcu)   = mauTolerance - mcuTolerance;
subject to eqnMgu: Fm*(qg*mgu - qc*mcu)   = mguTolerance - mcuTolerance;

subject to eqnMuc: Fm*(qu*muc - qc*mcu)   = mucTolerance - mcuTolerance;
#
subject to eqnMac: Fm*(qa*mac - qc*mcu)   = macTolerance - mcuTolerance;
subject to eqnMgc: Fm*(qg*mgc - qc*mcu)   = mgcTolerance - mcuTolerance;

subject to eqnMua: Fm*(qu*mua - qc*mcu)   = muaTolerance - mcuTolerance;
subject to eqnMca: Fm*(qc*mca - qc*mcu)   = mcaTolerance - mcuTolerance;
#
subject to eqnMga: Fm*(qg*mga - qc*mcu)   = mgaTolerance - mcuTolerance;

subject to eqnMug: Fm*(qu*mug - qc*mcu)   = mugTolerance - mcuTolerance;
subject to eqnMcg: Fm*(qc*mcg - qc*mcu)   = mcgTolerance - mcuTolerance;
subject to eqnMag: Fm*(qa*mag - qc*mcu)   = magTolerance - mcuTolerance;
#

subject to q_normalization: sumQ = 1;
subject to p_normalization: sumErrors = epsilon;

subject to u_normalization: u_normal = 1;
subject to c_normalization: c_normal = 1;
subject to a_normalization: a_normal = 1;
subject to g_normalization: g_normal = 1;
```

C.5 ts.method.oneRun.run

```
# ------------------------------------------------------------
# Copyright 2005, Alexander Gutfraind.  Free academic use.
```

```
#
# Script calls the TS models
# assumes the file mymodels\ts.method.mod exists
# and the directory mymodels\output exists
# ------------------------------------------------------------

reset;

#codon position biases in errors (uniform)
#param pos1 := 0.3333;
#param pos2 := 0.3333;
#param pos3 := 0.3333;
#codon position biases in errors (the HH bias)
param pos1 := 5/16;
param pos2 := 1/16;
param pos3 := 10/16;
#codon position biases in errors
#param pos1 := 0;
#param pos2 := 0;
#param pos3 := 1;

param epsilon  := 0.01;

#uniform bias
#param SxyMax := 550;
#param SxyMin := 400;
#param SxMax  := 600;
#param SxMin  := 400;

#the HH bias
param SxyMax := 750;
param SxyMin := 500;
param SxMax  := 750;
param SxMin  := 550;

param Fm := (SxyMax-SxyMin)   / ((epsilon/24));
#param Fq := (SxMax-SxMin)  /(0.25/2);
param Fq := 3651.484;

#param stopsIncluded  := 0;
param stopsIncluded  := 1;

param modName := "mymodels\ts_method.mod"  symbolic;
model (modName);
option solver minos;

#display _varname, _var;

# ------------------------------------------------------------

printf "linear qx and qxMxy eqns; %s code,\n\n\n", codeType > mymodels\output\TS_out.csv;
printf "model,, %s, \n", (modName) > mymodels\output\TS_out.csv;
printf "pos1, %f, \n", pos1 > mymodels\output\TS_out.csv;
printf "pos2, %f, \n", pos2 > mymodels\output\TS_out.csv;
printf "pos3, %f, \n", pos3 > mymodels\output\TS_out.csv;
printf "Fm, %f, \n", Fm > mymodels\output\TS_out.csv;
printf "Fq, %f, \n", Fq > mymodels\output\TS_out.csv;
printf "epsilon, %f, epslnIncldsQ(Yes/No), %d\n", epsilon, epsilonQ > mymodels\output\TS_out.csv;
printf "stopsIncluded, %f, \n", stopsIncluded > mymodels\output\TS_out.csv;
printf "iteration,pNorm,qNorm,dummyFn,infeas,,qu,qc,qa,qg,,quMuc,qcMcu,quMua,qaMau,quMug,qgMgu,qcMca,qaMac,qcMcg,qgMgc,qaMag,qgMga,,Muu,Mcc,Maa,Mgg,SUM\n"
  > mymodels\output\TS_start_out.csv;
printf "iteration,MSG,pNorm,qNorm,dummyFn,infeas,,qu,qc,qa,qg,GC,,quMuc,qcMcu,quMua,qaMau,quMug,qgMgu,qcMca,qaMac,qcMcg,qgMgc,qaMag,qgMga,,Muu,Mcc,Maa,Mgg,SUM\n"
  > mymodels\output\TS_out.csv;
```

```
param NAstart := 1;
param NAstop  := 200;
param NAstep  := 1;

var dummyQ{i in 1..4};
var dummyP{i in 1..12};
var S;
var infeas;

for {theta in NAstart .. NAstop by NAstep} {
    #---- Setup Neucleotides ----#
   let {i in 1..4} dummyQ[i] := Uniform01();
   let S := sum {i in 1..4} dummyQ[i];

   let qu := dummyQ[1] / S;
   let qc := dummyQ[2] / S;
   let qa := dummyQ[3] / S;
   let qg := dummyQ[4] / S;

   #---- Setup Mutations ----#
   let {i in 1..12} dummyP[i] := Uniform01();
   let S := 1/epsilon * sum{ i in 1..12} dummyP[i];

   let mcu  := dummyP[1] / S;
   let mau  := dummyP[2] / S;
   let mgu  := dummyP[3] / S;

   let muc  := dummyP[4] / S;
   let mac  := dummyP[5] / S;
   let mgc  := dummyP[6] / S;

   let mua  := dummyP[7] / S;
   let mca  := dummyP[8] / S;
   let mga  := dummyP[9] / S;

   let mug  := dummyP[10] / S;
   let mcg  := dummyP[11] / S;
   let mag  := dummyP[12] / S;

   let muu   := 1 - muc - mua - mug;
   let mcc   := 1 - mcu - mca - mcg;
   let maa   := 1 - mau - mac - mag;
   let mgg   := 1 - mgu - mgc - mga;

   printf "\nStarting iteration number: %d\n", theta;
   printf "\nStarting iteration number: %d\n", theta > mymodels\output\TS_out.txt;
   solve > mymodels\output\TS_out.txt;

   let infeas := abs(Fq*(qu - qc) - (QuSafety-QcSafety)) + abs(Fq*(qu - qa) - (QuSafety-QaSafety)) + abs(Fq*(qu - qg) - (QuSafety-QgSafety));
   let infeas := infeas + abs(Fm*(qa*mau - qc*mcu)  - (mauTolerance - mcuTolerance)) + abs(Fm*(qg*mgu - qc*mcu)  - (mguTolerance - mcuTolerance));
   let infeas := infeas + abs(Fm*(qu*muc - qc*mcu)  - (mucTolerance - mcuTolerance)) + abs(Fm*(qa*mac - qc*mcu)  - (macTolerance - mcuTolerance));
   let infeas := infeas + abs(Fm*(qg*mgc - qc*mcu)  - (mgcTolerance - mcuTolerance)) + abs(Fm*(qu*mua - qc*mcu)  - (muaTolerance - mcuTolerance));
   let infeas := infeas + abs(Fm*(qc*mca - qc*mcu)  - (mcaTolerance - mcuTolerance)) + abs(Fm*(qg*mga - qc*mcu)  - (mgaTolerance - mcuTolerance))
   let infeas := infeas + abs(Fm*(qu*mug - qc*mcu)  - (mugTolerance - mcuTolerance)) + abs(Fm*(qc*mcg - qc*mcu)  - (mcgTolerance - mcuTolerance))
   let infeas := infeas + abs(Fm*(qa*mag - qc*mcu)  - (magTolerance - mcuTolerance));
   let infeas := infeas + abs(sumQ - 1) + abs(sumErrors - epsilon) + abs(u_normal - 1) + abs(c_normal - 1) + abs(a_normal - 1) + abs(g_normal - 1);

   printf "\n%d,%s,%E,%E,%E,%E,,", theta,  substr(solve_message, 1, 25), sumP, sumQ, dummyFn, infeas > mymodels\output\TS_out.csv;
   printf "%E,%E,%E,%E,%E,,%E,%E,%E,%E, %E,%E,%E,%E, %E,%E,%E,%E,,%E,%E,%E,%E,%E,", qu,qc,qa,qg,qc+qg, qu*muc,qc*mcu,qu*mua,qa*mau,
              qu*mug,qg*mgu,qc*mca,qa*mac, qc*mcg,qg*mgc,qa*mag,qg*mga, muu,mcc,maa,mgg, muu+mcc+maa+mgg > mymodels\output\TS_out.csv;
}
printf "\n*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,*,"> mymodels\output\TS_out.csv;
```

Bibliography

[1] C. Alff-Steinberger. The genetic code and error transmission. *Proc. Natl. Acad. Sci. U. S. A.*, 64(2):584–591, 1969.

[2] D.H. Ardell. On error minimization in a sequential origin of the standard gnetic code. *J. Mol. Evol.*, 47(1):1–13, 1998.

[3] R.B. Ash. *Information Theory*. Dover, New York, 1990.

[4] F. Bagnoli and M. Bezzi. Eigen's error threshold and mutational meltdown in a quasispecies model. *International Journal of Modern Physics C*, 9:999–1005, 1998.

[5] K. Bebenek, C.M. Joyce, M.P. Fitzgerald, and T.A. Kunkel. The fidelity of DNA synthesis catalyzed by derivatives of escherichia coli DNA polymerase I. *J. Biol. Chem.*, 265(23):13878–87, 1990.

[6] S. A. Benner, A. Ricardo, and M. A. Carrigan. Is there a common chemical model for life in the universe? *Curr. Opin. Chem. Biol.*, 8:672–689, 2004.

[7] S.A. Benner. Expanding the genetic lexicon: incorporating non-standard amino acids into proteins by ribosome-based synthesis. *Trends Biotechnol.*, 12(5):158–63, 1994.

[8] S.A. Benner, M.A. Cohen, and G.H. Gonnet. Amino acid substitution during functionally constrained divergent evolution of protein sequences. *Protein Eng.*, 7(11):1323–32, 1994.

[9] R. J. Broker. *Genetics: Analysis and Principles*. Benjamin/Cummings, California, 1999.

[10] T.M Cover and J.A. Thomas. *Elements of Information Theory*. Wiley Interscience, New York, 1991.

[11] F. H. C. Crick. The origin of the genetic code. *J. Mol. Biol.*, 38:367–379, 1968.

[12] F. H. C. Crick, J. S. Griffith, and L. E. Orgel. Codes without commas. *Proc. Natl. Acad. Sci. U. S. A.*, 43:416–421, 1957.

[13] C. Darwin. *The Origin of Species*. Castle Books, New Jersey, 2004.

[14] R. Dawkins. *The Blind Watchmaker*. Longman Scientific and Technical, Essex, 1986.

[15] M. Di Giulio. The late stage of genetic code structuring took place at a high temperature. *Gene*, 261:189–195, 2000.

[16] M. Di Giulio. The universal ancestor was a thermophile or a hyperthermophile: Tests and further evidence. *J. Theor. Biol.*, 221:425–436, 2003.

[17] M. Di Giulio and M. Medugno. The level and landscape of optimization in the origin of the genetic code. *J. Mol. Evol.*, 52:372–382, 2001.

[18] M. Eigen. Error catastrophe and antiviral strategy. *Proc. Natl. Acad. Sci. U. S. A.*, 99:13374–13376, 2002.

[19] M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self-Organization*. Springer-Verlag, New York, 1979.

[20] W. H. Elliott and D. C. Elliott. *Biochemistry and Molecular Biology*. Oxford University Press, USA, 2nd paperback edition, 2001.

[21] P. Forterre and H. Philippe. Where is the root of the universal tree of life? *BioEssays*, 21:871–879, 1999.

[22] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Cole Publishing Company, California, 2002.

[23] S.J. Freeland and L.D. Hurst. The genetic code is one in a million. *J. Mol. Evol.*, 47:238–248, 1998.

[24] S.J. Freeland, R.D. Knight, and L.F. Landweber. Do proteins predate DNA? *Science*, 286(5440):690–692, 1999.

[25] S.J. Freeland, T. Wu, and N. Keulmann. The case for an error minimizing standard genetic code. *Orig. Life Evol. Bios.*, 33:457–477, 2003.

[26] S. Freeman and J.C. Herron. *Evolutionary Analysis*. Prentice Hall, New Jersey, 2nd edition, 2001.

[27] N. Galtier and J.R. Lobry. Relationships between genomic G+C content, RNA secondary structures, and optimal growth temperature in prokaryotes. *J. Mol. Evol.*, 44:632–636, 1997.

[28] P. P. Gardner, B. R. Holland, V. Moulton, M. Hendy, and D. Penny. Optimal alphabets for an RNA world. *Proc. R. Soc. Lond. B*, 270:1177–1182, 2003.

[29] R. Gil, F.J. Silva, J. Peret, and A. Moya. Determination of the core of a minimal bacterial gene set. *Microbiology and Molecular Biology Reviews*, 68(3):518–537, 2004.

[30] W. Gilbert. The RNA world. *Nature*, 319:618, 1986.

[31] D. Gilis, S. Massar, and M. Rooman. Optimality of the genetic code with respect to protein stability and amino-acid frequencies. *Genome Biol.*, 2(11):research0049, 2001.

[32] S.W. Golomb. Efficient coding for the desoxyribonucleic channel. In *Mathematical Problems in the Biological Sciences*, volume 14 of *Proceedings of Symposia in Applied Mathematics*, pages 87–100, Rhode Island, 1962. American Mathematical Society.

[33] R. Grantham. Amino acid difference formulae to help explain protein evolution. *Science*, 185:862–864, 1974.

[34] A. Gutfraind and A. Kempf. Structure of genetic code indicates non-thermophile origin. *Submitted*, 2006.

[35] D. Haig and L.D. Hurst. A quantitative measure of error minimization in the genetic code. *J. Mol. Evol.*, 33:412–417, 1991.

[36] R.W. Hamming. *Coding and Information Theory*. Prentice-Hall, New Jersey, 1980.

[37] B. Hayes. The invention of the genetic code. *American Scientist*, 86(1):8–14, May 1998.

[38] P.G. Higgs and G. Woodcock. The accumulation of mutations in asexual populations and the structure of genealogical trees in the presence of selection. *J. Math. Biol.*, 33(7):677–702, 1995.

[39] S.P. Jackson. Sensing and repairing DNA double-strand breaks. *Carcinogenesis*, 23(5):687–696, 2002.

[40] D.C. Jeffares, A.M. Poole, and D. Penny. Relics from the RNA world. *J. Mol. Evol.*, 46(1):18–36, 1998.

[41] J.-L. Jestin and A. Kempf. Chain termination codons and polymerase-induced frameshift mutations. *FEBS Lett.*, 419:153–156, 1997.

[42] O.P. Judson and D. Haydon. The genetic code: what is it good for? an analysis of the effects of selection pressures on genetic codes. *J. Mol. Evol.*, 49(5):539–550, 1999.

[43] S. Kazakov and S. Altman. A trinucleotide can promote metal ion-dependent specific cleavage of RNA. *Proc. Natl. Acad. Sci. U. S. A.*, 89:7939–7943, 1992.

[44] R.D. Knight, S.J. Freeland, and L.F. Landweber. Selection, history and chemistry: the three faces of the genetic code. *Trends Biochem. Sci.*, 24(6):241–247, Jun 1999.

[45] R.D. Knight, S.J. Freeland, and L.F. Landweber. Rewiring the keyboard: Evolvability of the genetic code. *Nat. Rev. Genet.*, 2:49–58, 2001.

[46] R.D. Knight, S.J. Freeland, and L.F. Landweber. A simple model based on mutation and selection explains trends in codon and amino-acid usage and GC composition within and across genomes. *Genome Biol.*, 2(4):research0010, 2001.

[47] R.D. Knight, L.F. Landweber, and M. Yarus. How mitochondria redefine the code. *J. Mol. Evol.*, 53:299–313, 2001.

[48] J. Kåhre. *The Mathematical Theory of Information*. Kluwer Academic, Massachusetts, 2002.

[49] A. Kun, M. Santos, and E. Szathmary. Real ribozymes suggest a relaxed error threshold. *Nature Genetics*, 37:1008–1011, 2005.

[50] P. Li and N. Goldman. Models of molecular evolution and phylogeny - a review. *Genome Res.*, 8(12):1233–1244, 1998.

[51] W.-H. Li. *Molecular Evolution*. Sinauer Associates, Inc., Massachusetts, 1996.

[52] D.A. Mac Dónaill. Why nature chose A, C, G and U/T: An error-coding perspective of nucleotide alphabet composition. *Orig. Life Evol. Biol.*, 33:433–455, 2003.

[53] S.L. Miller and A. Lazcano. The origin of life–did it occur at high temperatures? *J. Mol. Evol.*, 41:689–692, 1995.

[54] B. G. Mirkin, T. I. Fenner, M. Y. Galperin, and E. V. Koonin. Algorithms for computing parsimonious evolutionary scenarios for genome evolution, the last universal common ancestor and dominance of horizontal gene transfer in the evolution of prokaryotes. *BMC Evol. Biol.*, 3(2):1–34, 2003.

[55] D. Mitchell and Bridge R. A test of chargaff's second rule. *Biochem. Biophys. Res. Commun.*, 340(1):90–4, 2005.

[56] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Massachusetts, 1996.

[57] S.J. Mojzsis, T.M. Harrison, and R.T. Pidgeon. Oxygen-isotope evidence from ancient zircons for liquid water at the earth's surface 4,300 Myr ago. *Biochem. Biophys. Res. Commun.*, 409(6817):178–181, 2001.

[58] J. Mynard Smith and E. Szathmary. *The Major Transitions in Evolution*. W.H. Freeman, New York, 1995.

[59] P. Nissen, J. Hansen, N. Ban, P.B. Moore, and T.A. Steitz. The structural basis of ribosome activity in peptide bond synthesis. *Biochem. Biophys. Res. Commun.*, 289(5481):920–930, 2000.

[60] S. Osawa and T. H. Jukes. Evolution of the genetic code as affected by anticodon content. *Trends Genet.*, 4:191–198, 1988.

[61] R. Ota and D. Penny. Estimating changes in mutational mechanisms of evolution. *J. Mol. Evol.*, 57(S):233–240, 2003.

[62] N. R. Pace. A molecular view of microbial diversity and the biosphere. *Science*, 276:734–40, 1997.

[63] J. Parker. Errors and alternatives in reading the universal genetic code. *Microbiol. Rev.*, 53(3):273–298, 1989.

[64] D. Penny, M. D. Hendy, and A. M. Poole. Testing fundamental evolutionary hypotheses. *J. Theor. Biol.*, 223(3):377–385, 2003.

[65] A.L. Peressini, F.E. Sullivan, and J.J. Uhl Jr. *The Mathematics of Nonlinear Programming*. Springer-Verlag, New York, 1988.

[66] A.M. Poole. Modern mRNA proofreading and repair: clues that the last universal common ancestor possessed an RNA genome? *Mol. Biol. Evol.*, 22(6):1444–1455, 2005.

[67] A.M. Poole, D. Jeffares, and D. Penny. Early evolution: prokaryotes, the new kids on the block. *BioEssays*, 21:880–889, 1999.

[68] A.M. Poole, D. Penny, and B. M. Sjberg. Methyl-RNA: an evolutionary bridge between RNA and DNA? *Chem. Biol.*, 7:R207–R216, 2000.

[69] J. Pressing and D.C. Reanney. Divided genomes and instrinsic noise. *J. Mol. Evol.*, 20:135–146, 1984.

[70] D.C. Reanney. Genetic error and genome design. *Trends in Genetics*, 2(2):41–46, 1986.

[71] D.C. Reanney and Pressing J. Temperature as a determinative factor in the evolution of genetic systems. *J. Mol. Evol.*, 21:72–75, 1984.

[72] J.W. Schopf. Microfossils of the early archean apex chert - new evidence of the antiquity of life. *Science*, 260(5108):640–646, 1993.

[73] D. W. Schultz and M. Yarus. Transfer RNA mutation and the malleability of the genetic code. *J. Mol. Biol.*, 235:1377–1380, 1994.

[74] D.W. Schwartzman and C.H. Lineweaver. The hyperthermophilic origin of life revisited. *Biochem. Soc. Trans.*, 32(2):168–71, 2004.

[75] S. Sengupta and P.G. Higgs. A unified model of codon reassignment in alternative genetic codes. *Genetics*, 170(2):831–840, 2005.

[76] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois, Illinois, 1963.

[77] N.K. Sinha and M.D. Haimes. Molecular mechanisms of substitution mutagenesis. an experimental test of the Watson-Crick and Topal-Fresco models of base mispairings. *J. Biol. Chem.*, 256(20):10671–83, 1981.

[78] R.L. Sinsheimer. Is the nucleic acid message in a two-symbol code? *J. Mol. Biol.*, 1:218–220, 1959.

[79] T. M. Sonneborn. *Degeneracy of the genetic code: extent, nature, and genetic implications in Bryson, V. and Vogel, H.K. Evolving Genes and Proteins*. Academic Press Inc., New York, 1965.

[80] R. Swanson. Unifying concept for the amino acid code. *Bull. Math. Biol.*, 46(2):187–203, 1984.

[81] E. Szathmary. What is the optimal size for the genetic alphabet? *Proc. Natl. Acad. Sci. U.S.A.*, 89:2614–2618, 1992.

[82] E. Szathmary. Coding coenzyme handles: A hypothesis for the origin of the genetic code. *Proc. Natl. Acad. Sci. U.S.A.*, 90:9916–9920, 1993.

[83] E. Szathmary. The origin of the genetic code: amino acids as cofactors in an RNA world. *Trends Genet.*, 15(6):223–9, June 1999.

[84] E. Szathmary. Why are there four letters in the genetic alphabet? *Nat. Rev. Genet.*, 4(12):995–1001, 2003.

[85] R. Togneri and C.J.S. deSilva. *Fundamentals of information theory and coding theory.* Chapman and Hall/CRC, Florida, 2002.

[86] M.D. Topal and J.R. Fresco. Complementary base pairing and the origin of substitution mutations. *Nature*, 263:285–289, 1976.

[87] H.-c. Wang and D.A. Hickey. Evidence for strong selective constraint acting on the nucleotide composition of 16S ribosomal RNA genes. *Nuc. Acids Res.*, 30(11):2501–2507, 2002.

[88] C. R. Woese. On the evolution of the genetic code. *Proc. Nat. Acad. Sci. USA*, 54:1546–1552, 1965.

[89] J. T.-F. Wong. A co-evolution theory of the genetic code. *Proc. Nat. Acad. Sci. USA*, 72(5):1909–1912, 1975.

[90] L.Y. Yampolsky and A. Stoltzfus. The exchangeability of amino acids in proteins. *Genetics*, 170(4):1459–1472, 2005.

[91] H.P. Yockey. *Information Theory, Evolution, and the Origin of Life.* Cambridge University Press, Cambridge, 2005.

[92] W. Zhu and S.J. Freeland. The standard genetic code enhances adaptive evolution of proteins. *J. Theor. Biol.*, 239(1):63–70, 2006.