**MASTER IN COMPUTER SCIENCE**

# RNA Sequence Optimizations for Efficient Heterologous Protein Expression: A Pipeline Using a Genetic Algorithm and a Simulated Annealing Approach

MESTRÉ, Michaël

*Award date:*
2023

*Awarding institution:*
University of Namur

Link to publication

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2022-2023

# RNA Sequence Optimizations for Efficient Heterologous Protein Expression: A Pipeline Using a Genetic Algorithm and a Simulated Annealing Approach

Mestré Michaël

........................ (Signature pour approbation du dépôt - REE art. 40)

Promoteur : Jacquet Jean-Marie

Mémoire présenté en vue de l'obtention du grade de Master 60 en Sciences Informatiques

Faculté d'Informatique – Université de Namur

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

# Acknowledgments

First and foremost, I would like to thank my supervisor, Mr. Jacquet, for his patience and the time he granted me to complete this project. I am also grateful for his attentive proofreading and constructive feedback, which helped me progress in my writing.

I would also like to express my sincere appreciation to the professors and mentors at the Faculty of Computer Science in Namur for their guidance and support throughout my studies. Their expertise and dedication have been invaluable in shaping my understanding of the field.

I would like to extend my thanks to the administrative staff and secretaries at the Faculty of Computer Science in Namur for their kind assistance and support. Their help with administrative procedures, scheduling, and other logistical matters has been instrumental in making my studies run smoothly.

I would like to extend my thanks to Grégory Mathy and Marianne Dewerchin for proposing the subject that allowed me to combine my newfound knowledge in computer science with my original field of expertise in biology. Their initial explanations helped me understand the relevance of this research and what it could bring.

Finally, I am deeply grateful to my family for supporting me during these past years. Pursuing studies at the age of 35 while already working is not an easy task and has a significant impact on family life. I would like to thank my partner, Frédérique, for taking on more family responsibilities and my children, Léa and Rémy, who had to share their dad a lot these past years.

# Abstract

In the context of vaccine development, it is common to consider different strategies, including that of attempting to express certain sub-parts of the virus made up of proteins in a heterologous cellular host. The latter is generally chosen for its ease of maintenance in cell culture, which is the case of the E coli bacterium. But transposing a coding sequence from one organism to another is far from trivial and several problems can occur such as the choice of different synonymous codons, the introduction of alternative ribosome binding sites, or a secondary structure of the messenger RNA preventing ribosome pairing. All these elements can lead to the synthesis of a non-functional protein. The present work proposes a sequence design solution using a pipeline of different modules to satisfy different constraints when transposing the expression of a protein from an original host to a heterologous host. Due to the complexity and the number of possibilities, a manual search is not feasible, nor is an exhaustive computer search. Other strategies such as a genetic algorithm or simulated annealing must be considered to try to obtain an optimal solution satisfying the different constraints in a reasonable time.

Dans le cadre du dévelopement de vaccins il est courant d'envisager différentes stratégies dont celle de tenter d'exprimer certaines sous parties de virus constituées de protéines dans un hôte cellulaire hétérologue. Ce dernier est généralement choisi pour sa facilité de maintenance en culture cellulaire, ce qui est le cas de la bactérie E coli. Mais transposer une séquence d'un codante d'un organisme a un autre est loin d'être trivial et plusieurs problèmes peuvent survenir comme le choix de codons synonymes différents, l'introduction de site de fixation du ribosome alternatif, ou une structure secondaire de l'ARN messager empechant l'appariement du ribosome. Tous ces éléments peuvent conduire à la synthèse d'une proteine non fonctionnelle. Le présent travail propose une solution de design de séquence à l'aide d'un pipeline de différents modules permettant de satisfaire différentes contraintes lorsqu'il s'agit de transposer l'expression d'une protéine depuis un hôte d'origine vers un hôte hétérologue. De par la complexité et le nombre de possibilités, une recherche manuelle n'est pas envisageable, de même qu'une recherche exhaustive de manière informatique. D'autres stratégies telles qu'un algorithme génétique ou un recuit simulé doivent être envisagées pour tenter d'obtenir une solution optimale satisfiant les différentes contraintes en un temps raisonnable.

# Contents

# Chapter 1

# Introduction

## 1.1 Context

In today's world, the importance of vaccination cannot be overstated, as it has been instrumental in reducing the incidence of major diseases. While improvements in housing and nutrition were the primary drivers of the decline in infant mortality from infectious diseases prior to the development of vaccines, vaccination has played a significant role in enhancing human well-being. The history of vaccination can be traced back to Edward Jenner's experiments in 1796, where he used cowpox instead of smallpox to stimulate an immune response. This replaced the earlier practice of variolation, in which live smallpox viruses were inoculated into individuals, which posed a high risk of severe disease. The success of Jenner's experiment laid the foundation for subsequent research on attenuation, or the reduction of a virus's ability to cause illness. This principle has been the driving force behind all subsequent vaccine research, and it has culminated in the official eradication of smallpox in 1979 [1]. The attenuated virus is still capable of inducing an immune response that can recognize and protect individuals from the live virus.

The attenuation concept has evolved over the past three centuries from an empirical approach to a more rational design since the mid-20th century. This shift has been driven by a better understanding of the immunological system and the ability of scientists to distinguish between an antibody-mediated response and a lymphocyte-mediated response [2].

The ability to cultivate cells *in vitro* using advanced techniques has been a crucial development that has shifted the paradigm of vaccine development towards a more rational approach. By culturing cells, scientists can now select the best cellular clones for the replication of viruses that lack their ability to infect humans [3].

Since the 1970s, advances in molecular biology have led to the development of recombinant expression-systems, such as bacteria, for the production of vaccines. These systems are genetically modified to express proteins, specifically subunits of the target virus, which they would not normally produce. This approach is considered to be cheaper and safer than producing the whole virus. However, the immunological response to subunit vaccines is often lower, requiring additional strategies to enhance it, such as the use of adjuvants. Selecting the appropriate subunits of the virus to induce the desired immune response can be challenging, requiring careful consideration of amino acid sequences. This process may involve a significant number of experiments[4].

Regarding the production of virus subunits, the initial expression system utilized is the bacterium *Escherichia coli* (*E. coli*) due to its simplicity and cost-effectiveness. However, more complex and costly cell types such as yeasts, insects, mammalian or plant cells are used if certain post-translational mechanisms are absent in *E. coli*, or if the protein formed is rapidly degraded or insoluble. The translation mechanism from the messenger ribonucleic acid (mRNA) to amino acid (AA) sequence also varies from one cell type to another, which can lead to non-functional proteins. This is particularly problematic since inadequate folding, which is mainly responsible for the protein's properties and function, can arise if some slow-folding areas of the protein do not fold properly [5]. In the context of vaccines, a protein must mimic a viral subunit to induce the expected immune response. Therefore, any alteration in the translation speed, such as the synthesis of truncated proteins due to the RNA sequence's translation not starting at the beginning of the coding region, must also be taken into account to ensure the protein's functionality.

The purpose of our work is to explore how to optimize an RNA sequence encoding a protein in its original host to be used by the bacterium *E. coli* to produce the same protein. The goal is to express the desired protein in the same form as the original host and maximize its production. However, these processes involve complex data structures that are challenging to process manually, so computational modeling tools are necessary. Therefore, this study will conduct an inventory of the various tools available to optimize an mRNA sequence for expression in a heterologous host. Optimization parameters, including codon usage bias, will be examined, and different strategies for each will be evaluated. Additionally, this work will investigate other critical parameters, such as the introduction of ribosome binding sites or larger mRNA secondary structures, that are essential for functional protein synthesis in large quantities. Existing bioinformatics tools will be assessed to determine whether they can be incorporated into optimization strategies. It will be necessary to explore and implement these different strategies governed by multiple constraints. The ultimate goal is to develop a solution that allows users to generate a sequence with the highest probability of being translated into a protein of the desired form by a heterologous host (and more precisely *E coli*).

## 1.2 Content

The first chapter will present the biological context (Chapter 2) of the production of heterologous proteins, the general mechanisms of protein synthesis will be briefly explained, including transcription and translation. This will provide a necessary basic understanding for the subsequent discussion. Although some of these concepts may seem extraneous at first, they will become relevant in the proposed solution to the problem at hand. The factors determining the use of *E coli* as an expression host will then be briefly described. Codon usage bias is a crucial mechanism for transposing sequences to a non-native host. It involves the variation in the frequency of nucleotide triplets (a codon) that code for the same amino acid and can occur within a species, within a cell based on the expressed gene, or between different species. To evaluate the impact of codon usage bias on protein synthesis, a literature review will be performed. Analyzing the various protein translation mechanisms can assist in determining the crucial factors when transferring a nucleotide sequence from one cellular host to another, specifically within the *E. coli* bacterium. The identified parameters can then be incorporated into a computer model for further analysis and optimizations.

The next chapter will present an overview of the IT tools documented in the literature (Chapter 3), and conduct a critical analysis to identify which ones are suitable for addressing the various constraints discussed in the chapter on biological context. The focus will be on the tools available for measuring and adjusting sequences based on codon usage bias, as it is believed to impact protein expression and conformation. However, these algorithms may have unintended consequences, such as the creation of alternative ribosome binding sites or RNA secondary structures, as discussed in the biological context chapter. Therefore, this study will also comprehensively examine algorithms used to predict the secondary structure of RNA strands.

This study will center on the mRNA sequence as it is the initial factor that can impact protein production. mRNA is no longer viewed solely as a relay of information, but also plays a role in regulating and ensuring the correct structure of proteins. Although other factors are involved, if the translation step is not efficiently optimized, subsequent processes will likely begin on an unfavorable note. The aim of this research is to manipulate specific parameters for transposing a sequence from a source host to a target host, as modifications are evidently necessary. The subsequent elements will be the main focus of this study:

- A modification of the composition in synonymous codons taking into account the specificities of the host is necessary. Indeed, depending on the organism there are preferential codons coding for the same amino acid. Many algorithms exist but the current trend is that one should try to harmonize the codon usage bias of the target host with that of the originating host.

- It is necessary to avoid a secondary mRNA structure (folds, hairpins) at the beginning of the sequence in order to allow a good pairing of the ribosome (which can be considered as the reading head of the RNA) with the strand.

- It is necessary to favor codons less frequently used at the beginning of the sequence to slow down the ribosomes progression at the beginning, which makes it possible to avoid traffic jams further on. These traffic jams having a negative effect on the lifespan of the mRNA (which will therefore be translated fewer times) and on the reading by the ribosome since if it remains stopped for too long it will end up being detached, which will lead to the production of a truncated protein.

- A modification of the coding sequence could generate alternative ribosome binding sites within the sequence. If these binding sites are some codons upstream of a start codon (AUG for methionine), it may result in ribosome pairing and start of translation which would not be the actual starting point of the protein. This would result in truncated proteins. It is therefore advisable to check that such types of sequences are not present.

Some of these factors may contradict each other, so it will not always be possible to satisfy all of them. It will be necessary to allocate a higher weight to some of them and to check the validity of the generated sequence when all these steps have been executed.

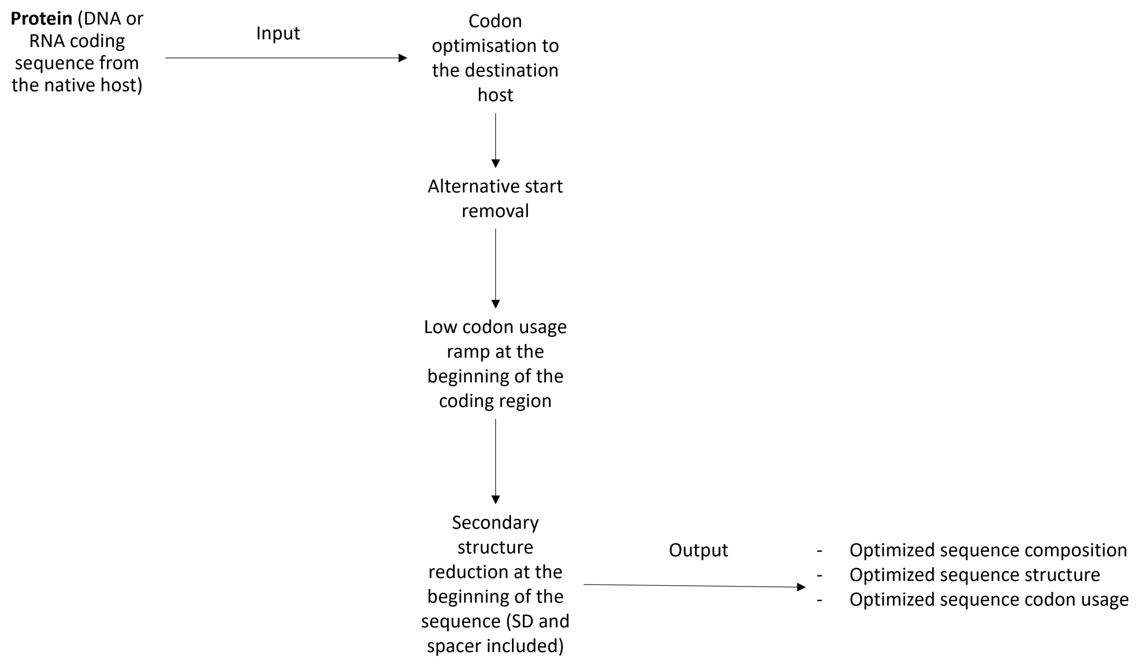

**Protein** (DNA or RNA coding sequence from the native host)
Input
Codon optimisation to the destination host
Alternative start removal
Low codon usage ramp at the beginning of the coding region
Secondary structure reduction at the beginning of the sequence (SD and spacer included)
Output
- Optimized sequence composition
- Optimized sequence structure
- Optimized sequence codon usage

Figure 1.1: Simplified execution process flow.

For many years, the field of computer science has been assisting biology in problem-solving. This case study aims to design an mRNA sequence that can enable the synthesis of a protein from one host in another, based on certain constraints mentioned earlier. Based on the review of the different algorithms, some of them will be retained and new ones will be created to meet the above-mentioned criteria. They will then be assembled into a complete solution (Chapter 4) to make it possible for a non computer scientist to use. This assembly will nevertheless be built in a modular way in order to be able to evaluate different possibilities and allow for future modifications.

A diagram of the execution plan is proposed in Figure 1.1. The algorithm takes a nucleotide sequence that encodes a protein from its original host as input. It then undergoes several steps, including codon optimization for the destination host, removal of any possible alternative start sites, introduction of a ramp of low usage codons at the sequence's beginning, and reduction of secondary structure at the sequence's start. The algorithm runs automatically without user intervention between steps and does not provide any intermediate output. Ultimately, the optimized sequence for the destination host is generated as output, along with information characterizing it, such as an assessment of its secondary structure and a graph comparing the native sequence with the generated sequence in terms of codon usage. This is a very abstract vision at first which will be more developed when the choice of concrete elements to solve the problem have been identified.

# Chapter 2

# Biological context

## 2.1 General mechanism of protein syntheis

Given that this research focuses on the production of proteins using a foreign cellular host, this chapter will provide an overview of the general mechanism of protein synthesis and their structures. As certain steps may differ depending on the cell type, the distinction between eukaryotic and prokaryotic families will be made. Eukaryotic cells have a nucleus and other organelles in their cytoplasm, while prokaryotic cells lack a true nucleus and distinct organelles. Therefore, the focus of this study will be on prokaryotic cells, as the primary goal is to investigate protein expression in this cell type.

The process of protein synthesis within a cell occurs through the transcription of DNA into RNA, which is then translated into amino acids, forming peptides, and ultimately proteins. While other protein production mechanisms do exist, they will not be addressed in this study, as they are beyond the scope of this research. This process is described in greater detail in the subsequent sections and draws heavily upon the Nature eBook: "Essentials of Cell Biology" [6], except for certain advanced topics explicitly cited.

### 2.1.1 Transcription from DNA to RNA

The process of protein synthesis originates from the genetic code of the cell, which is encoded in its DNA. Despite the distinct roles of specialized cells, all cells in a multicellular organism possess the same genetic code. Depending on the cell's context and specialization, certain parts of the genome are expressed. The DNA consists of four basic elements called deoxyribonucleic acids: adenine (A), thymine (T), guanine (G), and cytosine (C). These building blocks combine to form a filament that pairs with a complementary filament to create the double-stranded helix of DNA, as proposed by Watson and Crick in 1953 [7]. The structure of DNA is illustrated in Figure 2.1. Pairing between these elements can only occur among complementary bases, where A only pairs with T, and G only pairs with C. From a computer science perspective, DNA can be considered as a double string of complementary characters consisting solely of the letters A, T, G, and C, without taking into account the various molecular interactions that may occur among these elements.

The cells of the same organism all contain the same DNA sequence, but depending on their specialization and in response to the stimuli that surround them, these will transcribe only certain parts of their genome into RNA. The DNA double helix is opened and made accessible to an enzyme: the RNA polymerase whose role is to generate an RNA strand complementary to one of the DNA strands. In eucaryotes, there are three different types of RNA polymerase [9], contrary to procaryotes where there is only one responsible of the transcription into all RNAs.

In the **initiation** step, the RNA polymerase binds to a specific region of the DNA called the promoter which is a specific nucleotides sequence.

During the **elongation** step, RNA polymerase continues to read the DNA strand in the 3'-5' direction and matches the corresponding nucleotides. This has the effect of creating a complementary RNA sequence in the 5'-3' direction (Figure 2.2).

RNA differs from DNA in that it is single stranded and also because uracil (U) replaces thymine as the nucleic base. RNA can therefore also be seen as a character string composed of the letters A, **U**, G, C (with A pairing with U and G with C).

In eukaryotes, it is necessary to wait for the end of transcription to take place before the RNA can be matured out of the nucleus. In prokaryotes, on the other hand, the genomic DNA is not enclosed in a nucleus and the messenger RNA synthesised is rapidly used for translation even though its transcription is not complete.

Three different types of RNA are synthesized:

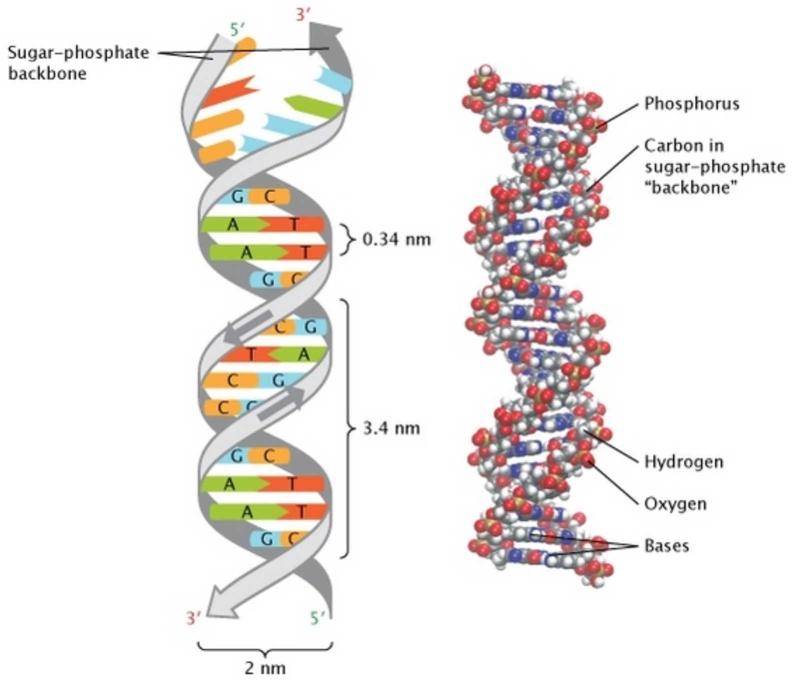


Figure 2.1: The double-helical structure of DNA. [8]

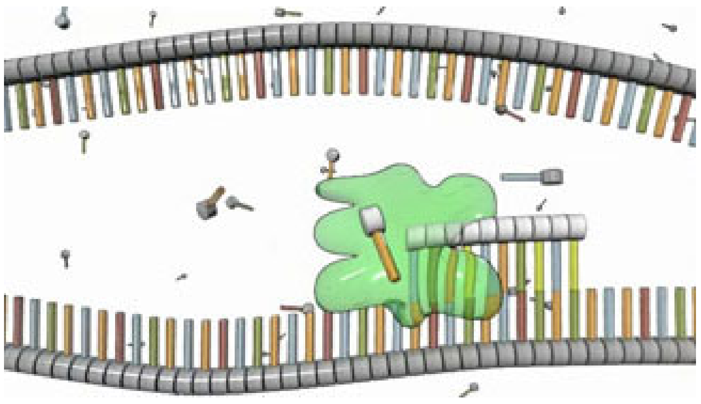


Figure 2.2: The RNA polymerase at work. [8]

- Messenger RNA (mRNA): the sequence to be translated in AA (also called **transcripts**).
- Transfer RNA (tRNA): RNA carrying the amino acid to be added to the peptide sequence.
- Ribosomal RNA (rRNA): principal component of the ribosomes where the translation takes place.

### The mRNAs specificities

The mRNA is therefore the sequence to be translated into protein, but not all parts are coding. Indeed, at the start of the sequence, on the 5' side and at the end of the sequence on the 3' side, there are untranslated regions (UTR) which are used for the initiation and termination of the reading of the strand. In eukaryotes, an mRNA sequence only codes for one protein whereas in prokaryotes, multiple translation initiation sites may be present on the strand. For the latter, a particular sequence known as Shine-Dalgarno (SD [10]) allows the subunit of a ribosome to attach upstream of the translation start sequence (see section 2.1.2) by complementarity with an anti-SD rRNA sequence that it contains. Its center is usually placed between 5 and 13 bases upstream of the start codon (with a preference for E coli between 8 and 10). The full canonical SD sequence is AGGAGGUAA with some variations, but the central GGAGG part is generally well preserved. This region is also called the ribosome binding site (RBS) and although it is not mandatory for the initiation of the translation, a strong SD sequence correlates with a high protein expression ([11]. Figure 2.3 illustrates the arrangement of these different building blocks of mRNA.

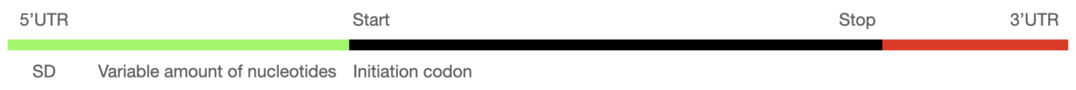


Figure 2.3: The bulding blocks of the mRNA: the 5' UTR containing the SD sequence (RBS), the coding region starting with the initiation codon and finishing with a stop codon before the 3' UTR

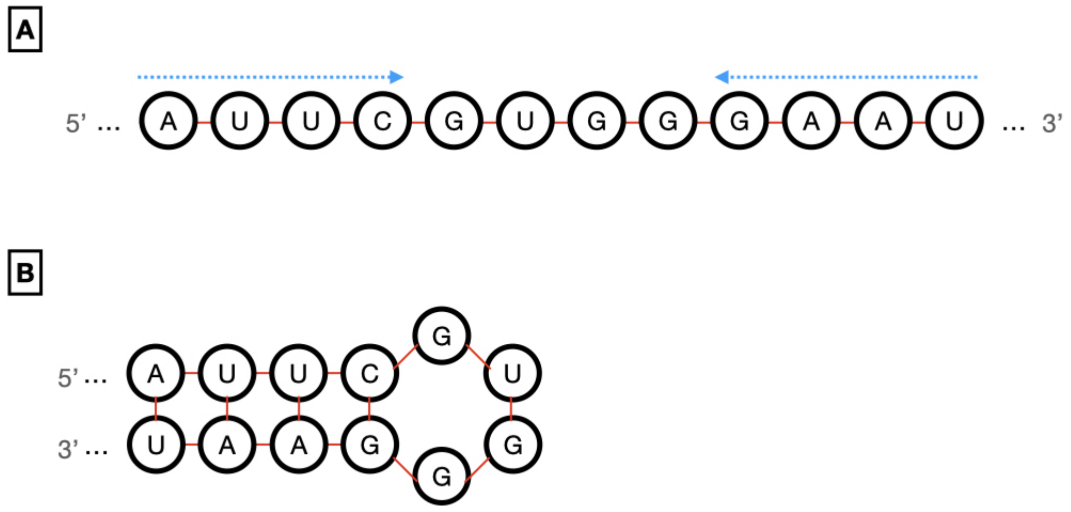


Figure 2.4: The creation of a hairpin loop in a RNA sequence.**A)** The RNA in its linear structure with two inverse sequences separated by 4 nucleotides. **B)** The formation of a hairpin.

Although it is a single-stranded molecule, it can nevertheless generate a secondary structure by intrinsic pairing of complementary nucleotides. If two inverse sequences are in contact, there is a risk of canonical base-pairing, which can lead to the apparition of hairpins or pseudo-knots. This risk is inversely proportional to the distance separating these two regions (see an example in Figure 2.4). The presence of such secondary structures stabilises the RNA but can be problematic for subsequent reading. Indeed, during the translation phase in AAs sequence, a ribosome must pass through the RNA strand and the folded structures can slow down or even completely block its progress. This impact, which seems negative at first glance, is nevertheless increasingly questioned and a secondary structure seems necessary for the stability of the RNA. Since the stability of this structure itself has an impact on the lifespan of RNA within the cytoplasm, it is reasonable to think that a greater degree of secondary structure will allow more time for the intracellular machinery to translate it in protein. RNA will have the opportunity to be translated a greater number of times. We see here the need to find a compromise between a need for secondary structure increasing the lifetime, while reducing the complexity of the latter to promote efficient translation. The messenger RNA, which was merely a vector of information between DNA and protein, is now considered to be a fully-fledged player in the regulation of the synthesis and therefore of the level of expression.

This hairpin structure is also one of the **transcription** termination factor (from DNA to RNA) in prokaryotes. During the progression of RNA polymerase, if an inverted repeat structure appears followed by about 8 U, this causes rapid detachment of the complex formed by the strands of DNA, RNA and RNA polymerase, which is synonymous to the end of the transcription ([12].

### 2.1.2 Translation from mRNA to AA

The mRNA sequence is responsible for the order in which the AAs making up the protein will be assembled. This step is carried out by the ribosomes which are composed of two subunits themselves composed of RNA and proteins. During translation initiation, the ribosomes subunits associate with the 5' side of the mRNA at its non-coding part. In procaryotes, the ribosome binds to the SD sequence in the 5'UTR then begins its reading of the messenger RNA strand which takes place by nucleotides triplets, called **codons**. Once it reaches a start codon (AUG only for eukaryotes; AUG 83%, GUG 14%, UUG 3% in procaryotes [13]), the large ribosomal subunit receives a tRNA loaded with a specific amino acid. If the tRNA presented is complementary to the

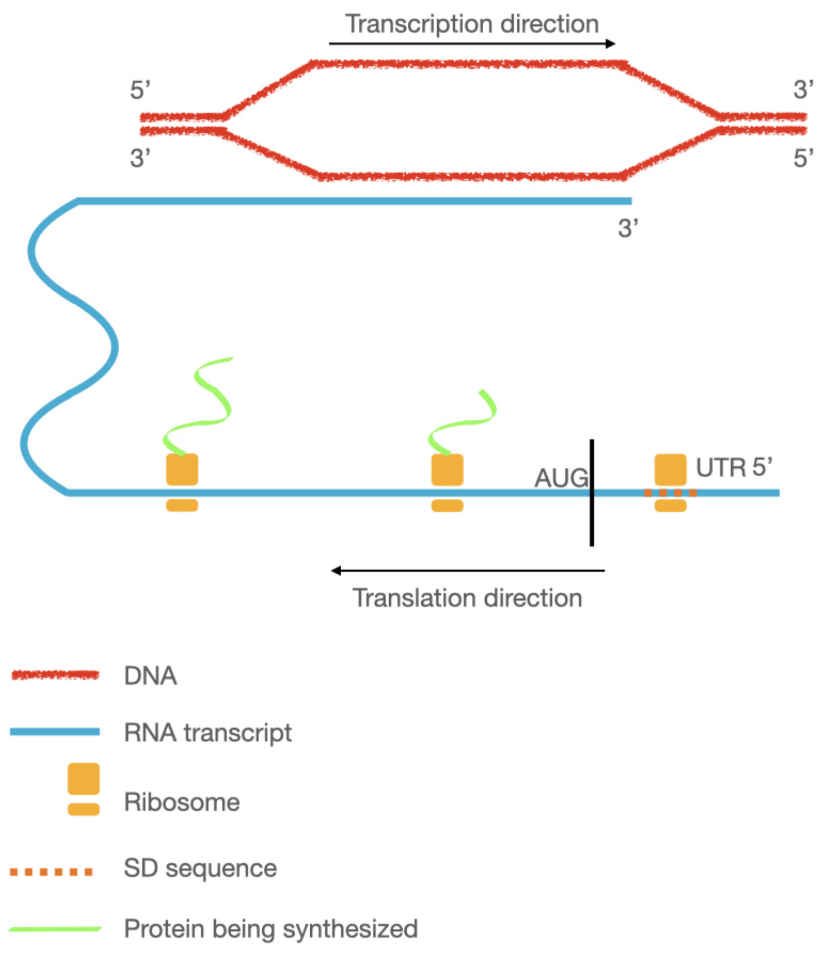


Figure 2.5: A DNA strand is transcribed in a complementary RNA strand which is read by ribosomes in the 5'-3' direction to synthesize the protein.

codon being read, the AA is added to the chain being formed (or is maintained on the ribosome if it is the first). If not, the tRNA is released by the ribosome and another one is tested. An overview of the whole process is given in the Figure 2.5. The upper part of the image shows a double strand of DNA that has been opened by an enzyme (the helices) to make one strand available for the transcription in the 3'-5' direction by the RNA polymerase. This forms a strand of RNA which can be directly used by ribosomes to be translated 5'-3' direction into protein in prokaryotes (unlike in eukaryotes where the RNA must be completely translated, leave the nucleus and undergo maturation before being available). The protein being formed remains attached to the ribosome until it reaches a stop codon which interrupts translation.

A series of three letters of the mRNA (a codon) codes for one amino acid. The particularity of this code is that it is said to be "degenerated". Indeed, since there are four possibilities for each of the basic elements making up a codon, it should be possible to code for 64 amino acids (making 3 times a choice among the 4 possible nucleotides: $4^3$). However, there are only 20 AAs. In reality, some codons are synonymous and code for the same AA. Among these 64 possibilities, there are three particular codons which cannot be translated (UAA, UAG, UGA), they serve as a stop signal to interrupt the lengthening of the AAs sequence. The correspondence between a codon and its AA is given in the table 2.1, the AAs name and abbreviations are given in the table 2.2.

A protein can be made up of several hundred AAs assembled in a linear chain (also called a polypeptide chain). The translation phase of a mRNA sequence into a sequence of AAs linked by covalent bonds requires a large amount of energy.

| 1st Base | 2nd Base | | | | 3rd Base |
|---|---|---|---|---|---|
| | U | C | A | G | |
| U | Phe | Ser | Tyr | Cys | U |
| | Phe | Ser | Tyr | Cys | C |
| | Leu | Ser | STOP | STOP | A |
| | Leu | Ser | STOP | Trp | G |
| C | Leu | Pro | His | Arg | U |
| | Leu | Pro | His | Arg | C |
| | Leu | Pro | Gln | Arg | A |
| | Leu | Pro | Gln | Arg | G |
| A | Ile | Thr | Asn | Ser | U |
| | Ile | Thr | Asn | Ser | C |
| | Met | Thr | Lys | Arg | A |
| | Met | Thr | Lys | Arg | G |
| G | Val | Ala | Asp | Gly | U |
| | Val | Ala | Asp | Gly | C |
| | Val | Ala | Glu | Gly | A |
| | Val | Ala | Glu | Gly | G |

Table 2.1: The RNA codon table

| Amino Acid | Abbreviation | Letter |
|---|---|---|
| Alanine | Ala | A |
| Arginine | Arg | R |
| Asparagine | Asn | N |
| Aspartic acid | Asp | D |
| Cysteine | Cys | C |
| Glutamic acid | Glu | E |
| Glutamine | Gln | Q |
| Glycine | Gly | G |
| Histidine | His | H |
| Isoleucine | Ile | I |
| Leucine | Leu | L |
| Lysine | Lys | K |
| Methionine | Met | M |
| Phenylalanine | Phe | F |
| Proline | Pro | P |
| Serine | Ser | S |
| Threonine | Thr | T |
| Tryptophan | Trp | W |
| Tyrosine | Tyr | Y |
| Valine | Val | V |

Table 2.2: The amino acid abbreviations

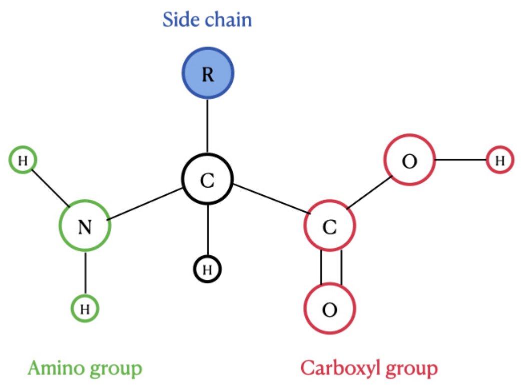


Figure 2.6: The molecular structure of an AA.

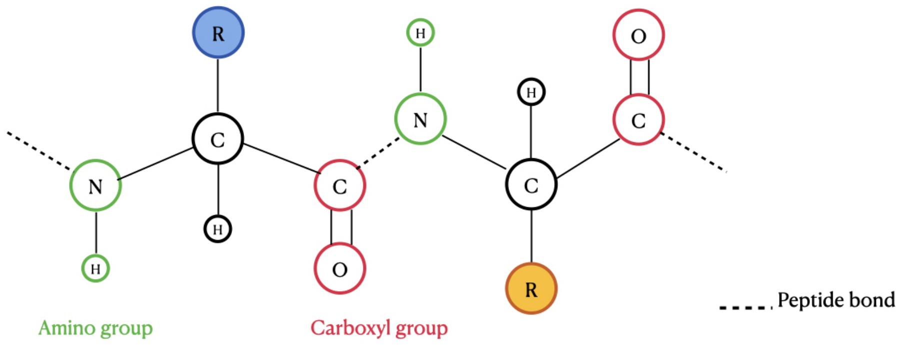


Figure 2.7: A link between two amino acids.

### 2.1.3 Protein conformation

#### Different level of protein structure

The linear assembly of AAs as described in the previous section is important since it determines the three-dimensional structure which specifies its function. This assembly constitutes the **primary structure**. In 1973, Anfinsen proved that the primary structure of amino acids governs all higher level structures of protein ([14]). This allowed him to be awarded the Nobel Prize in Chemistry in 1972 for related work.

So far, these sequences (DNA, RNA, AAs) where considered as strings of characters but to better understand the formation of the final structure, we must delve a little deeper into the molecular structure.

Amino acids (Figure 2.6) are actually made up of a central carbon atom linked to:

- an amino group
- a carboxyl group
- a hydrogen atom
- a variable element called side chain which makes the specificity of each amino acid.

During the translation step, a chemical reaction links the AAs in the ribosome through their amino and carboxyl groups, releasing a molecule of water (Figure 2.7).

These assembled amino acids can also interact through the atoms of their side chains to form **secondary structures** such as $\alpha$-helices and $\beta$-sheets (Figure 2.8). These are patterns that can be formed at different places in the protein.

The **tertiary structure** is the set of all these links which can take place within the same chain of AAs. It is the complete three-dimensional structure of a peptide.

Finally, the **quaternary structure** is a last level of abstraction that is not present for all proteins. Indeed, it is the assembly of several protein subunits forming a macro-molecule assembly. It is the arrangement that these different elements take ([15]).

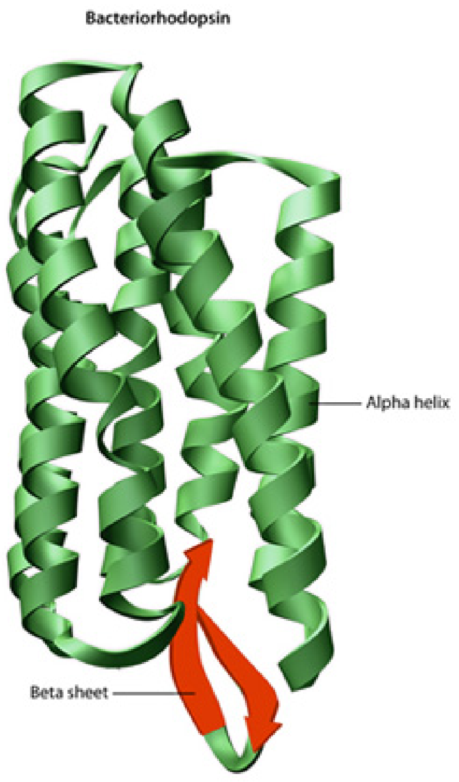


Figure 2.8: Example of a protein containing $\alpha$-helices and $\beta$-sheets.

### Protein Folding

The "final" (also called native) form of the protein is obtained when it has a minimum free energy. Indeed, the different molecules of the side chains of AAs exert between them forces of repulsion and attraction; the goal is to reach the most stable version. It is possible to unfold a protein following different treatments (temperature, use of solvent), this is called denaturation. Some proteins are able to fold back to their native state when returned to the right conditions. This confirms the hypothesis that the information about how a protein folds in its native form is contained in its sequence of AAs ([16, The Shape and Structure of Proteins]).

Already in 1968, Cyrus Levinthal predicted the deterministic nature of the folding mechanism. He introduced the Levinthal's paradox which states that the search for the final structure of a protein cannot be achieved by brute force. Indeed, the amount of conformation possible is such that if they all had to be tested sequentially (even if each test took a picosecond), it would take longer than the age of the universe ([17]). Since the proteins have a much lower folding time, he deduced that they must pass through stable intermediate states. He was simply expressing that a random search could not succeed. This reasoning is interesting in the context of computer modeling since this means that, on the sole basis of the amino acid sequence, an exhaustive search for the stable native state will never be able to succeed in a reasonable time.

Some structures start to fold when they are not yet fully translated and the final conformation also depends on the intracellular conditions. Some proteins can also aid in folding (molecular chaperones); they do not modify the way the structure folds, but they assist it. Proteins are between 50 and 2000 AAs in size and typically fold into subdomains that eventually assemble together.

Natural selection has induced mechanisms to produce stable molecules with predictable character and functionality within the cell. This long process has led to such fine and precise constructions that modifying just a few atoms can destabilize the entire structure and render the protein non-functional ([16, The Shape and Structure of Proteins]).

### Determination of the three-dimensional structure

It is possible to determine the tertiary structure of proteins using laboratory methods such as x-ray crystallography, nuclear magnetic resonance spectroscopy or cryomicroscopy, but these are long and very expensive processes. The increase in knowledge with these techniques therefore remains limited.

Consequently, considerable efforts are made to develop computerized prediction tools. An event is also organized every two years to assess the performance of progress in this area: the Critical Assessment of protein

Structure Prediction (CASP).

Google's DeepMind company has also made significant progress and won the challenge during the 2020 edition (CASP14 [18]). Their alphafold2 algorithm can predict the structure of a protein almost as reliably as with experimental methods ([19]). The company provides access to nearly a million human protein structure predictions as well as a few others from other species (`https://alphafold.ebi.ac.uk`). The alphafold approach requires training data from other folded proteins under the same conditions and in the same cell host.

Another strategy is to perform all-atom molecular dynamics simulations. From a complete sequence, the different interactions at the atomic level are evaluated in order to simulate the dynamics of a folding towards a native state. It is thus possible to obtain all the thermodynamic and kinetic information. The main concern nevertheless lies in the fact that it is only possible to perform these simulations on a limited number of AAs due to the large amount of computing resources required. To carry out simulations of mechanisms of the order of a millisecond, it is necessary to use large clusters working in parallel, which is not accessible to everyone. Other algorithms taking certain shortcuts have been developed to reduce the amount of resources needed, but this remains a complex field ([20]).

These two types of approaches are interesting for visualizing a 3D structure based on the complete sequence. They therefore only depend on the intrinsic factors of the sequence but not on the dynamics of translation (transient states of the protein during synthesis and possible folds during this one), nor interactions with the close environment (the cytoplasm being highly crowded and also containing chaperone proteins). To obtain an identical AA sequence, the use of the same mRNA sequence in different cellular organisms can lead to proteins having a different three-dimensional structure.

#### Why does it matters?

The chaperone molecules help folding by driving the structure towards the lowest free energy (global minimum), but if small modifications occur, other minima can appear and generate another form of stable structure than that expected. This change can be aggravated by a high concentration within the cytoplasm or an increase in temperature. In a healthy organism, control mechanisms make it possible to eliminate these malformed proteins or even sometimes to unfold them to bring them back to the desired state. In the cases where these mechanisms are deficient, different scenarios are then observed: the functionality supposed to be conferred to the cell is deficient (since the desired form is not present) or the new aggregated protein has a toxic effect. Among the diseases related to these problems, we find for example phenylketonuria, cystic fibrosis, Parkinson to name a few ([21]).

In the context of this work and recombinant proteins in *E. coli*, a misfolding is also problematic. If the protein are malformed, they can either be degraded or aggregated. This shape (called an inclusion body) is regularly used to recover the protein. It is in fact an insoluble molecule that is relatively simple to isolate by centrifugation after cell lysis. Chemical agents are then used to unfold the protein and restore it to the desired shape using optimized solutions. But this treatment is not applicable in many cases and the strategy is then to optimize the protein so that it is synthesized in its native form *in vivo*([22]). Since it is the shape of the protein that gives it its function, if it does not obtain the conformation as in the original host, it will not induce the desired immune response. As in the case of degenerative diseases, a protein that does not have the optimal conformation can have a toxic effect on cells during the production phase in *E. coli* which will obviously have an impact on the productivity of the culture.

As explained previously, it is the mRNA sequence which is mainly responsible for the way in which the protein will fold, whether by its intrinsic sequence (the sequence of its nucleotides) and the interactions between AAs, but also by the kinetics of translation mediated by different parameters which will be explained further. In the case of *E coli*, the process is very fast, translation can take place while transcription from DNA is not yet complete. And during translation, folding of the chain of AAs in formation can occur, this is called co-translational folding. Many other factors come into play for the refolding of proteins in their native conformation but the optimization of the mRNA sequence according to the host which must express the protein has a central role.

## 2.2 Recombinant protein expression in *E. coli*

This section will explain the reasons for using *E. coli* for the expression of heterologous genes (which are genes not normally produced by this cell) and the means used to achieve this. It should also be remembered that this host can only be used in the absence of the need for post-translational mechanisms (such as glycosilation) for the synthesis of the protein of interest. The explanations below come from the review of Baneyx ([23]) which dates from 1999. Although some technological advances have emerged, the main mechanisms have hardly evolved but some interesting elements from a more recent review ([24]) will nevertheless be added.

*E. coli* is a well-known bacterium and a system of choice due to its ability to grow rapidly at high cell density in inexpensive culture media. Under favorable conditions, the doubling time of the cell quantity is 20 minutes (but this can be influenced by the synthesis of recombinant proteins which use the metabolic machinery). It is therefore possible to achieve very high cell densities of the order of 10E+10 cells/mL. Host transformation is quite fast and easy, the gene of interest is cloned, transformed to adapt it to the host, the production is induced to finally harvest and purify the protein.

To integrate the gene of interest, there are two possible tactics: insertion into the genome or the use of plasmids. The first method is rarely used because it raises many constraints and the second one is therefore widely preferred. It seems useful to remember that bacteria do not have a real nucleus and that their genome resides in the cytoplasm. Plasmids are DNA sequences distinct from chromosomal DNA and capable of autonomous replication. They are generally circular and it is possible to integrate artificial constructs to express. A simple plasmid consists of a replicon, a promoter and a selection marker.

The replicon controls the number of plasmids present in a cell. It might be tempting to add a large number of plasmids in order to produce a lot of mRNA and thus a lot of proteins, but in reality, this has rather a deleterious effect on the cell because it uses up its energy resources in large quantities. Growth is then reduced and this does not benefit production. Depending on the case, the number of plasmids is therefore limited to between 15 and a few hundred.

During cell division, it is not sure that the plasmid of interest will be present in the daughter cells. It could happen that the cells that do not contain them become predominant, which would have an impact on production. The main tactic to avoid this is to insert a resistance gene for a certain antibiotic into the plasmid. The antibiotic is added during the culture and the cells that do not have the resistance gene are then killed. The review by Rosano et al ([24]) mentions another selection marker using a cell line whose genome has been modified and does not contain a gene essential for its survival. This gene is added to the plasmid used to express the protein of interest. If the cell does not have this plasmid, it is unable to express a protein essential to its survival and it dies. However, this practice is not yet widely used.

The production of the protein does not take place throughout the whole culture. The gene of interest must be induced by a promoter when the end of the cell growth phase is reached. Different types of promoters exist with varying efficiencies. One of them is dependent on the addition of lactose in order to express the protein of interest, which can then represent 15-30% of the amount of total protein in the cell. The bacteriophage T7 promoter is also an interesting candidate since it makes it possible to achieve production of the order of 40-50% with respect to the total quantity of proteins. Finally, it is possible to use temperature-sensitive promoters, but they are rapidly repressed (after 1-2h) which may be too short for significant protein accumulation. All these possibilities are interesting, but it is difficult to predict which will be the best strategy and they will have to be tested.

The mRNA is rather unstable in *E. coli* with a half-life of the order of 30 seconds to 20 minutes. This relatively short time does not allow a sequence to be translated a large number of times and some strategies are evaluated to increase its stability (addition of a poly A tail, increase of the secondary structure by synonymous modification of some codons to name a few).

As explained previously, the translation requires a SD sequence in 5'UTR followed by 4 to 14 nucleotides then a start codon. A stable region (with RNA secondary structure) between the SD sequence and the start codon can reduce the expression because it interferes with the ribosome binding site. Differences in codon usage (see next section) between prokaryotes and eukaryotes can have a significant impact on the production of heterologous proteins. The presence of unused codons in *E. coli* can induce mRNA, plasmid, and production instability. It can also inhibit protein synthesis or have a toxic effect on the cell. This is particularly the case of the AGA codon coding for arginine which does not have any complementary tRNA in *E. coli.* This will then sometimes be translated into lysine, especially when using an impoverished culture medium.

Overproduction of protein in the cytoplasm can lead to misfolding, segregation, or the generation of insoluble aggregates known as inclusion bodies. These can easily be isolated and solutions exist to fold them correctly but there are no guarantee and it often results in failure (rarely published in the literature). Growing at a lower temperature can sometimes help. Many research are also carried out to use chaperone molecules and foldases which make it possible to assist in folding. A co-overproduction of these assistants could make it possible to increase the quantity of soluble proteins. But there again, there is no certainty of success.

This section had no other purpose than to illustrate that a lot of parameters come into play to achieve efficient production of recombinant proteins and there is no rule or ready-made formula that helps ensure success. But before testing all these factors, the DNA sequence (and the RNA by direct transcription) is the first essential element. If this sequence is poorly designed, varying the above parameters will have no impact.

Rosano et al ([24]) mentions that there is really no winning combination and that the different possibilities will have to be tested. This is made easier by high-throughput laboratory techniques and ever-increasing automation. They nevertheless propose some troubleshooting tips. One of them requires a computational approach, optimizing the mRNA sequence by taking advantage of codon usage bias (see next section for deeper explanations). Codon usage bias is the difference in usage of synonymous codons between the host of origin and the host of destination. This can affect the level of synthesis or the functionality of the protein. Rare codons are defined as codons used at less than 1%. This is for example the case of the AGG codon for arginine (<0.2% in *E. coli*). To resolve this difference, two strategies are proposed:

- A silent mutagenesis that does not alter the AA sequence. This can be done for the whole gene or on some parts only. This problem is far from trivial given the number of possible combinations and the possible strategies. One tactic is to always use the codon most frequently used by the host for each AA. But one of the hypotheses would be that in reality, the use of different codons would play more of a role on the secondary structure and that a better expression would take place when the structure is weaker. The authors state that new sequence optimization algorithms based on codon usage bias should take into account the secondary structure of the RNA, the presence of SD-like sequences and other elements that make it possible to mimic the kinetics of translation to allow for co-translational folding.

- An increase in the availability of tRNA loaded with their specific AA by modifying the host genome so that it overexpresses the less frequent ones. This can generally increase the level of expression, but often results in a decrease in solubility due to misfolding problems.

Among all these elements, the codon usage bias is very often used since a lot of evidence points that it influences the dynamics of the translation and the co-translational folding of proteins. Although other mechanisms intervene downstream, if this step is already faulty, there is a high risk of problems later on. In some cases, it is possible to isolate the synthesized protein, unfold and refold it into the desired shape, but this is not always possible and involves additional processing. It therefore remains preferable to produce the protein directly in its native form. This is the reason why emphasis on the use of codon bias will be made in this work. This is the subject of the following section which will explain what it is, its impact and how it is possible to use this property.

## 2.3 Codon bias usage

Until now, the translation step was considered by the recognition of a codon (3 nucleotide bases) by a tRNA presenting a complementary sequence (an anti-codon). But it is possible that certain tRNAs are not expressed (their gene is not present for certain organisms). When reading mRNA, it may still happen that a codon requires a tRNA that is not expressed within the cell. To overcome this lack, it is possible that this pairing does not take place on the basis of a canonical association (Watson-Crick), but that a more permissive association occurs at the level of the 3rd base. This is the wobble effect ([25]).

The degenerate nature of the genetic code was exposed previously through the possibilities for different codons to code for the same AA. One might think that the choice of the codon is arbitrary and that the choice of a so called synonymous triplet will result in the same protein. Since they code for the same AA, their behavior seems *a priori* identical, but this is not the case. In reality, great disparities exist from one species to another, within the same species between genes and even within the same gene. Preferential codons coding for the same AA are used instead of others, this is called the codon usage bias (CUB). This particularity is now recognized as essential and has an impact on different steps of protein synthesis such as RNA secondary structure, translation and protein folding. Modifications of the RNA code can induce an increase in the expression of a transgene (the incorporation of a gene not normally expressed naturally by the cell) of more than 1000 times in certain cases ([26]).

The article proposed by Plotkin et al ([27]), draws an inventory of knowledge on the subject. All points will not be covered, but it is interesting to focus on the origins of this CUB and how this can have an impact on the quantity and quality of proteins synthesized, whether they are endogenous or resulting from a genetic modification allowing heterologous expression.

Two families of hypotheses (not mutually exclusive) attempt to explain these codon usage variations: mutational versus selective reasons. The mutational explanation postulate that the CUB is caused by punctual mutations that have arisen randomly during evolution. This mutation is seen as neutral because it is not an adaptive result that would give any advantage or disadvantage. This hypothesis is the main clue for explaining inter-species mutations. The explanation of selection, on the other hand, postulates that these mutations influence the state of the organism, which can make it weaker or stronger, which will repress or favor them during evolution.

### 2.3.1 Endogenous genes expression

The greatest variation in codon usage between species is the GC content which is linked to mutational processes at the genome level. Another element that can co-evolve is the regulation of tRNA expression. Indeed, the adaptation of codon choice can also be directed by the abundance of tRNA present in the cell. In general, the CUB is correlated to the respective amounts of complementary tRNAs.

Within the same genome, the choice of preferential codons between the different proteins corresponds rather to the selection hypothesis. A strong correlation is observed between the abundance of a certain protein and the use of preferential codons (and therefore to the abundance of corresponding tRNAs). However, the use of optimal codons does not seem to correspond to the idea that it is the initiation phase which limits translation for endogenous genes (contrary to heterologous genes, see below). To optimize the quantity of protein produced, it is rather a question of using a promoter making it possible to synthesize more mRNA than of using codon mutations which will have little effect. One of the hypotheses in favor of the use of optimal codons would be that it allows a reduction in energy requirements, but this aspect is still debated within the scientific community.

Within the same gene, the choice of codons for the same AA can also vary because this influences the speed of translation of certain sequences which can induce ribosomal pauses to promote co-translational folding. These pauses are due to two reasons:

- a suboptimal codon
- a nucleotide sequence inducing a secondary structure of the mRNA that prevent the efficient progress of the ribosome.

Even though highly expressed endogenous proteins have more optimal codons within their mRNA than low expressed proteins, suboptimal codons are still present. We therefore deduce that the speed of translation for endogenous genes does not have an impact on the quantity of proteins produced, but rather on its quality.

The study of Mauger et al. ([28]) carried out on eukaryotes sheds some light on these remarks to be able to dissociate the impact of these two factors. Indeed, these are intimately linked since a substitution of a codon by a synonym will inevitably have an impact on the secondary structure of the RNA. It is therefore impossible to determine whether it is the impact of the codon change alone or the secondary structure that induces changes in expression. It is commonly accepted that highly expressed genes mainly use optimal codons and that suboptimal codons induce ribosomal pauses and therefore mRNA half-life reduction. If the lifespan is reduced, the number of possible translations by the ribosomes will be reduced accordingly. A large secondary structure in the 5' UTR will have a significant impact on translation since it will make it difficult for the ribosomes to attach to the initiation site, but the impact of the secondary structure in the coding region is difficult to assess. They were able to study this impact by using modified nucleotides (eg pseudouridine) which allow the same codon to be retained, but have properties which strengthen or weaken the secondary structure of the mRNA. They were able to conclude that the weakening of the secondary structure in 5' UTR and of the first 30 nucleotides of the coding sequence favored a better expression of the protein. For the rest of the coding sequence and the 3' UTR region, on the other hand, they were able to determine that an increase in the secondary structure allowed an improvement in expression. They therefore deduced that codon optimality and a stronger secondary structure in the coding region have distinct effects that combine to enhance protein expression in eucaryotes. Their hypothesis is that these parameters influence the movement of ribosomes which are slowed down to allow the intermediate folds of the protein being synthesized. They also suggest that these ribosomal pauses allows a better distribution along the strand in order to avoid ribosome collisions, which degrade the mRNA.

One of the important parameters arguing for the selection hypothesis is a greater secondary structure of the mRNA on the side of the initiation site. A large structure on the 5' side can indeed prevent proper pairing of the ribosome and limit the initiation of translation.

This use of the codon bias is also exploited for highly expressed genes just after the initiation phase during the start of translation (about 50 nucleotides). A non-optimal codon ramp in the first nucleotides causes slow elongation at the start of translation, which avoids traffic jams at the end (3') or even in the middle of the sequence when the ribosome reads a rare codon. This avoids translation abort since if a ribosome remains blocked for too long it ends up detaching itself and produces a protein which is not fully formed. This mechanism may also promote early recruitment of chaperone molecules ([29]). A schematic view of this codon ramp is shown in Figure (2.9).

### 2.3.2 Heterologous genes expression

The first synthesis of a protein (somatostatin composed of 14 AAs) from the human genome into the *E coli* bacterium was carried out in 1977 ([30]). At the time, the *E coli* genome was not yet well known and the data from another phage had been used to generate a sequence that took into account the specificities of

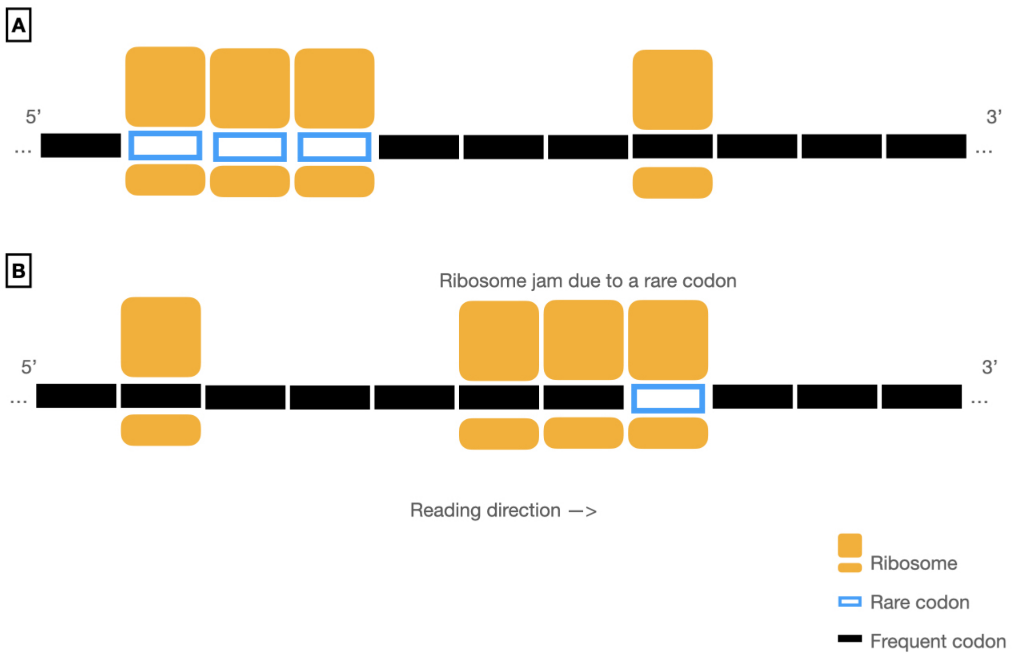


Figure 2.9: A) At the beginning of the translation, some rare codon are used to slow down the progression of the ribosome, then frequent codon are used, this allows spacing between the ribosomes for further elongation. B) Frequent codon are used, then a rare codon occur, which blocks the most advanced ribosome and creates a traffic jam.

the codon choice. They also focused on removing possible interactions between nucleotides (which could have created secondary mRNA structures) that could have hindered synthesis. They also avoided GC-rich sequences followed by AT-rich sequences which were considered transcription termination factors at the time. It was a small sequence that was relatively easy to edit without the need for computerized solutions, but it nevertheless showed the need to modify the original sequence.

A principal component analysis of the codon usage preference of different organisms is proposed by Gustafsson et al ([26]) and its representation is given in Figure (2.10). This analysis of the frequency of use of 62 codons (Met and Trp having been eliminated from the analysis because they are encoded by one single codon) for eight different species allows a 2-dimensional visualization of the similarity or not between them. It is clear that the codon usage bias of *E coli* and humans is very different and that the transposition of the genetic sequence from one to the other will not result in efficient synthesis of the desired protein.

In the case of the expression of a heterologous gene, the choice of codons seems to play a predominant role. It should indeed be remembered that in this case, it is not uncommon for the expression of the protein of interest to exceed 30% of the total quantity of intracellular proteins. Therefore, the use of different codons can have a significant impact. Indeed, if rare codons are used for an amino acid, there is a risk of rapid exhaustion of the corresponding tRNAs, which could lead to interruptions in translation. Additionally the amount of mRNA for this expression may be such that translation would be limited by the amount of ribosomes available. If the mRNA contains a lot of infrequent codons, the elongation time will be long, reducing the availability of ribosomes for a new initiation (since they are in use), which will have an impact on the synthesis. Initiation is therefore less impacted by its intrinsic sequence than is the case for endogenous proteins. The limiting factor would rather be the availability of the ribosomes and optimizing the coding sequence using optimal codons would make more sense here since rapid translation would make the ribosomes more quickly available for new initiation. The impact would then be redirected to the amount of tRNA available.

Codon usage frequency correlates with tRNA abundance. Indeed, during the course of the mRNA by the ribosomes, when a codon is read, a tRNA is presented. If this corresponds to the codon, the amino acid it holds is added to the peptide chain being formed. It is therefore obvious that if a specific tRNA is abundant, there is a greater chance that the process will proceed quickly. Therefore, two strategies exist to optimize the synthesis: the modification of the host so that it expresses in greater quantity the weakly present tRNAs or the modification of the mRNA sequence so that it uses the codons preferentially used by the host.

The first approach was discarded because it induced modifications to the basic metabolism of the cell, but also because although the quantity of protein of interest was increased, its soluble fraction (and therefore the ease of purification) was reduced. The most probable explanation being that synthesis was proceeding too

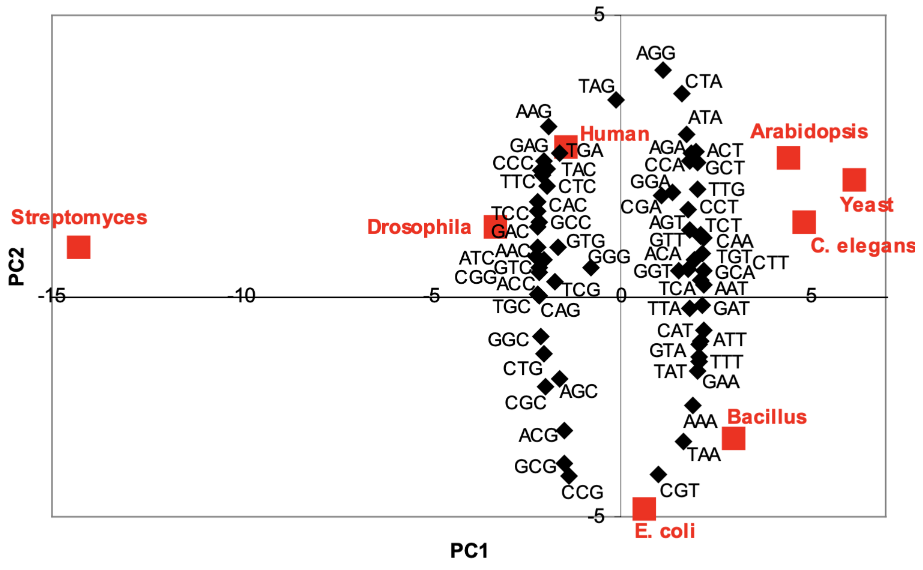


Figure 2.10: Principal Component Analysis of codon usage between 8 different species.

quickly and co-translational folds were not occurring, leading to non-soluble forms of the protein ([31]).

The second approach is far from being trivial since different tactics can also be applied and they have collateral effects, but it is mainly in this area that research is most active. Commercial services exist (of which the best known are Genewiz and Genscript) but the way in which the optimizations are implemented are not revealed and it is difficult to assess which type of modification of the sequence has a positive impact or not on the synthesis.

A recent study aimed at the production of the hepatitis virus in E coli ([32]) was able to confirm these assumptions. Many unsuccessful attempts to produce certain proteins of this virus in this bacteria have been made since the 1970s so that eukaryotic expression systems are still used today despite their constraints (mainly cost). The authors were able to demonstrate that the absence of secondary structure near the initiation site, but also downstream of it, made it possible to synthesize proteins that had never been synthesized until now.

The presence of large amount of G and C nucleotides has a significant impact on the secondary structure of mRNA since they are more strongly linked to each other than A and U nucleotides. The presence of such elements at the start of the sequence is therefore deleterious to synthesis in prokaryotes, which implies that more than 40% of the genes expressed in humans cannot be expressed in *E coli* without modification. However, the entire structure should not be systematically weakened by introducing synonymous codons containing A and U, since these mRNAs would then have a too weak a structure which is rapidly subject to degradation (so translated a fewer amount of times).

## 2.4 Conclusion

In this chapter, the process of protein synthesis was discussed in both eukaryotes and prokaryotes to understand the optimization problem. The proper folding of proteins was emphasized as crucial for their three-dimensional structure and function. The case of *E.coli* was also presented as a preferred host for heterologous production due to its low cost and fast growth. The determinants of high quantity and quality production were highlighted, with codon usage bias being a significant factor. To achieve optimal results, it is necessary to modify the original sequence based on the specificities of the destination host, including avoiding a high-use codon ramp and a secondary structure at the beginning of the sequence to facilitate efficient ribosome pairing. The next chapter will provide a detailed explanation of how to calculate codon usage bias and predict RNA secondary structure, as well as well as algorithms strategies to modify sequences to meet the criteria mentioned above.

# Chapter 3

# Computer science context

This section will explore the different measures used for designing and characterizing an optimal sequence. An evolution of the different algorithms will then be presented, starting from a simple optimization always choosing the same preferred codon for an amino acid and then moving on to harmonization solutions that take into account the specificities of the destination host that must synthesize a foreign protein. Within this category of algorithms, different classes are to be considered: a randomization of the choice of the codon according to the frequency of use in a set of reference genes or even a solution taking into account clusters of codons less frequently used in the reference host that must be passed on to the destination host.

But these elements only address one factor of sequence optimization and do not take into account the secondary structure that an RNA strand can take. As described previously, this is an important element during the initiation of translation where it is preferable to minimize the pairing of complementary nucleotides, which can hinder the proper attachment of the ribosome. It will therefore be important to find tools to predict these potential secondary structures and determine a strategy to minimize them.

## 3.1 Codon usage

In order to optimize a sequence, it is necessary to use numerical data. Different solutions have been proposed. Although these all correlate with each other, they have different uses and meanings. These metrics nevertheless always come from the same source: genome sequencing data.

Different measures can be obtained depending on the objective and optimality index used. One commonly used measure is the CUB index, which condenses the information in a coding sequence to provide a biological interpretation. The CUB index has several advantages, such as being computationally simple and not requiring experimental methods. For instance, one method of calculating the CUB index involves counting the number of gene copies that encode tRNAs complementary to codons. These tRNAs are crucial in delivering amino acids for protein synthesis when they pair with the appropriate codons. The hypothesis behind this approach is that the greater the number of gene copies that code for the same tRNA, the more essential it is for the cell to possess these tRNAs, since they are mainly used for the corresponding codons. This is the codon usage value used by *Waldman et al* [33] to define the tRNA adaptation index (tAI).

The conventional approach to calculate the CUB is to identify gene regions in the genome that code for specific proteins. By analyzing both the DNA sequencing data and the corresponding amino acid sequences, it is possible to annotate the specific regions of DNA that serve as the basis for the amino acid sequences constituting the proteins. These DNA sequences, which lie between the start and stop codons, are known as open reading frames (ORFs). The complete collection of all ORFs in a genome is referred to as the ORFeome, which is a subset of the genome that encodes all the proteins of the organism. These are just two examples of the many methods available for computing the CUB. A recent review of existing indices [34] reveals that all of these values can be used to calculate indices that correlate with each other even though they are actually intended to show different aspects of CUB.

In the present study, the second approach has been chosen because it forms the foundation of the most commonly utilized index, namely the codon adaptive index (CAI), which will be discussed in subsequent sections. It has been observed that highly expressed genes tend to contain a higher proportion of frequent codons and fewer rare codons, as opposed to weakly expressed proteins. This insight has led to the selection of the aforementioned approach for the current investigation [35].

### 3.1.1 Codon usage frequency table

With the availability of DNA sequencing data, it is now feasible to extract information from an entire genome and align it with its corresponding protein. These sequencing data enable the determination of the frequency of usage for each codon, which is expressed in terms of frequency per thousand. However, it should be noted that these frequency values can be highly dependent on the dataset used and may vary both within a genome and between species. Depending on the specific research goals, it may be desirable to prioritize the usage of the most frequently observed codons in a subset. For example, if the aim is to investigate the CUB of highly expressed proteins, then it would be interesting to calculate the usage frequencies using a dataset of highly expressed proteins.

When calculating the optimality indices and conducting optimization analyses, the objective would then be to reproduce the frequency of codon usage observed in the highly expressed proteins of interest on the mRNA sequence that codes for the target heterologous protein. To generate such a frequency table, only the genome (or a portion of it) in the form of a character string is required. However, it is important to ensure that this string comprises only legal DNA characters (i.e., A, T, G, and C) and that its length is a multiple of three (which corresponds to the size of a codon). Such sequencing data are widely available online for different species and are typically well-annotated. Custom datasets can also be generated based on information coding for specific proteins of interest, whereby the DNA sequences of each protein are retrieved and combined into a single text file.

For each of the 62 codon possibilities, the following expression must be calculated:

$$X_{ifreq} = \frac{number\ of\ X_i\ in\ the\ sequence}{total\ number\ of\ codons\ in\ the\ sequence} * 1000 \tag{3.1}$$

Where $X_i$ is the occurence of a specific codon.

The algorithm to compute all the codon usage frequency with a dictionary structure is trivial and is presented in pseudocode 1. It considers that the sequence data to be analysed is already imported in a “sequence” variable.

**Algorithm 1** The computation of a codon usage table

```
$CODON\_SIZE \leftarrow 3$
$codon\_usage\_table \leftarrow \{'TCA' : 0, 'ATT' : 0, ..., 'GAC' : 0\}$    ▷ The 62 possibilities
$nucleotide\_count \leftarrow length(sequence)$
if $sequence \ \% \ CODON\_SIZE \neq 0$ then
    Throw Exception
end if
$codon\_count \leftarrow \frac{nucleotide\_count}{CODON\_SIZE}$
for $i = 0; i < nucleotide\_count; i+ = CODON\_SIZE$ do
    $codon \leftarrow sequence[i : i + CODON\_SIZE]$
    $codon\_usage\_table[codon]+ = 1$
end for
for each codon in codon_usage_table do
    $codon\_usage\_table[codon] = \frac{codon\_usage\_table[codon]}{codon\_count} * 1000$
end for
return $codon\_usage\_table$
```

The computation of the CUB table may seem straightforward, but it heavily relies on the input dataset. In the case of using the complete genome of a host, publicly available databases can be used to avoid the search for the genome and streamline the calculation process. One of the most commonly used databases is the Kazusa database [1], although it has not been updated since 2007. Recent research by *Matthew et al* [36] suggests that the HIVE-CUT database [2] is a better-maintained alternative and provides a more extensive coding sequence for the calculation of CUB tables. This database is regularly updated and can provide up to 8000 times more coding sequences than the Kazusa tables for a particular strain of *E. coli*. Notably, the HIVE-CUT database shows a more pronounced codon usage bias. In the present work, following the authors’ recommendation of using the latest database is advisable when analyzing the entire genome of the host organism.

In cases where only a more restricted set of the genome should be considered (use of highly expressed proteins only for example), the calculation using the algorithm explained above will still be necessary.

[1]http://www.kazusa.or.jp/codon/
[2]https://dnahive.fda.gov/dna.cgi?cmd=cuts_main

An example of codon usage values for *E. coli* strain K12 is provided in table 3.1. In this table are compared the values obtained from the Kazusa and Hive Cut database as well as the data calculated on the basis of a subset of highly expressed proteins only (see 3.2.1 for more explanation on the protein making up this data set). As explained previously, the values vary greatly depending on the reference data set chosen. The sum of all frequencies is more or less equal to 1000 subject to rounding errors.

Table 3.1: Codon usage table for the K12 strain of *E coli* considering the whole genome (for HiveCut and Kazusa) or a subset of highly expressed genes.

| Begin of codon usage table | | | |
|---|---|---|---|
| Codon | HiveCut | Kazusa | Highly Expressed proteins |
| TTT | 22,31 | 19,70 | 7,30 |
| TTC | 16,54 | 15,00 | 25,25 |
| TTA | 13,76 | 15,20 | 1,37 |
| TTG | 13,65 | 11,90 | 1,22 |
| CTT | 11,07 | 11,90 | 2,89 |
| CTC | 11,14 | 10,50 | 2,43 |
| CTA | 3,90 | 5,30 | 0,46 |
| CTG | 53,09 | 46,90 | 64,19 |
| ATT | 30,53 | 30,50 | 8,97 |
| ATC | 25,19 | 18,20 | 46,09 |
| ATA | 4,24 | 3,70 | 0,15 |
| ATG | 27,89 | 24,80 | 24,49 |
| GTT | 18,31 | 16,80 | 48,52 |
| GTC | 15,32 | 11,70 | 3,65 |
| GTA | 10,92 | 11,50 | 24,49 |
| GTG | 26,31 | 26,40 | 10,95 |
| TCT | 8,43 | 5,70 | 18,56 |
| TCC | 8,65 | 5,50 | 13,69 |
| TCA | 7,09 | 7,80 | 1,37 |
| TCG | 8,87 | 8,00 | 0,30 |
| CCT | 7,01 | 8,40 | 1,98 |
| CCC | 5,44 | 6,40 | 0,30 |
| CCA | 8,42 | 6,60 | 3,95 |
| CCG | 23,33 | 26,70 | 26,77 |
| ACT | 8,86 | 8,00 | 26,32 |
| ACC | 23,40 | 22,80 | 27,23 |
| ACA | 6,95 | 6,40 | 2,13 |
| ACG | 14,39 | 11,50 | 2,43 |
| GCT | 15,29 | 10,70 | 49,13 |
| GCC | 25,70 | 31,60 | 6,39 |
| GCA | 20,19 | 21,10 | 29,21 |
| GCG | 33,85 | 38,50 | 20,38 |
| TAT | 16,08 | 16,80 | 5,17 |
| TAC | 12,21 | 14,60 | 20,38 |
| TAA | 2,04 | 1,80 | 3,95 |
| TAG | 0,22 | 0,00 | 0,00 |
| CAT | 12,84 | 15,80 | 3,50 |
| CAC | 9,63 | 13,10 | 12,47 |
| CAA | 15,30 | 12,10 | 3,80 |
| CAG | 28,96 | 27,70 | 32,10 |
| AAT | 17,54 | 21,90 | 2,28 |
| AAC | 21,54 | 24,40 | 40,46 |
| AAA | 33,69 | 33,20 | 59,63 |
| AAG | 10,32 | 12,10 | 15,52 |
| GAT | 32,14 | 37,90 | 18,25 |
| GAC | 19,16 | 20,50 | 42,44 |

| Continuation of Table 3.1 | | | |
|---|---|---|---|
| Codon | HiveCut | Kazusa | Highly Expressed proteins |
| GAA | 39,70 | 43,70 | 58,11 |
| GAG | 17,96 | 18,40 | 14,76 |
| TGT | 5,11 | 5,90 | 1,52 |
| TGC | 6,40 | 8,00 | 3,04 |
| TGA | 0,93 | 1,00 | 0,15 |
| TGG | 15,21 | 10,70 | 5,63 |
| CGT | 21,01 | 21,10 | 41,22 |
| CGC | 22,03 | 26,00 | 15,52 |
| CGA | 3,50 | 4,30 | 0,15 |
| CGG | 5,33 | 4,10 | 0,15 |
| AGT | 8,67 | 7,20 | 1,67 |
| AGC | 16,00 | 16,60 | 7,91 |
| AGA | 2,01 | 1,40 | 0,15 |
| AGG | 1,11 | 1,60 | 0,00 |
| GGT | 24,77 | 21,30 | 50,81 |
| GGC | 29,64 | 33,40 | 35,29 |
| GGA | 7,88 | 9,20 | 0,46 |
| GGG | 11,04 | 8,60 | 0,91 |
| Sum | 1000,01 | 1000,10 | 999,96 |
| End of Table | | | |

However, this table is not directly used for calculating indices or optimizing a sequence. It needs to be transformed into another form based on the specific needs. The aim of these transformations is to present a **relative value of codon usage with respect to its corresponding amino acid**. In the original table, if the reference data set used contains a small proportion of a particular amino acid, the codon usage value of all these codons will be underestimated. Thus, it is necessary to normalize these data by considering the proportion of each amino acid. In the following sections, we will discuss four useful transformations that can be applied to these data.

### 3.1.2 Relative synonymous codon usage

In 1987, *Sharp et al* [37] proposed an index of optimality of a sequence from the point of view of its CUB, the codon adptation index which will be discussed in a later section (3.2.1). In order to calculate this index, the first step is to compute a table of the codon usage relative to each amino acid. They call this value the relative synonymous codon usage (RSCU) which is the value of the observed frequency of a codon divided by the expected frequency assuming all codons are used equally for the same amino acid.

For a particular codon (X), it is calculated as follows:

$$RSCU_{ij} = \frac{X_{ij}}{\frac{1}{n_i}\sum_{j=1}^{n_i} X_{ij}} \tag{3.2}$$

Where $X_{ij}$ is the observed number of the $j^{th}$ codon for the $i^{th}$ amino acid and $n_i$ is the number of possible codon (from on to six) for the $i_th$ amino acid.

To perform this calculation programmatically, a table of codon frequency values is needed as input. The algorithm first calculates the sum of frequency values for each codon associated with an amino acid, in order to avoid redundant calculations. Then, for each codon, the formula mentioned in (2) is applied. It should be noted that the sum of RSCU values for all codons that code for the same amino acid equals the number of codons that encode it, ranging from 1 to 6.

### 3.1.3 Relative adaptiveness of a codon

The RSCU values are not normalized across different amino acids since the scale of the value representation depends on the number of codons for each amino acid. To address this issue, the authors of the RSCU proposed an alternative approach by calculating the relative adaptiveness of a codon. This measure represents the frequency of use of a codon relative to the most frequently used codon for a given amino acid. The relative adaptiveness is calculated using the following equation:

**Algorithm 2** The computation of a RSCU table

```
Require: $CODON\_USAGE\_TABLE$
  $aa\_sum\_freq\_related\_codons \leftarrow \{'S':0,'N':0,...,'Q':0\}$        ▷ The 20 possibilities
  for each aa in aa_sum_freq_related_codons do
      $aa\_sum\_freq\_related\_codons[aa] = sum(codons\_coding\_for\_this\_aa)$
  end for
  $rscu\_codons \leftarrow \{'TCA':0,'ATT':0,...,'GAC':0\}$        ▷ The 62 possibilities
  for each codon in rscu_codons do
      $codon = \frac{codonusagevaluefromCODON\_USAGE\_TABLE}{\frac{sum\_codons\_for\_corresponding\_aa}{len(codons\_for\_corresponding\_aa)}}$
  end for
  return $rscu\_codons$
```

$$W_{ij} = \frac{RSCU_{ij}}{RSCU_{iMax}} = \frac{X_{ij}}{X_{iMax}} \tag{3.3}$$

Here, $RSCU_{ij}$ is the RSCU value of the $j$th codon for the $i$th amino acid, and $RSCU_{iMax}$ is the RSCU value of the most frequently used codon for the $i$th amino acid. $X_{ij}$ is the observed frequency of the $j$th codon for the $i$th amino acid, and $X_{iMax}$ is the observed frequency of the most frequently used codon for the $i$th amino acid.

The relative adaptiveness of a codon is represented by a value that ranges from 0 to 1.0, where 0 indicates that the codon is never used for the corresponding AA and 1.0 indicates that it is the most commonly used codon. This value is normalized to ensure consistency across AAs, as the scale of RSCU values varies depending on the number of codons for each AA.

### 3.1.4 Relative codon usage frequency

Another metric that can be useful for designing optimized sequences is the frequency of each codon used to encode the same amino acid. This can be calculated from the codon usage frequency table, and provides a relative frequency value between 0 and 1 for each codon that encodes a particular amino acid. The sum of the relative frequencies of all the codons that code for the same amino acid is always equal to 1. This metric is calculated using the RSCU values according to the following equation:

$$Y_{ij} = \frac{RSCU_{ij}}{n_i} \tag{3.4}$$

Here, $n_i$ represents the number of codons (1 to 6) that encode the $i^{th}$ amino acid.

For example, the relative frequency (to its AA) of each codon was calculated on the basis of the codon usage table of the Kazusa and Hive Cut databases, but also on the basis of the calculation of this table of codon usage on 27 of the most expressed proteins in *E coli* (see figure 3.1). Although all the data come from the same strain (K12), disparities are observed. As already exposed by *Ranaghan et al* [36], slight differences appear between the two databases due to a more recent update of the table coming from Hive Cut. The most notable difference is observed with respect to the subset considering only the most highly expressed proteins. This clearly illustrates that the reference data can generate quite different tables and that it is necessary to remain cautious in the choice of these according to the desired goal.

### 3.1.5 Relative codon ranking

Another way to derive information from the codon usage table is through the relative codon ranking. This ranking indicates the order of preference (from 1 to 6, depending on the number of codons) for each codon that codes for the same AA. This ranking is determined based on the RSCU values, and there is no specific calculation involved. The general approach is to create a list containing each codon and its RSCU value, sort the list in descending order based on the RSCU values, and then assign a rank value to each codon based on its position in the sorted list.

## 3.2 Sequence optimality

As explained earlier, while these values are useful for comparing sequences, they do not provide a comprehensive picture of the variation in codon usage within a sequence. To address this limitation, this section introduces

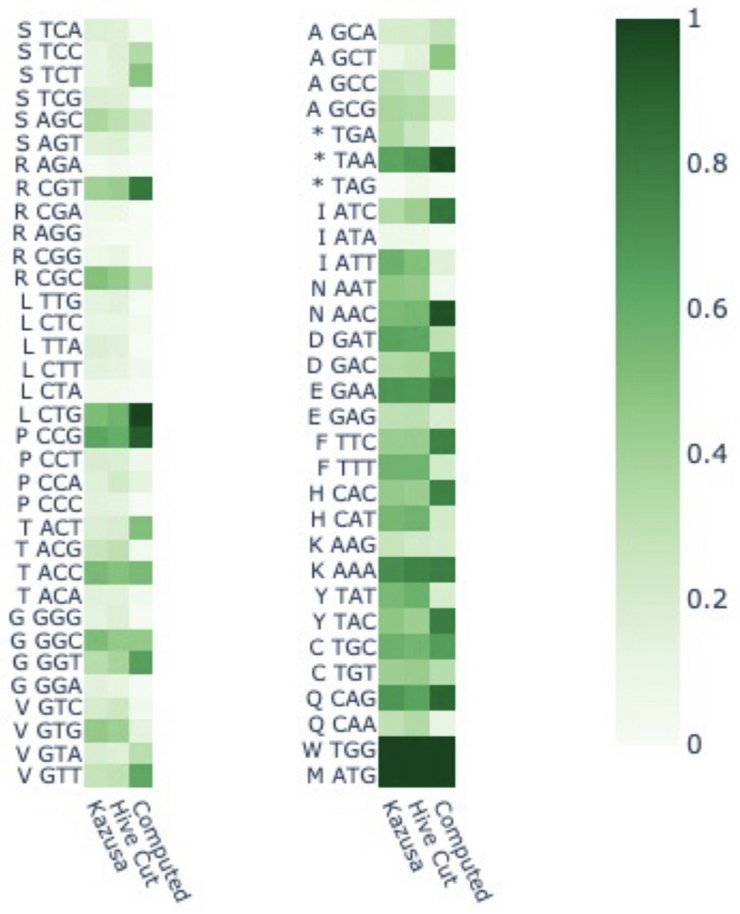


Figure 3.1: Comparison of the relative codon usage frequency for each amino acid computed from three different sources: The Kazusa codon usage table for the K12 *E coli* strain, The Hive Cut codon usage table for the K12 *E coli* strain and the computed codon usage table from the sequence of the 27 very highly expressed proteins in the K12 *E coli* strain.

the Codon Adaptation Index (CAI) as well as an algorithm called %MinMax for assessing codon optimality throughout a sequence. The CAI is an index of the optimal use of codons, while %MinMax identifies areas of translational pauses caused by the use of rare codons that facilitate protein folding. While the previous values are easy to interpret, they do not provide enough information on the variation of codon choice within a sequence.

### 3.2.1 Codon adaptation index

This index was first introduced in the same article that presented the RSCU [37]. The authors' hypothesis was that highly expressed proteins were translated more efficiently from sequences that had a strong codon bias towards certain preferred codons. They aimed to compare the codon usage in a given sequence to that of a dataset containing the sequences of the most highly expressed host proteins in order to determine the optimality of the codon choice.

To create this dataset, they selected and classified highly expressed proteins from *E. coli* in a previous publication, and computed the codon usage and RSCU tables from 27 of these genes. The resulting Codon Adaptation Index (CAI) is a value between 0 and 1, calculated as the ratio of the geometric mean of the RSCU values for each codon in the sequence to the geometric mean of the RSCU values if the optimal codon had been used. The CAI provides a measure of how close the codon usage in a sequence approaches the optimal codon usage in the dataset.

This CAI value is calculated by applying the following calculation steps:

$$CAI_{obs} = (\prod_{k=1}^{L} RSCU_k)^{\frac{1}{L}} \tag{3.5}$$

$$CAI_{max} = (\prod_{k=1}^{L} RSCU_{max})^{\frac{1}{L}} \tag{3.6}$$

$$CAI = \frac{CAI_{obs}}{CAI_{max}} \tag{3.7}$$

Where $CAI_{obs}$ is the CAI of the sequence beeing analyzed, $CAI_{max}$ is the CAI under the assumption that the preferred codon has been chosen for each AA, K is the position of the codon in the sequence and L is the length of the sequence (in term of AA/codon).

The CAI is calculated by comparing the codon usage of a given sequence with a reference dataset of highly expressed genes. The goal is to determine the degree to which each codon usage in the given sequence approaches a preferential choice in the reference dataset. However, it is important to avoid cases where a codon is never used in the reference dataset, as this would result in a CAI value of 0. To prevent this, a value of 0.5 is assigned to such codons. Additionally, the AUG and UGG codons are removed from the analysis since they are the only codons for their respective amino acids and therefore do not impact the CAI calculation. The pseudocode for the CAI algorithm 3 involves first cleaning the protein DNA sequence, then computing the observed frequency of each codon in the cleaned sequence. If the RSCU value for a codon is 0, it is replaced with 0.5 to avoid a CAI value of 0. The observed and maximum possible CAI values are then calculated using the observed and reference codon frequencies, respectively. Finally, the CAI is computed as the ratio of the observed and maximum possible CAI values.

The CAI index has a limitation in that it only provides a single value to represent a sequence, whereas studies have shown that fluctuations in the usage of different codons can impact the quality of the expressed protein. However, it is possible to overcome this limitation by using the CAI index on a sliding window locally along the sequence. This approach has been used by various studies, such as *Mignon C et al* [38] and *Wright et al* [39] in a sequence harmonization algorithm. The goal is to obtain a CAI profile of a sequence that considers the codon usage bias of the host organism. Generally, the codon usage table calculated from the entire genome of the host organism is used for this profiling.

As an example, the highly expressed protein rpoD in *E coli* was analyzed using a window of 15 consecutive codons with a sliding step of 5 codons, as proposed by *Mignon C et al* [38]. The global CAI of the protein was found to be 0.802, and the CAI profile 3.2 showed that the local CAI on each window varied around this global CAI. This observation highlights the variations in the use of preferred codons along a sequence.

**Algorithm 3** The computation of the CAI for a given sequence

```
Require: $CODON\_USAGE\_TABLE$
Require: $protein\_dna\_sequence$
  $cleaned\_protein \leftarrow \{remove\_single\_codons(protein\_dna\_sequence)\}$
  $freq\_obs \leftarrow []$
  for each codon in cleaned_protein do
      if codon rscu value $> 0$ then
          freq_obs.append(codon rscu value)
      else
          $freq\_obs.append(0.5)$
      end if
  end for
  $cai\_obs \leftarrow prod(freq\_obs)^{\frac{1}{len(freq\_obs)}}$
  $freq\_max \leftarrow []$
  for each codon in cleaned_protein do
      $freq\_max.append(preferred_codon_rscu_value)$
  end for
  $cai\_max \leftarrow prod(freq\_max)^{\frac{1}{len(freq\_max)}}$
  return $\frac{cai\_obs}{cai\_max}$
```

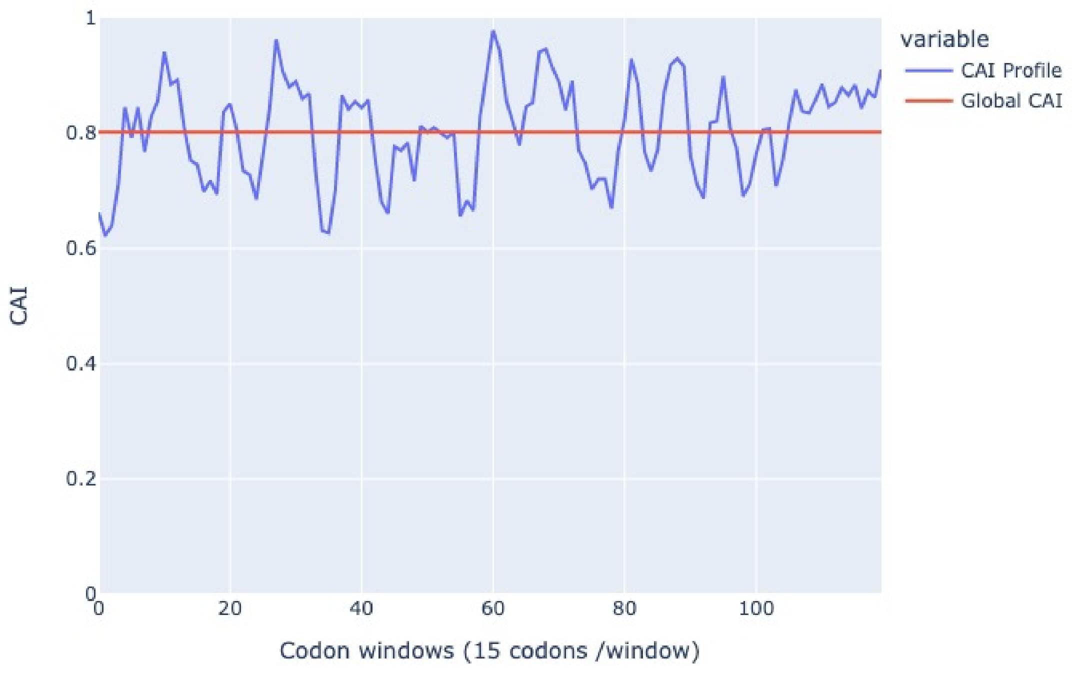


Figure 3.2: The CAI profile of the rpoD protein over a window size of 15 codons with a sliding step of 5 codons.

### 3.2.2 Rare codons clusters - %MinMax

In this new trend to monitor local fluctuations in codon usage choices that are predictive of translation kinetics, another tool has emerged: %MinMax [40] [41]. It is calculated on the basis of the codon usage table, but also uses the relative preference of each codon for its amino acid. It compares a given sequence to hypothetical sequences coding for the same protein but still using the preferred codon (max) or the rarest codon (min) according to the value of a sequence window compared to the average frequency of use of all codons coding for the same amino acid. In other words, a sliding window (between 5 and 30 codons) of codon is evaluated to allow a scoring, if the sum of use of each codon is higher (respectively lower) than the sum of the averages of frequency of use of the codon for the same amino acid, the algorithm generates a positive (negative) value which is relative to the use of the preferential (rare) codon. It is thus possible to create a graph as before where clusters of predominant codons (%Max) have a value between 0 and 100 and clusters of rarer codons (%Min) have a value between 0 and -100. A value of 100 therefore represents a window that would use the preferred codon every time. A value of 0 can be obtained for example in the case where two codons (one rare and one preferential) code for an amino acid. If, for a window, these two codons are used in equal proportion, the result for the window will be 0.

This tool was initially created to assess natural sequences. The authors examined known ORFeomes from various species and found that rare codon clusters are present throughout genes that are typically expressed by the host, even in highly expressed ones such as ribosomal proteins. This suggests that slower-translated regions are necessary even in highly expressed proteins.

The computational process, as outlined in the original papers, is explained below. For each *jth* codon of the *ith* amino acid in the window with $n$ synonymous codons, the algorithm computes the difference between the frequency of usage of that codon and the average frequency of the synonymous codons for the same amino acid. This value is then divided by the difference between the maximum or minimum codon frequency and the average codon frequency.

$$X_{avg,i} = \frac{1}{n_i}\sum_{j=1}^{n_i} X_{ij} \tag{3.8}$$

$$If \sum_{i=1}^{z} X_{ij} > \sum_{i=1}^{z} X_{avg,i} \ then \ \%Max \ = \ \frac{\sum_{i=1}^{z}(X_{ij} - X_{avg,i})}{\sum_{i=1}^{z}(X_{max,i} - X_{avg,i})} \tag{3.9}$$

$$Else \ \%Min \ = \ \frac{\sum_{i=1}^{z}(X_{avg,i} - X_{ij})}{\sum_{i=1}^{z}(X_{avg,i} - X_{min,i})} \tag{3.10}$$

Where X is calculated over a sliding window *z*. The output is always between 0 and 100, but the *Min* values are reported as negative values. So, for a given sequence the output is a list of values (between -100 and 100) over a sliding window (0 to z, 1 to z+1, ...). As theses values express the %MinMax usage of a window, it is placed at the index position of the central codon. It therefore impossible to compute values for the first positions before this central value as well as for the last positions. The list of values is then filled with “0” (the window size divided by 2) for the first and last positions.

This computational tool has found a wide range of applications. For instance, researchers have employed it to investigate whether rare codons are enriched at the ends (5’ and 3’) of sequences and whether this correlates with weaker secondary structure of the mRNA [42]. Their findings indicate that native host proteins indeed contain clusters of rare codons at the ends, particularly on the 5’ side, in the region between 25 and 50 codons after the start codon for proteins that are secreted out of the cell. Since this region is the initiation of translation, it is not conducive to the co-translational folding of the nascent protein, which is reduced in size. However, the rare codon area may enable better ribosome pairing and proper spacing of the ribosomes along the strand, as multiple ribosomes read mRNA simultaneously [29]. Slowing their progression at the beginning of the sequence helps avoid traffic jams further along the strand. As there is no correlation between this rare codon area and the level of secondary structure, it may be possible to design a sequence that is optimal for both parameters by favoring rare codons and low secondary structure in this 25-50 codon length region.

Although CAI correlates positively with total protein expression, it does not ensure its solubility. *Pellizza et al* [43] developed two new parameters based on the %MinMax algorithm: the Δ%MinMax and the MinMax Correlation. The Δ%MinMax is the sum of the absolute value of the difference between each %MinMax value of a sequence analyzed according to the initial and destination host, divided by the number of windows analyzed. This gives an idea of the overall difference in codon usage between the starting host and the destination host.

This parameter is negatively correlated with the expression level. In other words, the lower it is, the higher the expression of the protein will be. But no correlation could be found with the solubility.

$$\Delta\%MinMax = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n} \tag{3.11}$$

For some proteins, the %MinMax profile determined according to the codon usage bias of two different hosts can be quite well conserved. And when this is the case, the protein expressed in a heterologous host is highly soluble. From this observation, a correlation index between the MinMax profiles of the protein expressed in two cellular hosts was born. The experimental values of synthesis were able to show that, for the analyzed proteins, a positive correlation was observed between this solubility and the %MinMax correlation. In other words, the more similar the profiles are, the more likely the protein is to be expressed in its soluble form in the destination host. This %MinMax correlation is none other than the Pearson correlation whose formula is given here after.

$$correlation\ MinMax = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.12}$$

These two values were used in a context where the goal was to determine if a protein from one host had a high probability of being expressed in soluble form in another host based on codon usage tables only. A low $\Delta$%MinMax indicates a low frequency of codon usage and therefore a similar level of expression to the original one while the correlation allows to evaluate the probability of the protein to be expressed in its soluble form. This will allow us to evaluate the performance of optimization algorithms where we will try to minimize the $\Delta$%MinMax while maximizing the MinMax correlation. For each algorithm used, these values will be calculated to try to determine which one is the most optimal.

### 3.2.3 What is the best metric?

Of these different tools and observations it can be difficult to determine which is the most optimal. Should the CUB be calculated on an entire ORFeome, or only on a data set of highly expressed proteins? Which of the CAI profile or %MinMax is better at predicting areas of translation slowdown? What is the optimal window size for computing these profiles?

*Wright et al* [44] were able to confirm the results obtained in two other studies with respect to the choice of dataset for the calculation of codon usage bias. The different models for analyzing slowdowns calculated on the basis of the two types of datasets are strongly correlated with laboratory data that count the number of ribosomes present at specific locations (the presence of a large number of ribosomes indicating an area of translational slowdown).

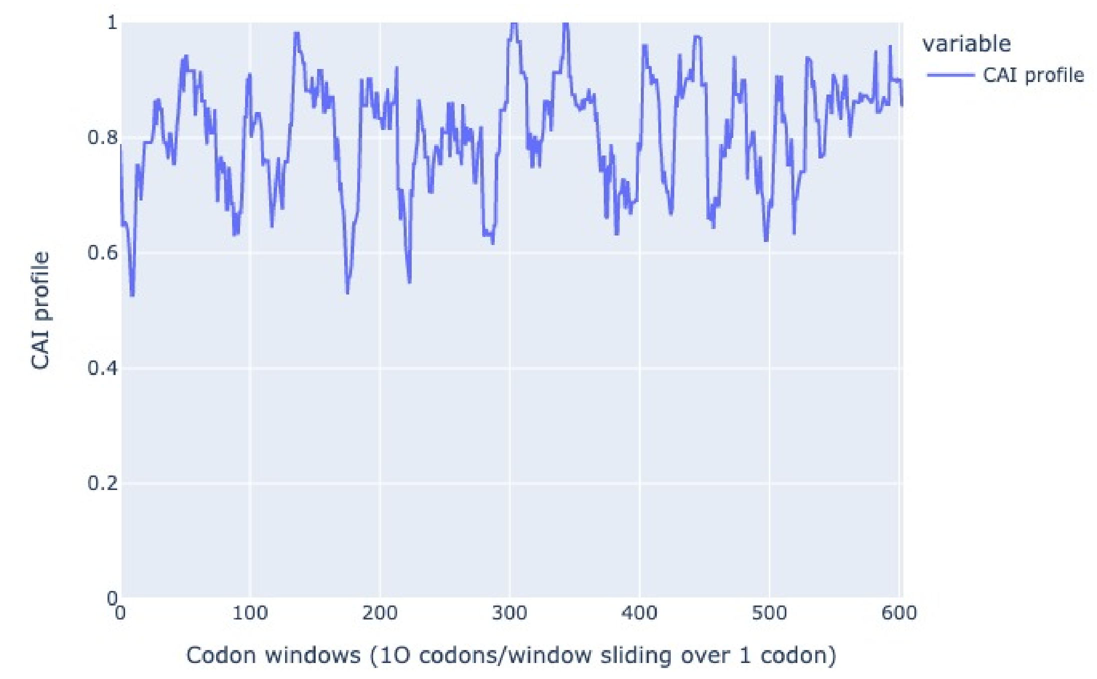
They were also able to determine that the optimal sliding window size was 10 codons, which roughly corresponds to the number of codons that are "hidden" by a ribosome during reading (the ribosome footprint).

There is no certainty as to which index will best predict whether a sequence is well suited for expression in a heterologous host or not. CAI and %MinMax were chosen because the first one, despite its age, is still widely used and because it correlates with the expression level, and the second one has shown its proofs in the prediction of sequences allowing co-translational folds. The main difference lies in their scale (CAI: 0-1; %MinMax: -100-100) which implies that the CAI index will tend to compress the differences between codons that are used less frequently than the average for the same AA. The authors of %MinMax are also the creators of a sequence harmonization algorithm which is nevertheless implemented with both types of indices. This reveals the difficulty of making a choice.

An example of the use of these two algorithms (CAI profile and %MinMax) is shown in Figure 3.3. This is the rpoD protein already used as an example previously analyzed according to these two algorithms using a codon usage table of the *E coli* protein (strain K12) from the HIVE-CUT database.

When profiling tools are used, the following settings will be applied:

- Use of the full ORFeome for the calculation of the codon usage table of a species (this is available via the HIVE-CUT database without having to search for the most highly expressed proteins)
- A sliding window of 10 codons
- A codon offset of 1 codon to read the sequence
- An implementation using %MinMax



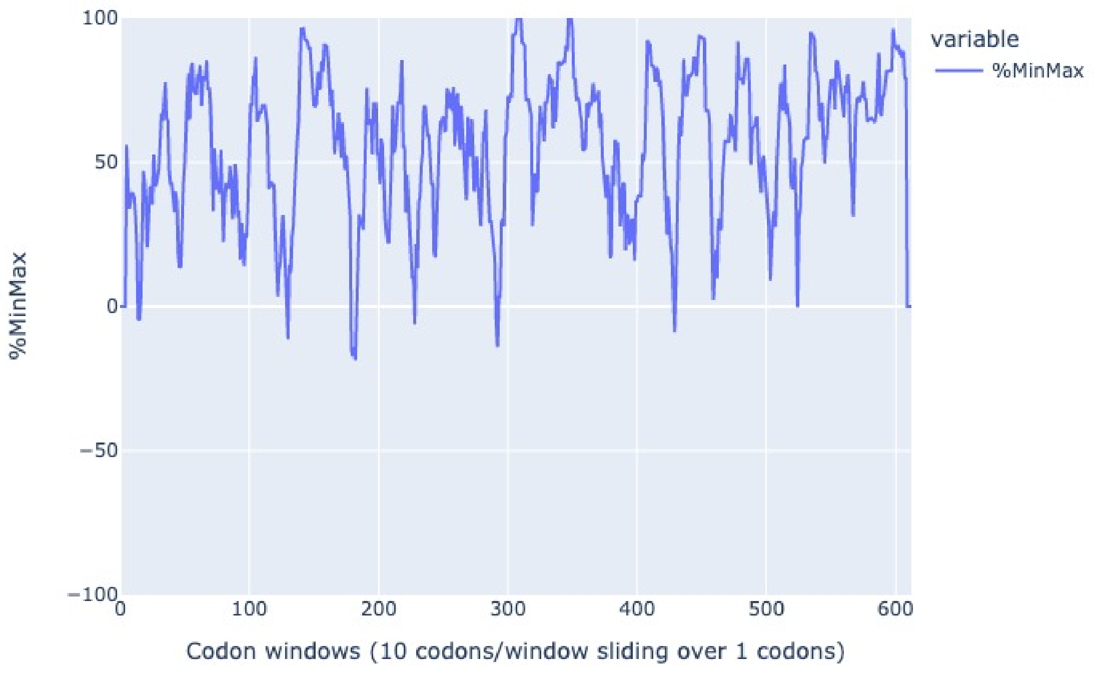


Figure 3.3: The CAI profile and %MinMax of the rpoD protein over a window size of 10 codons with a sliding step of 1 codon analysed with the Ecoli-K12 codon usage table from HIVE-CUT.

| Sequence | Host | CAI | Δ%MinMax | correlation MinMax |
|---|---|---|---|---|
| Wild type | *E coli* | 0.782 | 0.0 | 1.0 |
| Wild type | *S cer* | 0.624 | 44.6 | 0.36 |
| Optimized | *S cer* | 1.0 | 48.2 | 0.37 |

Table 3.2: Comparison of the wild type sequence of the rpOD protein expressed in *E coli* (the origin host) and *S cer* versus an optimized version for *S cer*.

## 3.3 RNA sequence design using codon bias values

Although the process of synthesizing recombinant proteins is well understood, designing an optimal sequence for achieving high expression of soluble proteins is still an area of uncertainty. It is crucial to adapt the mRNA coding region to suit the characteristics of the target host. However, authors who have successfully synthesized recombinant proteins do not always publish their sequence design methodology. Commercial services that offer optimized sequences also keep their software strategies secret, leading to black-box algorithms. Unfortunately, the results obtained from these services often do not meet scientists' expectations, with most of them using preferred codon optimization without any guarantee of success [36].

### 3.3.1 Optimization

In the past, efforts to modify sequences for expression in a specific host organism primarily focused on the codon usage bias of the host. Since highly expressed proteins exhibit a strong preference for certain codons (as reflected by a high CAI), the initial approach was to use the host's preferred codon for each amino acid, following a "one amino acid - one codon" strategy. While we could not locate the specific implementation of this optimization, the process itself is relatively straightforward. It involves obtaining the DNA or RNA sequence of the protein to be optimized for the new host, as well as a table that maps each amino acid to its most commonly used codon. This information can be readily obtained from a codon usage table, either from a database or by computing it on a specific set of genes.

The steps to generate such an optimized sequence can be summarized as follows:

- Create a dictionary which key-value pairs are: each AA - preferred codon for this AA
- Convert the protein nucleotide sequence in an AA sequence
- Create an empty string that will contain the optimized sequence
- For each AA in the protein AA sequence, append the corresponding preferred codon to the optimized sequence string

To verify that the sequence has been optimized, the CAI tool can be used. Indeed, since for all the AA, the preferred codon has been chosen, the value of the CAI must be maximal (i.e. equal to 1.0). This simple algorithm has been implemented in python and its validity has been verified by optimizing 7 proteins and calculating the CAI for each of the optimized sequences. As expected, the value returned for each optimized protein is 1.0.

However, such an approach is not synonymous with success. Although highly expressed proteins show codon usage bias and preferential codon choice, and thus high CAI, the opposite is not always true. A fully optimized sequence will not necessarily be strongly expressed. And if it is, it will not necessarily be in its native and soluble three-dimensional form.

The %MinMax profiles of the native sequence of the first 200 AAs of one of these proteins (rpoD) expressed in its original host (*E coli*) and in a destination host (*Saccharomyces cerevisiae*) as well as the version optimized for the latter host were calculated in order to measure the performance of the algorithm from the point of view of Δ%MinMax and correlation MinMax as well as the CAI. The profile is shown in figure 3.4 and the values are summarized in the table 3.2.

As expected, the CAI of the wild-type sequence expressed in the destination host is lower than in the original host while it is maximal for the optimized version. Nevertheless, the Δ%MinMax indicates a difference (for the wild type and the optimized version) and the correlation between the MinMax profile of the wild type expressed in *E coli* and the optimized version remains quite low. These observations inform us that there is a low probability that the protein will be expressed (if it is expressed) in a soluble form.

A recent study by Konczal et al. (2019) [31] investigated the expression levels and solubility of small human proteins (less than 200 amino acids) in *E. coli* using different optimization strategies. The study explored the impact of using alternative T7 transcription promoters, alternative start codons, and reintroducing suboptimal

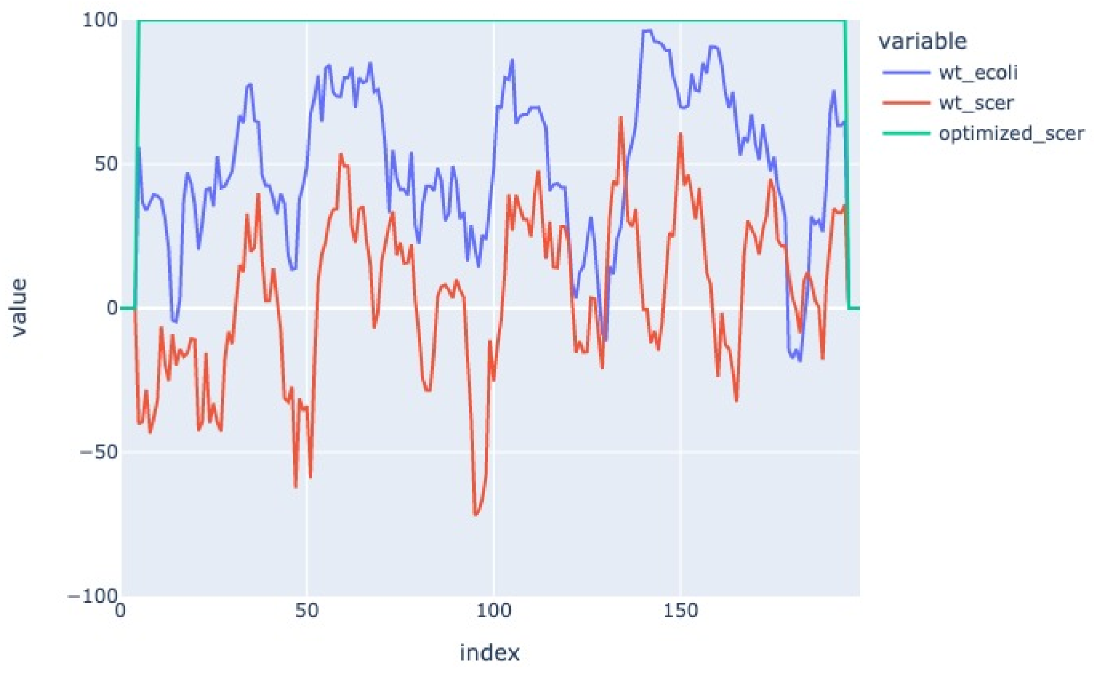


Figure 3.4: The %MinMax of the 200 firsts AAs of the rpoD protein over a window size of 10 codons with a sliding step of 1 codon. The %MinMax has been computed with the *Ecoli-K12* codon usage table for the wt_ecoli and the *S cer* for the wt_scer and optimized_ser. The profile of the native sequence expressed in *S cer* (wt_scer) is lower than that of the same sequence expressed in *E coli* (wt_ecoli) while the optimized version (optimized_scer) is much higher. The profiles of the different sequences are nevertheless very different from each other.

codons into a fully optimized sequence that only used the most frequent codons in *E. coli.* The results showed that the fully optimized sequences had higher productivity than the original host sequences, but most of the protein synthesized was insoluble, thereby reducing net productivity. The use of an alternative promoter reduced production, while modifying the classic AUG start codon resulted in reduced synthesis and changes in amino acid composition. Modifying the codon coding for arginine, isoleucine, or leucine by their least used alternative in *E. coli* at different positions increased the net gain of soluble proteins, but the number of modifications and their location varied among the three proteins. The study found that substituting a single codon at the end of the sequence increased the benefit, likely allowing more time for the protein to fold before translation termination. However, the determinants of maximum synthesis of soluble proteins could not be deduced from the study, and the results suggested the importance of preserving suboptimal codons for enhancing net production of the protein of interest in its soluble form. Other approaches have emerged to optimize codon usage, not solely based on preferential codons.

### 3.3.2 Randomization

In 2011, Menzella [45] conducted a study to compare two strategies aimed at enhancing the production of calf prochymosin in *E. coli* for industrial cheese making. The wild type sequence already had a high Codon Adaptation Index (CAI), but there was still room for optimization. The two strategies tested were one AA - one codon and codon randomization, which involved taking into account the frequency distribution of codons for each amino acid in an entire genome or subset of coding sequences, and then transposing this information to the desired sequence. The study found that the one AA - one codon strategy decreased prochymosin synthesis compared to the wild type, whereas codon randomization improved production from 21 to 71%. Six optimized sequences were tested with CAI values between 0.70 and 0.74, and no correlation was found between CAI and protein quantity or activity after refolding. The study suggests that reintroducing rare codons and mimicking the codon usage of the destination host may be a better approach than full optimization.

In order to generate the optimized sequences, a software package was used: Gene Morphing System (GeMS) [46]. This contains, among other tools, a randomization procedure, but it is no longer available. In addition than taking into account the frequencies of use of each codon (as inspired by [47] and [48]), the authors allowed certain codons to be excluded if their frequency of use was too low in the destination host (based on an arbitrary or user-encoded value) or to assign a higher weight to certain codons. The aim was to exclude codons that were too weakly used, which could cause a break in synthesis and therefore generate a truncated protein. If certain codons were excluded, a recalculation on the basis of those which were conserved was carried out in order to keep a sum of frequencies equal to 1.

A simplified version of this algorithm (not excluding very infrequent codons and without using weights) is proposed based on the article linked to the software package. It requires as input a codon usage table (generally based on the whole genome) and the DNA sequence of the protein of interest. The usage table is first transformed into usage frequency relative to each AA and the sequence transformed into an AA chain. The number of each AA thereof is counted and a codon pool respecting the frequency of use of each codon for each AA is generated. Depending on the specific number of AA present in the protein sequence, it is not always possible to respect the frequencies of use of each codon. Indeed, it is not always possible to obtain an integer codon values based on the frequencies and the total number of codons obtained may be less than the number of AAs required. The articles do not detail how to handle such a case. The choice was made to handle this as follows: the codon pool for a specific AA is completed with the codon having the highest frequency of use. This pool is then shuffled randomly. Finally, the protein's AA sequence is scanned to build a list of codons. For each amino acid, a codon is removed from the corresponding pool and added to the codon list being created.

This approach aims to optimize the codon usage of the protein sequence to mimic that of the host organism, which has been shown to increase productivity in some cases [38]. However, this method has two major criticisms. Firstly, it does not take into account local fluctuations in codon usage that can cause translational pauses along the sequence. Secondly, the randomized nature of the algorithm means that each call can potentially generate a different sequence, making it difficult to search for an optimal solution. Generating all possible combinations while respecting the frequency of use of each codon and testing them is not practical.

To illustrate this algorithm, the coding sequence of the 200 first AA of the rpoD protein from the bacterium *E coli* was randomized two times using the codon usage table of the yeast Saccharomyces cerevisiae (*S cer*). This demonstration is performed on 200 AAs only for visibility reasons for the graphic representation. Figure 3.5 shows the %MinMax profiles over sliding windows of 10 codons in steps of 1 of the native sequence and 2 randomized sequences. The dotted line represents the %MinMax profile of the native sequence when it is expressed in the host of origin (*E coli*), this same sequence expressed in the host of destination is the continuous black line.

It clearly appears that the sequence of origin has a lower CAI profile when expressed in the host of destination (global CAI on the 200 first AAs = 0.624) as compared to the native host (global CAI on the 200 first AAs

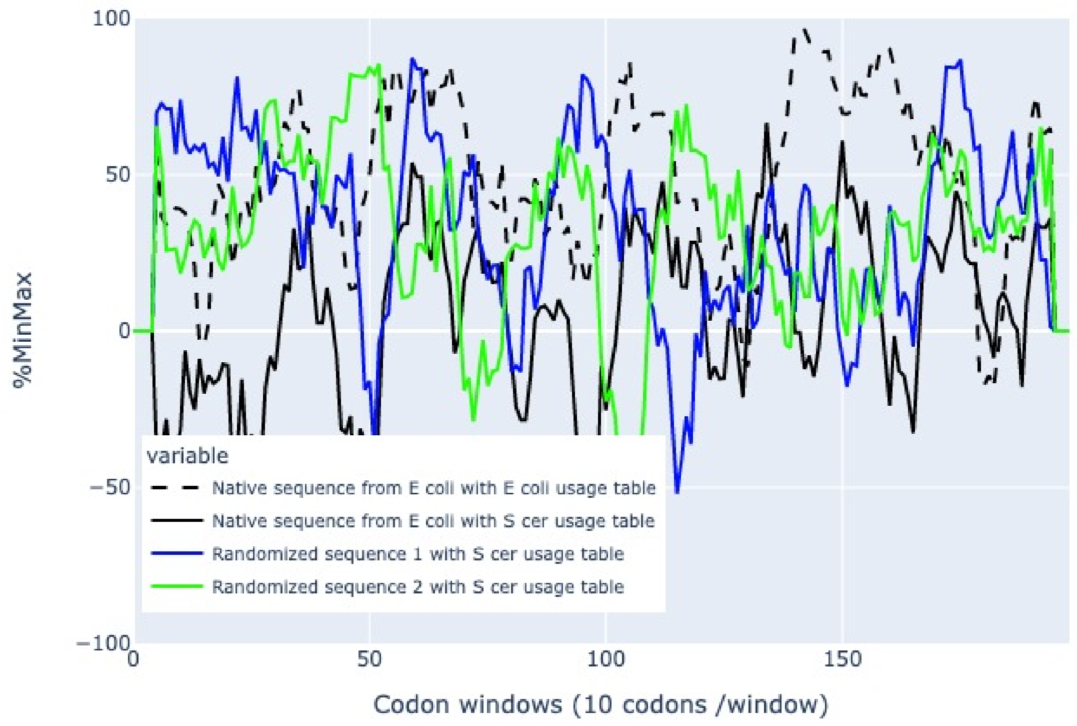


Figure 3.5: Example of the randomization algorithm performed 2 times on the 200 firsts AA of the rpoD protein originating from *E coli* taking into account the relative frequency of use of each codon of the yeast Saccharomyces cerevisiae.

| Sequence | Host | CAI | Δ%MinMax | correlation MinMax |
|---|---|---|---|---|
| Wild type | *E coli* | 0.782 | 0.0 | 1.0 |
| Wild type | *S cer* | 0.624 | 44.6 | 0.36 |
| Randomized_1 | *S cer* | 0.734 | 34.9 | -0.02 |
| Randomized_2 | *S cer* | 0.734 | 33.5 | -0.10 |

Table 3.3: Comparison of the wild type sequence of the rpoD protein expressed in *E coli* (the origin host) and *S cer* versus two randomized versions for *S cer*.

= 0.782). The 2 randomized sequences taking into account *S cer* make it possible to raise the overall level of the %MinMax profile (with a global CAI equal to 0.734 for both). As expected, the 2 sequences generated are different in their profile altough they have an equal CAI, but they do not take into account the codon usage profile of the original sequence.

As for the optimization algorithm, the %MinMax profiles were analyzed using Δ %MinMax and MinMax correlation. The summary of these calculations is given in table 3.3 .These values indicate that although the CAI is higher for the randomized versions than for the wild-type version expressed in *S cer* (which is thought to indicate that the expression level should be higher), the MinMax correlation with respect to the wild-type sequence expressed in *E coli* is worse for the randomized versions than for the wild-type version expressed in *S cer*. This alerts to the fact that these sequences do not respect the fluctuations in choice of use of the initial host codon.

### 3.3.3 Harmonization

Since codon usage frequency seems to be correlated with translation kinetics, other approaches attempt to transpose this parameter from one organism to another. In other words, the goal is to harmonize the usage of different codons in a destination host with the frequency of codon usage observed on the wild-type sequence in the original host but considering the fluctuations along the sequence rather then taking it globally.

It might be tempting to test all possible codon combinations in order to maximize the MinMax correlation and thus harmonize the sequence as close as possible to the original host. However, this approach is not possible in practice. Indeed, the average length of a protein in E coli is about 300 AA and the average number of codons coding for each of them is 3.05. It would therefore be necessary to test $3.05^{3}00$ (or $2x10^{1}45$) sequences, which is not possible in practice. The other argument in favor of such an approach is that the sequences are usually explored on a sliding window and therefore different optimal sequences could result. It is therefore necessary to find other more realistic approaches which make it possible to approach the optimum (defined as a maximum MinMax correlation with respect to the expression of the native sequence in the original host) [39].

This new tendency to mimic the behavior of the original host was explored semi-empirically by *Angov et al* in 2008 [49] which resulted in a 4 to 1000 fold increase in the expression of a protein in its soluble and active form in *E coli* in some cases.

It is based on the idea that regions of complex structures at the protein level (alpha helices, beta sheets) are translated quickly and separated by regions of less structured links that are translated more slowly, that this first algorithm was proposed. These linkage regions should allow the more complex structures to fold before the translation of other interfering elements takes place. It is therefore necessary to be able to identify and recode the linkage regions as well as other areas of the wild-type gene.

The proposed algorithm focuses only on 10 amino acids that appear more frequently in the unstructured linking regions (Tyr, His, Trp, Ile, Leu, Val, Ser, Thr, Pro and Cys) in *E coli* and consists of an identification of codons to be modified, followed by a 3-step iteration. The goal of this iteration is to modify the frequency threshold identifying a codon as rare to reach a supposed optimal number of rare codons clusters. This would be calculated as the number of amino acids divided by 30 in *E coli* as demonstrated empirically by *Thanaraj et al* [50]. This algorithm is followed by a last step which harmonizes the elements not being in these zones. The goal here is to first determine the frequency considered as the threshold to declare a codon as rare. This threshold is defined as the frequency allowing to obtain a number of clusters (rare codons belonging to the same group) equal to the value obtained by the formula determined empirically for *E Coli*. Once this frequency is determined, codons identified as rare and belonging to the binding domains with a frequency lower than or equal to this threshold frequency must be modified considering the codon usage frequency of the destination host. Once this step is done, all the other codons are harmonized trying to come as close as possible to the frequency of use observed in the original host. This is the main difference, the codons identified cannot have a frequency higher than the threshold while the others are harmonized with a frequency that can be higher than this threshold in some cases.

This algorithm showed good results in this study where the goal was to increase the production of a protein from *Plasmodium falciparum* to *E coli* in a malaria vaccination program. Nevertheless, it remains semi-empirical since it only takes into account 10 AAs supposedly predominant in the binding regions and assumes that the number of these regions would be the number defined by the following rule: number of codons divided by 30. However, this takes into account an average observed on proteins expressed in *E coli*; a protein coming from another organism could present a different number of clusters, which would have to be transposed in *E coli*. The following sections will explore other algorithms that attempt to approximate the codon usage fluctuations of the original host by considering all AAs and without taking into account an arbitrary number of clusters.

### Codon ranking method (Rodriguez harmonization)

To adjust a sequence to the codon usage bias of a different host organism, one approach is to match the rank of each amino acid's codons based on their frequency of usage in the original host. This involves taking into account the codon usage table of the destination host and potentially modifying the codons accordingly. This method, known as the Rodriguez initialization, was introduced by *Rodriguez et al.* [41] and has been used as an initial step in more complex algorithms. However, this approach does not guarantee an exact match with the %MinMax or CAI profiles, as the rank of two codons may be equivalent but their frequencies may differ, which can affect the calculated indices. For instance, the most frequently used codon for asparagus in *E coli* is 'CGC', whereas in *S cer*, it is 'AGA'. Although these codons have the same rank, their relative frequency of usage is different, resulting in different

Below is the pseudocode for the algorithm 4 that is used to harmonize the codon usage of a protein. It takes as input the protein sequence to be optimized, as well as the codon usage tables of the origin host and the destination host. The algorithm reads the original sequence from the origin host and evaluates all the possible codons that can code for the same amino acid. It then searches for the codon with the same rank in the destinations. The selected codon is then added to the harmonized sequence. This process is repeated for all the codons in the original sequence, and the resulting sequence with harmonized codon usage is returned.

**Algorithm 4** The harmonization based on the codon ranking for a given sequence

```
Require: $ECOLI\_CODON\_RANKING\_TABLE$
Require: $SCER\_CODON\_RANKING\_TABLE$
Require: $protein\_dna\_sequence$
  $harmonized\_sequence \leftarrow$ ""                    ▷ Initialization to an empty string
  for each codon in protein_dna_sequence do
    $aa \leftarrow CODON\_TO\_AA\_DICT[codon]$
    $possible\_codons \leftarrow AA\_TO\_CODON\_DICT[aa]$
    for cod in possible_codons do
      if ECOLI_CODON_RANKING_TABLE[cod]
            == SCER_CODON_RANKING_TABLE[cod] then
        $harmonized\_sequence.append(cod)$
      end if
    end for
  end for
  return $harmonized\_sequence$
```

To illustrate this algorithm, the rpoD protein from *E coli* was again used to be expressed in *S cer*. For readability purposes, only the first 200 AAs were used. The %MinMax profiles of the native protein expressed in its native and destination host as well as the harmonized version are presented in figure 3.6. It is clear that the harmonized version is close to the native version expressed in its native host but that the correlation is not perfect. The CAI of the wild-type sequence in the native host is 0.782 and 0.624 in *S cer*; for the harmonized version, it is 0.751. This results in a sequence that optimizes the CAI for the host and accounts for local fluctuations in codon usage. These two criteria were identified as determinants for synthesis (CAI for expression level and local fluctuation for solubility).To confirm these observations, the delta MinMax is 44.6 for the wild type expressed in *S cer* compared to *E coli*. On the other hand, it is only 14.9 for the harmonized version. As a reminder, the lower this value is, the closer we should be to the expression level of the starting host. The MinMax correlation was also calculated on the same profiles. This value is 0.36 for the wild type sequence expressed in *S cer* compared to *E coli*. The harmonized version has a correlation of 0.88. This being higher, there is a greater probability that it will be soluble in the destination host. All these values are summarized in the table 3.4.

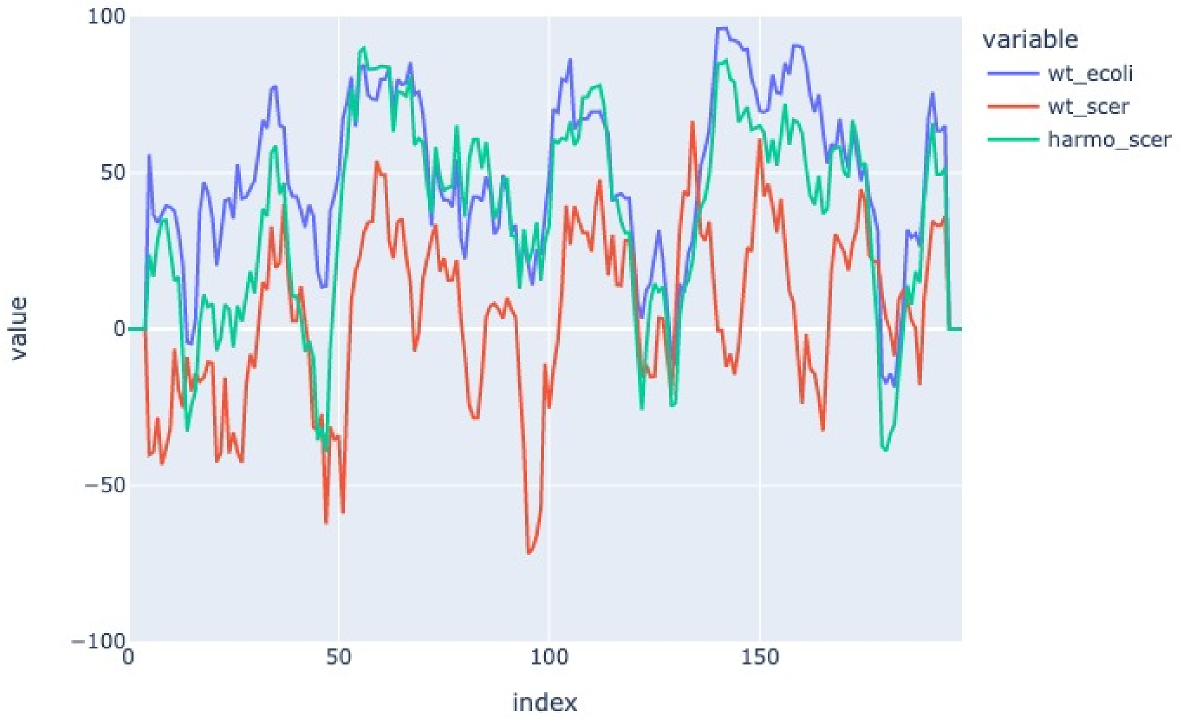


Figure 3.6: Example of the Rodriguez harmonization algorithm performed on the 200 firsts AA of the rpoD protein originating from *E coli* taking into account the relative frequency of use of each codon of the yeast Saccharomyces cerevisiae.

| Sequence | Host | CAI | Δ%MinMax | correlation MinMax |
|---|---|---|---|---|
| Wild type | *E coli* | 0.782 | 0.0 | 1.0 |
| Wild type | *S cer* | 0.624 | 44.6 | 0.36 |
| Harmonized | *S cer* | 0.751 | 14.9 | 0.88 |

Table 3.4: Comparison of the wild type sequence of the rpoD protein expressed in *E coli* (the origin host) and *S cer* and one harmonized version for *S cer* using the Rodriguez harmonization method.

### Codon HARmonizING (CHARMING)

The previously exposed algorithm allows to approximate the local fluctuations of codon usage, which is clearly visible on the graph of the %MinMax profile with respect to the native sequence when expressed in another host. This is also highlighted by the correlation index between the %MinMax profiles. Although this is close the native host, it is still imperfect and another algorithm to better mimic this profile has been proposed.

*Wright et al* [39] recently proposed an algorithm that tries to approximate the original sequence as closely as possible: CHARMING for Codon HARmonizING). Since the index making it possible to generate an optimal sequence has not yet been established, they left the door open between the choice of the two currently most used: CAI and %MinMax. The implementation is open source and is made available through a dedicated site [3] or a GitHub repository [4] which facilitates the analysis. The web version is nevertheless limited to sequences of a maximum of 350 AAs and only allows the generation of 3 variations of harmonized sequences.

Six parameters (+ 1 optional) are required to call the algorithm:

1. The original sequence to optimize (a DNA sequence with a length mutiliple of 3)
2. The number of desired output sequence (an interger > 0)
3. The size of the sliding window to characterize the sequence
4. The desired codon calculator (CAI or %MinMax)
5. The name of the output file which will be created to write the harmonized sequence
6. The codon usage table of the host of origin
7. (Optional) The codon usage of the destination host if the intent is to harmonize the sequence to another species

This is a 7-step iterative process that is deterministic, but since the codon usage profile is evaluated on a rolling window, it is possible for several different sequences to correlate identically to the original sequence in the initial host. It may be interesting to generate some of these sequences to explore the different possibilities since although all these algorithms are useful, the final experimentation in the laboratory remains the final decision maker of the quality of the generated sequence.

The algorithm initially generates random sequences, which are used to produce a variety of different sequences. The output sequences vary depending on the initial random sequence used. The algorithm tests 10 times more sequences than the desired number of output sequences, and then selects the best ones. During the initialization step, one of the sequences may use the Rodriguez ranking algorithm that was previously described. This helps to reduce the number of iterations needed to reach an optimum solution. This is a typical approach for a genetic algorithm, as an exhaustive search is not feasible due to the large search space. Successive "jumps" are made to approach the optimum solution, without certainty of reaching it, but ensuring an approximate answer within a reasonable time. In the case of CHARMING, a set of random solutions is proposed, and a score is assigned to each. The n best solutions (depending on the desired number of outputs) are then retained after a scoring optimization step.

The algorithm first creates random alternative sequences then applies the following steps:

1. (Optional). This first step uses the Rodriguez harmonization based on the codon ranking as previously described. By definition, it will always generate the same sequence based on the codon usages of both hosts. For this reason it can only be used on only one of the random sequence generated. It is not suitable to explore harmonized alternatives of a sequence, but it improves the harmonization process of an average of 10.5% and accelerate the processing. It is nevertheless not suitable in two cases: when several output sequences are requested (in this case, only one sequence will use the initial step), when trying to create an alternative sequence for the host of origin.
2. The codon usage index (CAI profile or %MinMax) is then calculated on the current sequence using the destination host's codon usage table. This profile is calculated on the basis of a sliding window value introduced by the user (the recommended value being 10 to correspond to the footprint of a ribosome) which shifts by one codon. These values are then compared to the original sequence measured in the initial host.

[3]http://www.codons.org/codons.html
[4]https://github.com/wrightgs/CHARMING

3. The algorithm then identifies regions called consecutive deviation region (CDR) of more than 10 codons where the difference between origin and destination is consistently positive or negative. Then the candidates sections within a CDR is(are) selected as follows: the number is the CDR length/10 rounded to the nearest integer, these candidate regions are sparsed evenly on the CDR section and is composed of 5 codons (for example, if a CDR is 20 codons long, the number of candidate sections would be 2 and they will be at positions 6-10 and 16-20 in the CDR respectively).

4. Based on the iteration step, a codon is selected (iteration number modulo 5) for modification.

5. For CDRs that are systematically above (bellow) the target, the identified codon is replaced by it's alternative with the next lowest (highest) rank for the same amino acid. If it is already the lowest (highest) codon, it is not replaced and the algorithm continues to step 6.

6. The algorithm then calculates the net deviation from the target and estimates if their is improvement or not (if the deviation is reduced); if it is the case, the alteration is kept, otherwise, the codon is restored. The net deviation is computed as follows:

   The codon measure (CAI or MinMax) is calculated for the altered sequence with the destination host codon usage table over the same sliding window then the initial sequence/host. Then the deviation is simply the sum of the absolute of the difference between the target and the harmonized sequence:

$$Net\ deviation\ from\ target = \sum_{i=0}^{Sequence\ length} |Target\ value_i - Harmonized\ value_i| \qquad (3.13)$$

7. Once steps 4-6 have been performed for each CDR and each candidate positions within them, the iteration number is incremented. If no improvements are made five times in a row, the algorithm terminate, goes to the next sequence to optimize or output the result

Based on this implementation, the algorithm will always return an optimized protein following the initialization of Rodiguez which will always be identical at each call, the following sequences will be different since they are generated on the basis of randomized sequences without taking into account the frequencies of codon usage. The program actually multiplies the number of desired sequences by 10, sorts them in descending order according to the calculated Pearson correlation with respect to the target and returns only the best sequences according to the desired number.

The test sequence used so far (the first 200 codons of the rpoD protein naturally expressed in *E coli*) was subjected to this algorithm in order to generate three harmonized sequences. The parameters used were as followed:

- Initial codon usage table: *E coli*
- Destination codon usage table: *S cer*
- Codon profile measure: %MinMax
- Window size for codon profile: 10
- Number of output: 3
- Rodriguez initialization (with codon rank): Yes for the first random sequence

The graph of the native sequence expressed in *E coli*, this same native sequence as well as the harmonized versions expressed in *S cer* is presented in figure 3.7, the values of CAI, correlation %MinMax and Δ%MinMax are reported in table 3.5.This clearly indicates that the algorithm that uses the initialization achieves a very good correlation with the native sequence of the original host.

Another approach to create randomized alternative sequences is to use the previously described algorithm which allows to take into account the frequency of use of each codon to be carried over to the sequence under construction rather than starting from a totally random sequence. The CAI obtained is thus closer to that of the original host, the delta is lower and the correlation higher. However, this does not exceed the values obtained using Rodriguez's initailization.

According to the optimality criteria allowing the production of a protein in a heterologous host in a soluble manner as defined previously, this algorithm when preceded by Rodriguez initialization gives the best results. It is, at the time of writing, the most efficient solution that has been found.

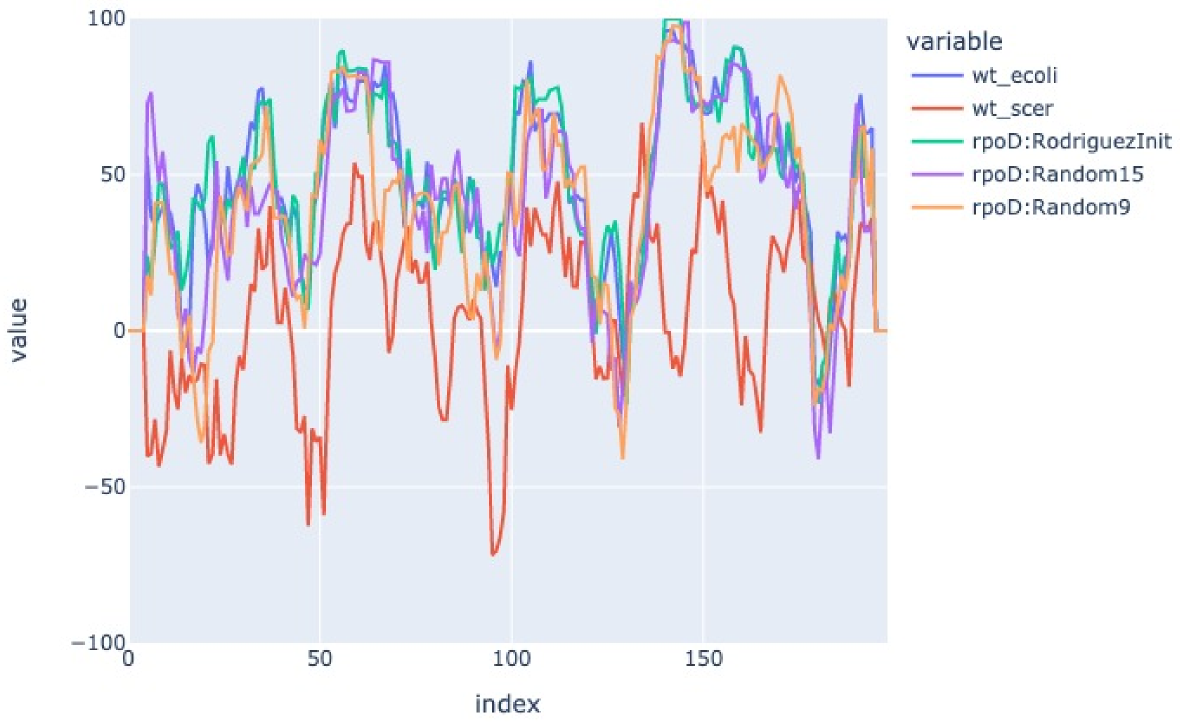


Figure 3.7: Example of the charming algorithm performed on the 200 firsts AA of the rpoD protein originating from *E coli* taking into account the relative frequency of use of each codon of the yeast Saccharomyces cerevisiae.

| Sequence | Host | CAI | Δ%MinMax | correlation MinMax |
|---|---|---|---|---|
| Wild type | *E coli* | 0.782 | 0.0 | 1.0 |
| Wild type | *S cer* | 0.624 | 44.6 | 0.36 |
| Harmonized + Rodriguez | *S cer* | 0.785 | 6.4 | 0.95 |
| Harmonized1 | *S cer* | 0.758 | 12.06 | 0.87 |
| Harmonized2 | *S cer* | 0.746 | 13.49 | 0.83 |

Table 3.5: Comparison of the wild type sequence of the rpoD protein expressed in *E coli* (the origin host) and *S cer* and three harmonized version for *S cer* using the Rodriguez harmonization for the initialization or not.

## 3.4 RNA secondary structure prediction

Now that an RNA sequence has been designed to respect the use of the original host codon and thus conserve fluctuations in translation speed, it is necessary to consider the structure that it can take. Indeed, as previously described, the presence of an RNA folding near the 5' end (translation initiation site) is deleterious to translation initiation. If the SD sequence upstream of the "AUG" start codon is included in such a structure, the pairing of a ribosome is hindered and the further reaction cannot take place. It is therefore important to check how the mRNA created behaves by adding the non-coding parts (before 5' and after 3') in order to investigate how it is structured at the first nucleotides. If such folding were to appear further along the strand, the impact would be less problematic from a translational point of view.

One approach could be to generate the different sequences, express them in cell culture in the laboratory and evaluate the structure of the synthesized RNA. Techniques such as x-ray crystallography, nuclear magnetic resonance spectroscopy, cryo-electron microscopy... are obviously expensive and it would not be possible in practice to test successively the different possibilities of codon modification at the beginning of the sequence in order to evaluate the structure. In the end, only the effects would be measured since the productivity and solubility of the proteins would also be evaluated in vitro. Another approach is therefore necessary to anticipate this problem.

Like proteins, RNA is also an important field of research in bioinformatics since it is now accepted that non-coding RNA has a role in cellular metabolism. Depending on the form that the different RNA families can take, their functionality will be different. Two types of algorithms are co-evolving, each with their advantages and disadvantages. It is necessary to distinguish the methods based knowledge then the one based on self-learning algorithms [51].

In the first case, it is necessary to know the different mechanisms that are responsible for folding in order to define rules that will constrain an algorithm to find an optimal solution. Assuming that a linear RNA strand (without structure) has a high energy level, the goal is to find a conformation with the lowest free energy level, supposedly more stable. This search then becomes an optimization problem that must respect certain rules. The main drawback of this type of method is that their reliability decreases as the considered sequence increases. Their algorithmic complexity is at least of an order $N^3$ (where N represents the length of the chain) which makes them slow.

In the second case, training data is necessary; reference is made here to machine learning algorithms (and their advanced deep learning versions). Here there is no need for a rule, based on a reference data set containing the sequence and the structure evaluated by a laboratory method, the algorithm will learn the rules by itself and build a model. This one requires a large number of parameters (about 70000 for ContextFold for example) which allows to obtain very reliable results in some cases. The advantage of such a method is that once the model is trained, the prediction of a new sequence is very efficient from a computational point of view. However, this approach suffers from a lack of training data. Very few structure data are available and concentrated on some RNA families (ribosomal RNA, transfer RNA). This leads to problems of overfitting and therefore a difficulty to generalize the models to other types of RNA.

In the context of this work, the second approach, although appealing, does not seem to be appropriate since there is not enough data for messenger RNAs. Moreover, even if these data were available, they would only concern the endogenous RNAs of the cells and it is very likely that the model generated from these data would not be generalizable to a modified RNA from another species. It is hoped that the current interest in RNA will generate the large amounts of sequencing data needed in the coming years. The first type of algorithm, although slower and with a lower prediction reliability rate, will be retained and evaluated in the next sections.

It is first necessary to determine the rules that define the structure of RNA. This problem is relatively simpler than proteins for two reasons:

- There are only 4 nucleotides to consider (against 20 amino acids) and their interactions are well known. The number of possible combinations is therefore reduced.

- Unlike proteins, the folding mechanism is hierarchical and sequential. This means that a local fold can be determined within a larger fold and that the strand goes from a primary structure to a secondary structure and then to a tertiary structure with little or no impact on the secondary structure. In the case of proteins, the tertiary structure can have significant impacts on the secondary structure to the point of completely reorganizing it.

On the basis of the information contained in the primary structure it is therefore a priori simpler to infer the secondary structure of an RNA. In the context of this work it is not necessary to try to predict the tertiary structure, the simple fact of knowing if simple folds are present at the beginning of the sequence is sufficient.

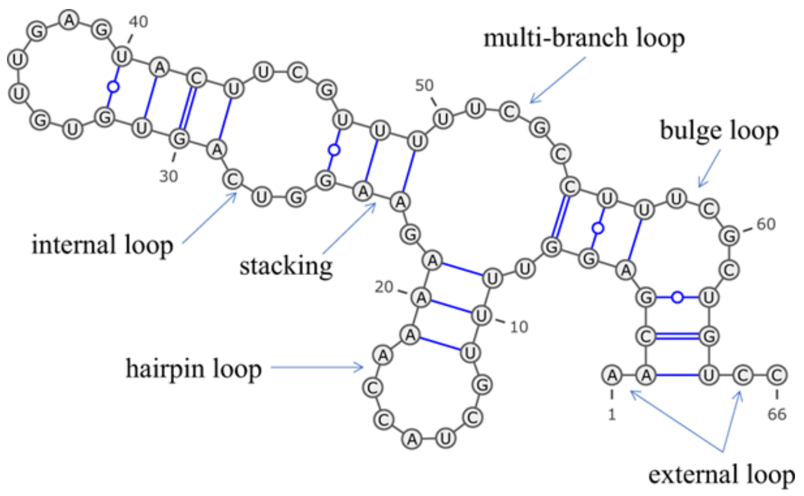


Figure 3.8: Example of an RNA strand with the different secondary structure possibilities. Image from Sato et al. [52]

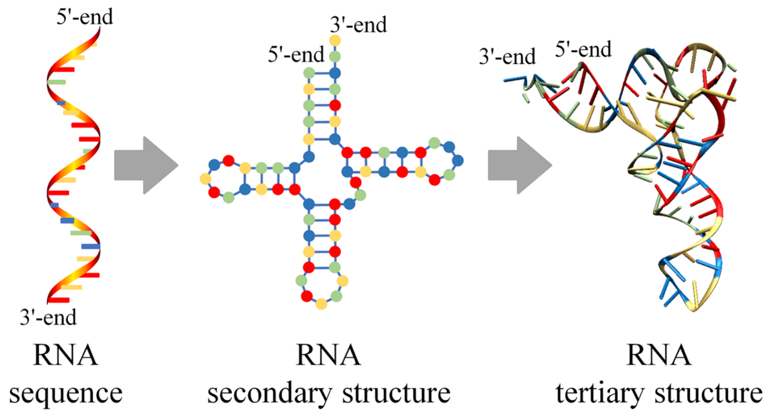


Figure 3.9: The mechanism of RNA folding from primary to tertiary structure. [53]

### 3.4.1 How RNA folds

Since the strategy of the chosen algorithm is dictated by explicit rules, it is necessary to know the determinants of RNA folds. The purpose of this section is not to explore all the mechanisms, but to expose the elements that have allowed to generate the computerized prediction solutions that are used nowadays.

RNA consists of 4 nucleotides divided into two groups: purines (G-C) and pyrimidines (A-U) whose electrostatic interactions are well known and influenced by the ionic strength of the solvent in which they are found. There are few basic secondary structures: helices, loops, bugles and junctions (see fig 3.8). These are the result of interactions between the bases:

- Canonical pairings (A-U), (G-C), the classic Watson-Cricks and (G-U), the wobble effect.
- But also non-canonical pairings, 3-nucleotide pairings and pseudknots.

It is also important to note that the G-C pair is the most stable of all (because it is linked by 3 hydrogen bonds), which is why a simple way to evaluate the structure is to determine the percentage of G-C nucleotides. Indeed, the more these elements are present, the higher the probability to find stable structures.

RNA, based on its primary structure (a linear strand), folds into a secondary and then tertiary structure. The energy required to break a secondary structure is much higher than for the tertiary structure, so it is possible to treat only the first one and consider the next one as a slight destabilization. The representation of the mechanism (see Figure 3.9) is unidirectional and denotes the fact that this it is hierarchical since the primary structure dictates the secondary which in turn dictates the tertiary.

But evaluating the folding in this way is a bit simplistic and thermodynamic and kinetic effects must also be included. Interactions will become more and more favorable as temperature decreases and ionic strength increases. It is furthermore necessary for several interactions to form to create a loop (or other structure), since a single pairing of two bases is a bit weak. Consecutive pairs are needed to stabilize the nascent structure. For this reason, in addition to being hierarchical, the folding mechanism is also considered as sequential [54].

If atomic interactions and spatial clutter are disregarded, the determination of the RNA structure can be seen as a combinatorial problem and be represented by a graph composed of nodes (the nucleotides) and edges (the base pairing) that must respect the following rules:

- Base pair only occur between nucleotides that form canonical pairs or GU.
- A pair cannot be a starting point for two secondary structures.
- Base pair encompass at least three unpaired bases.
- If the elements are placed in the order 5'-3' on an arc of a circle, and the arcs are drawn in a straight line, there can be no crossing over.

### 3.4.2 RNA secondary structure prediction algorithms

In the field of computer science, the first algorithms for predicting RNA secondary structures were developed in the 1970s. These early algorithms focused on simple matches between base pairs and used cost functions that depended on weights defined on the edges. However, these methods were not very reliable and additional factors needed to be considered, such as the requirement of stacking several base pairs to ensure structural stability, the negative impact of unpaired regions, and the use of probabilistic rules like nearest neighbor search. These rules continue to be refined as the field evolves. Although predicting the secondary structure of RNA is a separate area of research, many programs rely on dynamic programming algorithms, which will be discussed in detail below. These scoring methods, called traditional, have been the most widely used for over 40 years. Because the folding process is an optimization problem, it can be solved by breaking it down into simpler sub-problems, the solutions of which contribute to the overall solution. The various programs developed since the 1980s, such as those by Nussinov et al and Zuker et al (to name a few), are all based on dynamic programming algorithms.

To describe this algorithm originally designed by Nussinov [55], *Eddy*[56] consider the task of maximizing base pairings within an RNA sequence. This involves finding the structure with the highest number of pairings, as a higher pairing count implies lower energetic constraints and residual free enthalpy energy. This, in turn, leads to a more stable structure.

The author is discussing a scoring system that assigns a score of +1 for each base pair in a structure and a score of 0 for anything else. He explains that the algorithm is looking at a sub-sequence from position i to position j in a larger sequence of length N and calculating the score for the best possible structure that can be formed within that sub-sequence. The optimal score (S(i,j)) can be defined recursively using the optimal scores of smaller sub-sequences.

There are only 4 ways to consider how the elements i and j can be matched:

1. i,j are a base pair, added on to a structure for i + 1..j - 1.
2. i is unpaired, added on to a structure for i + 1..j.
3. j is unpaired, added on to a structure for i..j - 1.
4. i,j are paired, but not to each other; the structure for i..j adds together sub-structures for two sub-sequences, i..k and k + 1..j (a bifurcation).

In the first scenario, the score assigned to the subsequence S(i,j) is equal to the score of its inner subsequence S(i+1,j-1) plus one, as i and j are paired. In the second and third scenarios, because i and j do not form a pair, the optimal structure for S(i,j) will have a score equal to the score of either S(i+1,j) or S(i,j-1), both incremented by zero. In the last case, the score of the two subsequences is independent under the condition that i and j are not directly matched and the score is simply the sum of the scores of the two subsequences: S(i,k) + S(k+1, j). The optimal score S(i,j) is simply the maximum of these four possibilities which can therefore be expressed recursively as follow:

$$\boxed{S(i,j) = \max \begin{cases} S(i+1,j-1)+1, & \text{if } i,j \text{ base pair} \\ S(i+1,j), & \\ S(i,j+1), & \\ \max_{i<k<j}(S(i,k)+S(k+1,j)) & \end{cases}} \tag{3.14}$$

By using this formulation, a dynamic programming algorithm can be generated by constructing a square matrix with rows and columns representing the letters in the sequence to be analyzed (as shown in the fig 3.10).

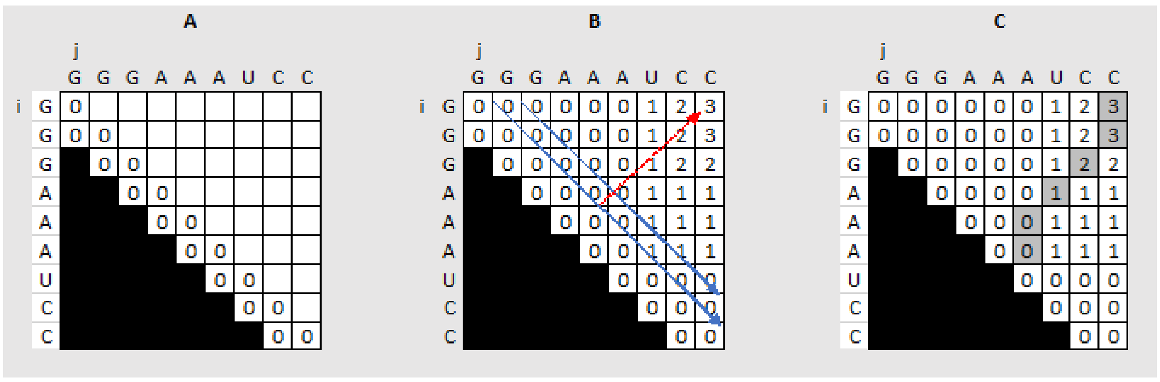


Figure 3.10: A simplified version of a dynamic programming algorithm to assess the RNA secondary structure. The sequence is displayed on a square matrix. **A** The initialization phase: the diagonal of sequence of length 0 or 1 is fixed at 0. **B** The recursive fill: larger sequences are analysed over the time. The calculation are performed until it reaches the upper right corner. **C** The backtracking: from the top right corner, the optimal path is assessed to determine pairing elements.

However, it is important to ensure that previous elements are already computed for the smallest sub-sequences. This can be achieved by initializing the diagonal elements of the matrix for sub-sequences of length 0 and 1 to 0 (i.e., S(i,j) and S(i, j-1) = 0) and performing computations on this diagonal by increasing j and decreasing i, until reaching the upper right corner. This position gives the score of the optimal structure (maximizing the number of pairs). From this point, it is possible to backtrack to determine the optimal path that allowed to reach this score. The memory space is thus reduced to a simple square matrix of the size of the sequence and the algorithm complexity is $N^3$ because of the last term of the recursion which must explore the different possible bifurcations.

The algorithm enables the determination of the maximum number of nucleotide pairings based on specified constraints. Although this approach is relatively straightforward, it serves as the foundation for energy minimization algorithms. To make these algorithms more realistic, additional parameters are continuously incorporated.

Of particular note are the contributions of Zucker et al and Institute for Theoretical Chemistry at Vienna, which builds a package upon Zucker's earlier work in the 1980s. Zucker's work led to the creation of the initial algorithm called mfold [57], which now has a web version available [5]. UNAfold, a modified and enhanced version of mfold, has also been developed [58], but it requires a license for use. The ViennaRNA package [59] consists of libraries written in C and contains many tools including RNAfold which is an RNA fold prediction algorithm. It is also regularly updated (the last stable version is the 2.5.1 and was released on the 2nd of june 2022) and available on the most used operating systems under the condition of citing the authors if it is used in a publication. Once installed, interfaces to other programming languages (Perl, Python 2, Python 3) are available, making it easy to use without having to call a program via the terminal. From a performance point of view, it is currently the package allowing to predict an RNA structure in a traditional way that is the fastest with a performance of 76% in average, which is superior to other algorithms of the same type.

The RNAfold program allows to calculate the free energy of the structure of an RNA sequence provided as input and to give a representation of the different pairs. Without additional options, this representation is a dot, parenthesis notation. Each dot represents an unpaired nucleotide, each pair of parenthesis, two paired nucleotides. It is also possible to obtain a Postscript representation and convert it to an image that can give a more expressive graphical visualization. As an example, the rpoD protein was used with this algorithm to generate the dot-parenthesis notation (see the beinning here below), compute the minimum free energy (617 kcal/mol) and the graphical representation in figure 3.11.

```
..((((......((((.(((((((....((((((((((....))))))))))...))))...((((((......))))))
    .(((.(((.....(((.
```

This point-parenthesis representation seems quite appropriate to determine precisely the nucleotides that generate a secondary structure by pairing between them.

[5]http://www.unafold.org/mfold/applications/rna-folding-form.php

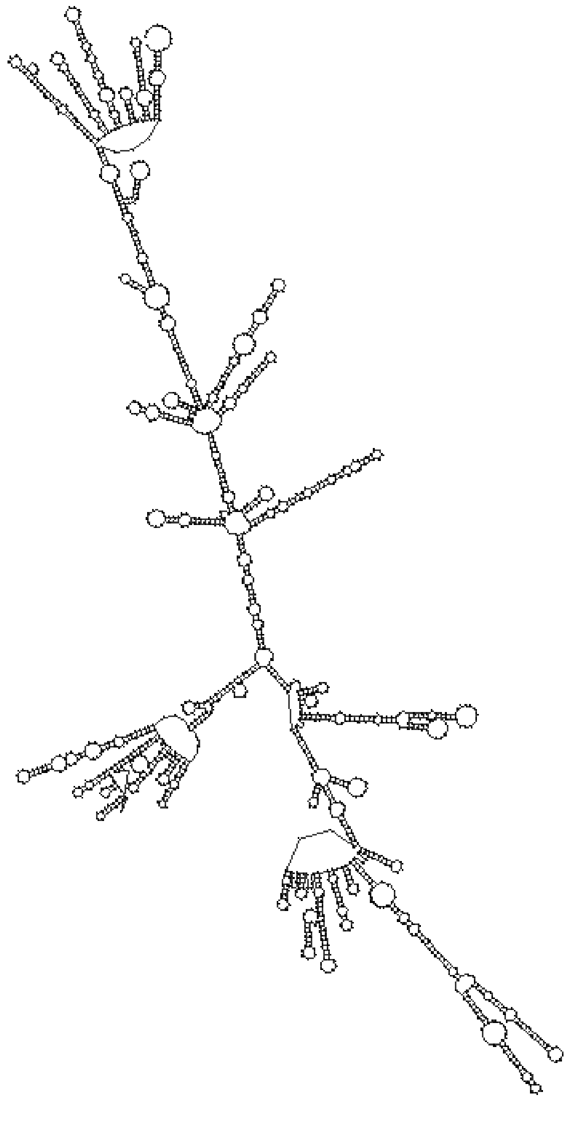


Figure 3.11: A visual representation of the RNA folding of the rpoD protein

## 3.5 Modifying RNA Secondary Structures: Tackling Combinatorial Explosion with Simulated Annealing

Now that we have a method for predicting the secondary structure of RNA, we need to develop a strategy for modifying it in order to eliminate folds that can occur at the beginning of the sequence (within the first 10 codons), which can pose problems for initialization. To achieve this, it is necessary to modify the nucleotides that pair with those further down the strand without altering the sequence of the resulting amino acids. The untranslated region at the beginning of the sequence needs to be considered for this purpose. This region consists of a Shine-Dalgarno sequence (AGGAGG), a spacer comprising 5 to 13 nucleotides, and the start codon 'ATG'. The nucleotides that make up the spacer can be randomly replaced with other nucleotides (A, T, G, C), whereas the first 10 codons of the translated region can only be substituted with a synonymous codon that codes for the same amino acid.

In the worst-case scenario where the 9 codons following the start codon (i.e., 10 - 1 since the start codon is always 'ATG') encode an amino acid with 6 synonymous codons and the spacer contains 13 nucleotides, there would be $4^{13}$ x 1 x $6^{9}$, or roughly 6.8x$10^{14}$, possible alternative sequences. For each of these sequences, the prediction algorithm should be used to determine the number of nucleotides in the beginning of the sequence responsible for the development of a secondary structure (represented by a parenthesis). The sequence with the lowest number of such elements would then be the selected alternative sequence. Nevertheless, testing all of these possibilities would be impractical for most sequences since the time required to compute the structure is dependent on the sequence's length. Therefore, an alternative approach to address this combinatorial explosion must be identified.

For this kind of problem, the general approach is as follow:

- Generate an alternative sequence
- Evaluate it and give it a score using an objective function
- Determine if this score is the best score obtained so far, and if it is the case, keep it as starting point for next alternative sequence generation

These steps must be repeated a fixed number of times or until the score obtained from the objective function has reached a target value. In this case, the scoring method would be to minimize the number of parentheses at the beginning of the sequence.

One possible strategy is to introduce specific mutations aimed at reducing the number of interacting components. The overall concept would be to identify pairs of codons that interact with each other, and then altering them to eliminate any connections pairing between their nucleotides. However, a challenge arises because these modified codons may subsequently interact with other elements located elsewhere on the same strand, sometimes exacerbating the problem. As a result, it is not feasible to employ a stepwise approach, as in CHARMING's genetic algorithm, to iteratively generate progressively better solutions.

A method to reduce the complexity of mRNA structure by preserving synonymous codons that result in the same amino acid sequence considering the whole sequence has already been proposed as a possible optimization method for mRNA [60]. The approach is to consider the entire sequence and use a scoring function based on increasing the minimum free energy, which involves minimizing the number of paired nucleotides, without requiring knowledge of the specific elements involved in a secondary structure. As this combinatorial problem presents a vast research space, the authors employed various methods to solve it, including a minimum free energy scoring function that was less resource-intensive than other predictive algorithms. Although this function was less precise, it was sufficient for their application. However, in our case, identifying problematic elements or penalizing the elements at the beginning of the sequence that are included in a structure is necessary. While the computational time of their scoring function is lower, the search space remains large. To address this issue, they proposed using a simulated annealing algorithm that allows for random mutations to explore the space. This approach enables the algorithm to approach the optimal solution and avoid getting stuck in a local minimum by accepting less optimal solutions based on a certain probability. Additionally, the number of mutations is gradually reduced during iterations.

This second method based on simulated annealing was proposed by *Kirkpatrick et al* in 1983 [61] and is still used nowadays to solve this kind of optimization problem. Simulated annealing is a stochastic optimization technique inspired by the physical process of annealing in metallurgy, which involves slowly cooling a material to increase its strength and reduce defects.

In the context of optimization, simulated annealing involves starting with an initial solution and then iteratively exploring the solution space by randomly perturbing the solution and accepting or rejecting the perturbation based on a probability distribution that allows for some "bad" moves to avoid getting trapped in local optima. The probability distribution is gradually decreased over time to reduce the likelihood of accepting bad moves as the search progresses.

Here are the classical steps of this approach:

1. **Initialization**: Choose an initial solution randomly or by a specific method.
2. **Evaluation**: Evaluate the quality of the initial solution according to the evaluation function.
3. **Main Loop**:
   (a) **Generate neighboring solution**: Generate a neighboring solution by slightly modifying the current solution.
   (b) **Cost calculation**: Evaluate the cost of the new neighboring solution based on the evaluation function.
   (c) **Accept the neighboring solution**: Determine whether the new neighboring solution should be accepted or rejected using an acceptance rule.
   (d) **Update current solution**: If the new solution is accepted, replace the current solution with the new solution.
   (e) **Reduce temperature**: Reduce the temperature using a cooling function.
   (f) **Stop condition**: Stop the algorithm after a fixed number of iterations or when the temperature reaches a predefined threshold.

These steps are repeated until a satisfactory solution is found or the algorithm reaches the stopping condition.

The authors demonstrated the effectiveness of simulated annealing on a variety of optimization problems, including the traveling salesman problem. They compared the results to other optimization techniques, such as gradient descent and random search, and showed that simulated annealing is often able to find better solutions in a reasonable amount of time.

## 3.6 Conclusion

In this chapter, we have further explored the concept of codon usage bias and how it can be computed and manipulated. The proposed examples (graphs and results tables) have been generated by the implementations described in the chapter 4. They respect the specifications of the original algorithms, but some adaptations improving the computation time have been proposed. They have all been written in the same language in order to make them more easily interoperable.

The material needed to develop these algorithms comes from DNA sequencing databases and annotation of coding regions that correspond to proteins. The computation of the codon usage bias frequency table (3.1.1) may be necessary in the case where it must be performed on a portion of the genome only. Otherwise, a regularly updated database (HIVE-CUT) has been proposed. These raw data are interesting but cannot be used as they are, it is necessary to normalize the values obtained and to relate them to their amino acid. Indeed, if an amino acid is under represented, the value of its alternative codons will also be under represented. Four normalization have been proposed: relative use of synonymous codons (3.1.2), relative adaptation of the codon (3.1.3), relative frequency of codon usage (3.1.4), relative codon ranking (3.1.5). The term relative refers to the specific amino acid of each codon. These elements constitute the basis of the measurement and sequence modification algorithms that are developed later; the choice to use one or the other depends on the context.

Then, tools for sequence optimality calculations were presented. At the beginning of the research in this field, the belief was that each host had a preferred codon for each amino acid and that if this codon was used in a higher proportion within a sequence, it would be highly expressed. The result was the CAI (3.2.1), which is a single value that characterizes the sequence by calculating the ratio of the geometric mean of the observed codon usage bias to the geometric mean of the codon usage bias if the preferential codon were used. It is thus an index of optimality with respect to the preferential codon for each amino acid. But during the course of the research, it was realized that the paradigm was to be considered in the other direction. In other words, this means that if a protein is highly expressed, it will have a high CAI, but the opposite is not necessarily true. Other parameters come into play and this single value does not predict whether a protein will be highly expressed or not. Another line of thought has therefore emerged: the fact that we must consider local fluctuations in the codon usage bias throughout the sequence. This is what the CAI profile and %MinMax (3.2.2) algorithms propose. CAI profile is the calculation of the CAI on a rolling window along the sequence. %MinMax has a similar approach, but compares the window analyzed against the average frequency of use of synonymous codons. If the window value is higher (respectively lower), the calculated index is compared to the maximum (respectively minimum) value.

This allows to better detect the presence of rare codon clusters that are suspected to be responsible for the local translation slowdown. This is this last algorithm that will be mostly used in the future. From this one, two other metrics have been defined, the delta MinMax and the correlation MinMax. These allow to compare the %MinMax profiles obtained for a same sequence but expressed in two different hosts. It has been shown that the lower the delta between the two profiles, the more the protein will be expressed in its heterologous host. The correlation allows to compare the fluctuations within the sequence. The closer the profile observed in the heterologous host is to that of the starting host (high correlation), the higher the probability that the protein is expressed in a soluble manner. These two values are the criteria of choice when comparing the sequence optimization algorithms. Indeed, it will be a question of choosing the algorithm which allows both to maximize the MinMax correlation, while minimizing the delta since we seek in priority to mimic the bias of use of the starting host codon in order to produce a soluble protein.

Four optimization algorithms from the literature were then proposed: an optimization by systematically using the preferred codon (3.3.1), a randomization of the codon choice taking into account the frequency of use (3.3.2) and finally two algorithms attempting to harmonize the codon use bias of the destination host sequence with that of the starting host: a method based on the ranking of each codon (3.3.3) and another one which harmonizes more deeply - CHARMING (3.3.3). The latter approach achieved the best results in terms of correlation and delta of the MinMax profiles and is the solution chosen for the development of the complete software pipeline.

But once this algorithm is used, another problem remains to be solved, that of the presence of a possible secondary structure of the RNA at the beginning of the sequence. The main RNA folding mechanisms were presented in order to then introduce the basis of classical prediction algorithms (which use rules and not training data sets) based on dynamic programming (3.4.2). A particular focus was then made on one of them: RNAfold from the ViennaRNA package since it is currently the one that reaches the best prediction performances (on average 76%). If a secondary structure is detected at the beginning of the sequence, it is necessary to try to eliminate or reduce it by modifying the nucleotide sequence while keeping the amino acid sequence. The naive approach of generating all possible sequences is not realistic since the search space is relatively large and the computation time of the secondary structure is high (it can go up to several seconds depending on the sequence size). A less resource intensive strategy has been briefly discussed: simulated annealing (3.5). The prediction function remains the same, but the number of calls to it is reduced since not all possible sequences are tested. It is a random approach that tries to maximize (or minimize) a score while avoiding falling into a local optimum.

These different algorithms should allow, once assembled, to modify a sequence so that it meets the criteria previously exposed (harmonization of the codon use bias with the original host, use of rarer codon at the beginning of the sequence, reduction of the secondary structure at the beginning of the sequence, elimination of alternative start sites). This is what the following chapter proposes by detailing the implementation of these algorithms and then assembling them into a pipeline.

# Chapter 4

# An optimization workflow

After this review of various algorithms, this chapter presents an implementation of a complete workflow for optimizing an mRNA sequence for heterologous host expression. Prior to this, we will discuss a utility class, functions, and slight modifications to the MinMax tool to improve calculation time. The way codon optimization algorithms are used will then be briefly explained. It should be noted that not all proposed algorithms were thoroughly described, but their calculation methods were outlined in the articles, enabling their reconstruction. The results obtained are consistent with those generated by the original authors. Finally, different strategies to reduce the secondary structure at the beginning of the sequence will be detailed (exhaustive search, simulated annealing or a genetic algorithm).

An implementation is proposed in the Python 3 [1] language to make it possible to build a uniform toolbox of compounds compatible with each other. The Python language was chosen since it is the main programming language used in the world of biological sciences which will make the understanding of the code easier for a biologist who would like to use it. The BioPython [2] package could provide support for the implementation of these algorithms, but it will not be used to better master the different concepts.

## 4.1 Utilities

In addition to optimization considerations, utility functions have been developed to represent the proteins and load a complete dataset. The aim is to gather relevant information and functions and provide a user-friendly interface for the various tools to utilize.

### 4.1.1 Constants

To avoid duplicating essential data structures across different modules, a file containing the following three constants can be imported:

- CODON_SIZE
- AA_TO_CODON_DICT
- CODON_TO_AA_DICT

This allows for rapid conversion from an amino acid to its possible codons, or vice versa, from a codon to its corresponding amino acid. For instance, the dictionary AA_TO_CODON_DICT is structured as a key-value pair where the key is an amino acid and the value is a list of its possible codons (for example: S: ['TCA', 'TCC','TCT','TCG','AGC','AGT']).

### 4.1.2 Proteins

A protein is instantiated on the basis of the name of the protein and its RNA or DNA coding sequence. A check of the validity of the character string is performed to verify that they belong to the possibilities of the type of sequence provided (ATGC for DNA, AUGC for RNA) and that the number of elements is multiple of the size of a codon (typically 3). Three parameters are therefore necessary: a name, a nucleotide sequence and the type of nucleotides sequence (DNA or RNA).

[1] `https://www.python.org`
[2] `https://biopython.org`

The purpose of this is to centralize several pieces of information that must be manipulated through the algorithms, such as the:

- RNA sequence
- DNA sequence
- AA sequence
- codon list
- number of nuclotides
- total number of AAs
- total number of occurence of a specific AA through the sequence

### 4.1.3 Proteins import

An easy way to import a protein into a program is to use a file given the length that an RNA sequence can take. A classical file format for the exchange of this type of information is the FASTA format which is nothing else than a text file structured as follows:

```
>ProteinID|Comment
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
```

The first line must contain the ">" sign and the rest can be empty even if it is recommended to add information like name or other comments. The following lines constitute the sequence which are generally less than 120 characters long. This type of file can be simple or multi sequence. The constraint to identify if it is a new protein is the presence of a line with the ">" sign followed by a line break and finally the sequence.

A utility function has been created to read this type of file based on its relative path. It returns a list of objects of type Protein whose name is extracted from the line containing the sign ">" and the sequence which follows. This function only considers the case of proteins encoded on the basis of a DNA sequence.

## 4.2 Codon usage

### 4.2.1 Codon usage table

The CodonUsage class is a representation of a codon usage table in the form of a dictionary where the key is the codon and the value is the frequency in per 1000. Its constructor signature is as follows:

CodonUsage(file, file_format = “sequence”)

This enables the creation of the table based on two types of files: one containing a set of DNA sequences encoding for selected proteins, and the other containing a pre-calculated table from a database. The "file_format" variable is used to specify the type of file provided as input (sequence” or cub” for codon usage bias). If this variable is not provided during the call, the initialization will assume that the file contains a set of sequences. Internal checks are performed on the data provided to generate a valid table.

When the ’cub’ option is used, the file must have 64 lines formatted as "codon" "value". Otherwise, a verification of the file’s validity is performed (i.e., the number of nucleotides is a multiple of 3 and composed only of ATGC elements) before calculating the frequency table for each codon (per thousand). Then, methods corresponding to the algorithms presented in the previous sections (such as RSCU, relative adaptiveness, codon ranking table, etc.) can be called to obtain various table variations (in the form of a dictionary where the keys are the codon and the values are calculated). An additional method that returns a dictionary of the most commonly used codon for each AA has also been implemented to facilitate the implementation of an optimization algorithm.

### 4.2.2 CAI and CAI profile

Two separate modules have been created for the calculation of the CAI and the CAI profile.

The main function to compute the CAI requires an object of type Protein and another of type CodonUsage representing the codon usage table of the expression host in order to return a single CAI value as described previously.

The function creating the CAI profile requires an analysis window size (an integer of the number of codons in the rolling window to be analyzed) and a sliding value that allows to shift the analysis window along the sequence. An object of type Protein of the size of the window to be analyzed is generated at each iteration as long as it can advance in order to compute the local CAI. The successive values of local CAI are added to a list which is returned at the end of the sequence.

### 4.2.3 %MinMax

Based on the study of this algorithm, it is necessary to have the theoretical values $X_{min}$, $X_{max}$, and $X_{avg}$ from the codon usage table for each AA. The authors' algorithm is available through another article [39] on a dedicated GitHub repository [3]. However, when studying the code, it appears that these values are systematically recalculated even though they could be stored in a data structure and called when needed. The code proposed in the appendix takes this optimization into account. A simulation was carried out by calculating the MinMax list for seven proteins 1000 times using both algorithms. The optimized version was about 30% faster, which is a significant difference if the goal is to analyze many proteins of a certain length in a batch or to make such a tool available online to minimize response time. This will also be important in sequence optimization algorithms such as Charming, where this profile must be rechecked many times.

Another variation has been introduced that allows the choice of the codon offset during each iteration. For the CAI profile, the proposed value was 5, while for this algorithm, the default value is 1 codon but can be changed as needed.

The main function of this module, calculate_min_max, requires the following inputs, as in the CAI profile: a protein, a reading window size, a codon usage table, and a shift value. It returns a list of numerical values computed as previously described, which then allows for graphical representation of the variation in codon usage along the sequence.

Finally, two functions have been written to evaluate the similarity of the profiles of two sequences. They take two MinMax profiles as input and return either the delta between them or their Pearson correlation. These two functions are particularly useful for comparing the performance of different sequence optimization algorithms.

## 4.3 Codon bias optimizations

Different sequence optimization algorithms have been generated and are the source of the graphs presented as examples in the "state of the art" section. They are distributed in dedicated files and are called through a dedicated function whose parameters depend on the type of algorithm. For the optimization algorithm for example, only the protein and the codon usage table of the destination host are needed, whereas for the harmonization algorithms, the original expression host must also be taken into account. Each solution returns an object of type Protein, except in the case of the CHARMING program since it generates several modified proteins contained in a list. Other secondary functions are also present to support the main algorithms.

The way in which these different optimizations work have already been widely developed. The CHARMING (which is the one that will be used in the full optimization workflow) algorithm was already available in a Python implementation but some modifications detailed here below were made without affecting the final result. This algorithm is based on the %MinMax tool which is recalculated a lot of times, so the improved version of this tool allows a faster processing time.

The signature of the function is as follows:

```
charming(input_protein, init_cu_table, dest_cu_table, codon_profile_method="MM",
                 number_output=3,sliding_window=10)
```

This allows by default to use the %MinMax algorithm in the scoring function, to generate 3 sequences and to use a reading window of 10 codons. It is possible to change the profiling method by providing the value 'CAI' when calling the function.

The code proposed by the authors was not much factorized which made its reading complicated and did not allow to master well its execution so a refactored and commented version was proposed in addition to bringing the modifications below.

[3]https://github.com/wrightgs/CHARMING

A major change has been made at the initialization were it is necessary to generate multiple random sequences (while respecting the choice of codons for each amino acid) defined by the number of alternative outputs requested by the user. The first sequence is created using the Rodriguez algorithm which respects the ranking of each codon. The other alternative sequences are generated in a totally random way in the version provided by the authors. We propose here to generate these sequences respecting the codon usage frequency of the target organism.

The tests performed show that, for the same sequence (the first 198 nucleotides of the rpoD protein), the number of iterations to be carried out is on average 29 for the version provided by the authors, and 22 in the version generating random sequences while preserving the global codon usage bias of the destination host. This represents a performance gain of about 25%.

## 4.4 Low codon usage ramp

In order to avoid bottlenecks generated by clusters of rare codons spread over the RNA strand, it is recommended to use rare codons at the beginning of the sequence in order to start the translation slowly and thus favor a good distribution of the ribosomes. A dedicated function allows to modify the codons on a desired length by selecting randomly the synonymous codons whose frequency of use does not exceed a certain threshold. Different tests were performed and the threshold of 0.4 gave satisfactory results allowing to reduce the profile %MinMax at the beginning of the sequence. An additional constraint was added: that of not using codons whose frequency of use is too low (or even zero) for the organism under consideration, which could lead to a stop in translation by detachment of the ribosome. The frequency of this low threshold has been arbitrarily set at 0.1. If the current codon already meets the criterion, it is kept, otherwise it is modified. The case where no codon meets the criteria is also considered. Indeed, if there is only one possible codon for an amino acid, its frequency of use will be 1. Similarly, if two codons are used for the same amino acid in equal proportion, their frequency of use will be 0.5. In this case, the algorithm chooses the codon that has the minimum frequency above the threshold of 0.1. The pseudo code of this algorithm is proposed below (see Algorithm 5).

The signature of the function is as follows:

replace_codon_with_rare(sequence, cu_table, threshold=0.4, length_to_change=16)

This allows to modify the threshold or the number of codons to be changed.

**Algorithm 5** The replacement of codons with rare codons at the beginning of the coding sequence

```
for each codon under the number of length_to_change do
    if 0.1 < frequencies[codon] < threshold then
        $new\_codon \leftarrow codon$
    else
        $rare\_codons \leftarrow list(codon\ coding\ for\ same\ aa\ with\ frequency\ between\ 0.1\ and\ 0.4)$
        if rare_codons is not empty then
            $new\_codon \leftarrow random\ choice\ in\ rare\_codons$
        else
            $new\_codon \leftarrow codon\ with\ min\ frequency\ above\ 0.1$
        end if
    end if
    replace current codon with new_codon
end for
return $modified\_sequence$
```

## 4.5 Alternative start removal

In order to prevent a ribosome from attaching and starting a translation at an alternative start site, it is necessary to de-detach and replace them. This could lead to the production of truncated proteins. These are not the desired end product and besides the fact that they would be produced by depleting the cell's energy resources, they could also form inclusion bodies or generate a toxic form for the cell. As a reminder, a translation start site is composed as follows

1. A Shine Dalgarno sequence (the most classical being 'AGGAGG' in E coli)
2. A spacer made of 4 to 14 nucleotides (random choice among ATGC)

3. The "ATG" start condon

It is thus necessary to be able to detect this type of sequences within the RNA and to modify them (except if it is the first of course).

This process is done in two steps. The first one is to find the starting indices of the alternative start sequences. A function takes in the sequence to be analyzed and a list of SD sequences (with a single default value which is the classic "AGGAGG" sequence). A regular expression is then used to detect these elementswith the following pattern in the case of the classical SD sequence:

pattern = "AGGAGG[A,T,G,C]{{4,14}}ATG"

This function then returns the start indices of such sections (except the first one) which will be used for the second step. It is then a question of replacing the codon which includes the different indices. The index of the first element of this codon is obtained by taking the integer division of the index by 3 (the size of a codon) and remultiplied by 3. This allows to deduce the amino acid and thus the different possibilities of codons that can be used to avoid this pattern.

## 4.6 Structure reduction

RNAfold predicts the structure of RNA and gives a dot-parenthesis representation. A parenthesis means that the nucleotide is involved in a pairing with another, which is the source of a secondary structure. We seek to modify the constituent elements of the sequence so that the first 30 nucleotides of the coding part are not included in such a structure. However, it is necessary to use the whole sequence in the evaluation since an early element may pair with a more distant element. The Shine Dalgarno sequence and the spacer linking it to the start codon ('ATG') should also be considered since they can also impact this structure and are important for proper ribosome pairing.

When modifying the elements, it must be taken into account that:

- the nucleotides of the SD sequence cannot be modified,
- the nucleotides of the spacer can be modified in a random way and be replaced by an authorized nucleotide (A,T,G,C),
- the 30 nucleotides of the sequence can be modified only by respecting the alternative codons coding for the same amino acid in order to keep the final protein sequence.

An additional constraint is added: since the goal is to modify the beginning of the sequence and since only the least frequent codons must be used to allow a good distribution of the ribosomes, the alternative codons to consider must respect this constraint. This has the advantageous side effect of reducing the number of possible sequences and therefore the search space.

It will also be assume that only the elements of the beginning of the sequence (contained in the spacer, or in the first 30 nucleotides of the coding sequence) can be modified. In other words, if a nucleotide from the beginning of the sequence pairs with a nucleotide outside these first elements, only this nucleotide is modified to reduce the structure. This constraint is necessary since the more distant elements have already been optimized to respect the codon usage profile of the destination host. In order to generate a complete solution, remodeling these elements would annihilate the efforts previously made to harmonize the sequence. It is possible that a better solution could be obtained by changing the elements further, but this would require a large number of modifications. It should also be noted that a local change can eliminate a structure, but that this can lead to another conformation (better or not), which is why the number of modifications can become enormous.

A struct_reduction module has been created within which a StructReduction class can be instantiated with a destination host codon usage table (necessary to know the possibilities of codons allowed under a certain threshold) and an instance of a reduction strategy class which will be developed in the next sections. The structure scoring function is the same for all strategies and is also present in this file. It calls the RNAfold structure prediction tool (version 2.5.1) and returns the number of opening parentheses found at the beginning of the sequence. Depending on the size of the complete sequence, this function can take several seconds to run. This is the reason why a strategy to minimize the number of calls to this evaluation function must be found.

This organization makes it easy to call the different searching strategies. The *reduce_structure* method of this class requires the coding sequence, the SD sequence, the spacer and uses the codon usage table of the instance to call the *run* method of the chosen strategy.

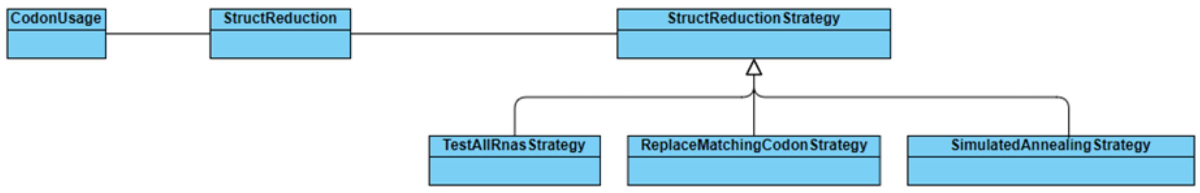


Figure 4.1: The class diagram of the structure reduction architecture

### 4.6.1 Exhaustive search

The exhaustive search method is easy to implement, but it may take a significant amount of time for execution in certain situations. However, it can serve as a benchmark for other strategies in simple scenarios. Its output guarantees to return a sequence (among multiple cases sometimes) with a minimal score, which can be compared with the minimal score obtained through other strategies. This class also output the number of sequences to test as well as the number of possible sequences with the best score. This can give an idea of how difficult it is to find an optimal solution for the many possibilities. The execution flow of its *run* method can be described as followed:

1. generate all possible sequences (see bellow for more information)
2. for each sequence:
   (a) get its structure (in dot parenthesis) and score
   (b) if the score is bellow the best one obtain until now:
      i. update the best score
      ii. update the best sequence and best spacer with these new ones
      iii. reinitialize the variable that count the number of sequence with the best score at 0
   (c) if the current score is equal to the best score
      i. increment by 1 the variable that count the number of best sequences
3. return the best spacer, the best sequence, the best score and the number of iteration before finding the first best sequence

For the random generation of sequences, a list of the length of the spacer is created. Each element being itself a list containing the elements “A,T,G,C”. A list of all possible combinations of these lists is then created. For example, in the case of a spacer of length 2, two list will be created within a list as followed: [[‘A’,‘T’,‘G’,‘C’],[‘A’,‘T’,‘G’,‘C’]]. Then all the combinations between these two list will be created to generate a new list as followed: [‘AA’,‘AT’, ‘AG’, ‘AC’, ‘TA’,...,‘CC’]. For the coding sequence part, the codon usage frequency constraint is respected as explained in the low codon ramp section. The only difference is that instead of choosing a codon randomly, all codons respecting the constraint are kept to generate a list of the different possibilities as for the spacer. The different spacers and coding regions are then crossed to generate all possible sequences. The list of these is randomly shuffled before being returned to the main algorithm. This final rearrangement is not necessary, but it should allow to reach more quickly the sequence with the best score.

### 4.6.2 Selective codon change

This method is more intricate and follows the progressive elimination principle of interacting elements. To eliminate elements, a nucleotide identified at the beginning of the sequence as paired with another must be replaced by a non-pairing element. However, the nucleotide’s position must be considered. If it is in the SD sequence, it cannot be modified; if it is in the spacer, it can easily be replaced with a non-pairing alternative. But if it is in the coding sequence, the codon to which it belongs must be identified and replaced with a synonymous codon that does not match the complementary element, while still meeting the frequency of use threshold constraint. This change can result in a better or worse solution as it induces a change of secondary structure, potentially causing the codon used to match elsewhere in the sequence. By progressively iterating through these changes, the algorithm gradually reduces the secondary structure at the beginning of the sequence. Unlike the previous approach, this one offers the option to modify codons beyond the first ten of the coding sequence. This alternative allows us to determine if an acceptable solution can be found without modifying the result provided by the sequence harmonization algorithm significantly. To instantiate this class, it is necessary to specify whether a modification further down the strand is allowed or not.

Broadly speaking, this strategy involves three steps:

1. Generate a sequence by replacing known pairing elements at the beginning of the sequence.
2. Evaluate the secondary structure and assign it a score..
3. Keep the sequence for the next iteration if it has a better score and update best score.

These are repeated unless the score reaches 0 (an optimal solution) or if there was more then 100 iterations without improvement.

All the complexity actually lies in the mutation method whose steps can be summarized as follows:

1. The algorithm takes as input a SD sequence, a spacer, a sequence to be mutated, a structure in dots-parenthesis, a number of nucleotides to be evaluated (corresponding to the sum of the nucleotides which constitute the sequence of SD, the spacer and the 10 first codons), a table of codon usage and a boolean authorizing or not the replacement of the codons beyond the ten first.
2. It then iterates over each position of the structure, and searches for an opening parenthesis.
3. If an opening parenthesis is found, the algorithm uses a support function to find the position of the corresponding closing parenthesis.
4. Then it checks if the position of the opening and closing parenthesis are in the SD sequence, the spacer or the first codons of the sequence.
   - If the position of the opening parenthesis is in the SD sequence, a replacement can take place if the corresponding codon is in the first elements. Otherwise, it will depend on the value of the variable "replacement_outside_first_allowed" which allows or not the replacement further on the strand.
   - If the position of the opening parentesis is in the spacer, the algorithm replaces the corresponding nucleotide by an alternative nucleotide not forming a pair.
   - If the position is in the coding region, the algorithm replaces the corresponding nucleotides by an alternative codon using the codon usage table "cu_table" by calling a secondary function that also considers the cases where the replacement beyond the first 10 elements is allowed or not. The goal is to find the best alternative that minimizes a matching score by considering the synonymous codons that respect a frequency of use threshold for the elements at the beginning of the sequence. This constraint is not applied further on the strand. The score is obtained by comparing 2 by 2 each letter constituting the two codons. The GC, AT and GT pairs (and their inverses) have a non-zero score and the sum of the scores obtained gives a codon matching score.
5. After each nucleotide (or codon) replacement, the algorithm jumps to the position of the beginning of the next nucleotide (or codon).
6. The algorithm continues to iterate on the positions of the structure until it has evaluated the number of nucleotides specified by "n_nucleotides_to_evaluate".
7. Finally, the algorithm returns the modified sequences for the spacer and the coding region.

Due to the nature of the algorithm it should be noted that it is possible to fall into a local minimum because it is assumed that the sequence with the best current score is necessarily on the path to the optimal solution which may not be the case.

### 4.6.3 Simulated annealing

Finally, the last strategy considered is that of simulated annealing which explores the search space in a random way while accepting less optimal solutions according to a certain probability, which avoids falling into a local minimum. Indeed, as already explained, a targeted mutation will perhaps eliminate a link, but the risk is that it will create others or even lead the algorithm towards modifications that do not lead to a global optimal solution. Accepting worse solutions at the beginning avoids falling into a local minimum and randomly exploring the entire search space.

As for the exhaustive search, only modifications at the beginning of the sequence will be allowed considering synonymous codons that are under a certain threshold.

The general principle of simulated annealing has already been presented, but the way it has been implemented for the problem of reducing the secondary structure of a sequence is explained below. The temperature parameter has two purposes: it is a parameter of the function which allows to accept or not a solution (even if it is worse than the current solution) and it also determines the number of modifications to be done in the mutation function of the sequence.

1. **Initialization**: The SD, the spacer and the sequence given are used as a starting point and considered as the best and current to continue the algorithm.
2. **Evaluation**: Evaluate the quality of the initial solution with the scoring function (the same used in the other strategies that count the number of opening parenthesis).
3. **Main Loop**:
   (a) **Generate neighboring solution**: Generate a neighboring solution by slightly modifying the current solution. This is performed by a mutation function that takes the current spacer, sequence, temperature and the codon usage table. The current temperature is used to determine the number of mutations to perform. During the iterations the temperature parameter of the simulated annealing will decrease which will progressively reduce the number of mutations that will be performed in this function. For the spacer, the nucleotides are chosen in a completely random way whereas for the coding sequence the codons must respect the amino acid for which it codes, but also the threshold of frequency of use (since it must be low at the beginning of the sequence). A random choice is made for each mutation among the different possibilities.

   (b) **Cost calculation**: Evaluate the score of this solution with the structure evaluation function.

   (c) **Accept the neighboring solution**: Determine whether the new neighboring solution should be accepted or rejected using an acceptance rule. If the score of the new solution is better then the current one, it is accepted, otherwise, it is accepted (or not) based on a probability distribution. It is based on the Boltzmann equation, which relates the probability of a system being in a particular state to its energy (the score of the structure) and temperature. The higher the temperature, the more likely the algorithm is to accept a worse solution. As the temperature decreases, the probability of accepting a worse solution also decreases.
   The Boltzmann equation is given by:

   $$P(E) = \frac{1}{Z}e^{-\frac{E}{kT}} \tag{4.1}$$

   where $P(E)$ is the probability of the system being in a state with energy $E$, $k$ is the Boltzmann constant, $T$ is the temperature, and $Z$ is the partition function, which ensures that the probabilities sum up to 1 over all possible energies.
   The Boltzmann equation is used to calculate the probability of accepting a candidate solution that has worst score than the current solution. The acceptance probability is given by:

   $$P_{accept} = e^{-\frac{\Delta E}{kT}} \tag{4.2}$$

   where $\Delta E$ is the difference in score between the candidate solution and the current solution. This equation corresponds to the Boltzmann distribution with $E = \Delta E$, so it expresses the probability of accepting a worse solution as a function of temperature and the difference in energy.
   To decide whether to accept the candidate solution, the algorithm generates a random number $r$ between 0 and 1, and accepts the solution if $r \leq P_{\text{accept}}$. This allows the algorithm to explore the solution space in a probabilistic way, and to escape from local optima by occasionally accepting worse solutions. As the temperature decreases, the acceptance probability decreases as well, leading the algorithm to converge towards the global optimum.

   (d) **Update current solution**: If the new solution is accepted, replace the current solution with the new solution.

   (e) If the new solution is better then the best one seen so far, update the score, the spacer and the sequence with the new values.

   (f) **Reduce temperature**: Reduce the temperature using a cooling function which depends on the number of iterations.

   (g) **Stop condition**: Stop the algorithm after a fixed number of iterations (1000 is proposed as a default value).

The best solution is always saved and updated if needed. This ensures that even if the algorithm continues to iterate in the wrong direction, the best solution is retained. This one is returned at the end of the execution.

| Strategy | Best Sequence until codon 10 | BestScore |
|---|---|---|
| Native | | 23 |
| Exhaustive search | AGGAGGAATGTGTCCACGTGTTGTTTTGTTGTTGTTC<br>.(((........)))...................((( | 6 |
| Selective change (A) | AGGAGGAATGTGTCCTCGTGTCGTCTTATTGTTATTC<br>.(((((((((........................... | 9 |
| Selective change (B) | AGGAGGAATGTGTCCCCGTGTGGTCTTGTTGTTATTC<br>.((.(((.....)))))................(((( | 9 |
| Simulated annealing | AGGAGGAATGTGTCCACGTGTTGTTTTGTTGTTGTTC<br>.(((........)))...................((( | 6 |

Table 4.1: Comparison of the output returned by the different structure reduction strategies.

### 4.6.4 Test of these different strategies

To test these different strategies, a viral protein will be used. This is the complete ORF2 protein which codes for the major part of the capsid of the hepatitis E virus. The Hecolin vaccine was approved in 2012 only in China against this disease using *E coli* as a production system for a part of this protein (the amino acids 368 to 606). During the production phase, the protein forms inclusion bodies, which means that it is not soluble, but it can be easily purified, unfolded and then refolded into its desired form. The goal is not to compare the sequence obtained with the sequence used in the vaccine but to use the full native sequence of the ORF2 protein only to illustrate the performance of the algorithm on a sequence of sufficient size. It consists of 660 amino acids and its complete wild type nucleotide sequence (1983 residues) can be found in the Appendix.

For the following assays, an SD sequence ("AGGAGG") and a single nucleotide 'A' spacer will be added before the coding sequence.

#### Exhaustive search

There are 2916 possible sequences that respect the different constraints, the initial score for the wild type sequence is 23 (this means that there are 23 opening brackets at the beginning of the sequence). The best possible score is 6 and concerns only 9 sequences out of 2916. The best score was found in this simulation after 357 iterations, but this obviously depends on how the different possibilities were mixed at the beginning. The beginning of the sequence and the structure of the first best solution retained is presented in the synthesis table 4.1, the best spacer was the original letter 'A'.

#### Selective codon change

As a reminder, there were two variants to this algorithm with the possibility (option A) or not (option B) to change codons beyond the first 10. For both variants the best score obtained is 9 which was reached after 2 iterations for version A and after 11 for version B. This also means that the next 100 iterations did not improve the situation and that the algorithm stopped. The best spacer differs for both options: 'A' for option A and 'G' for option B.

#### Simulated annealing

This version reached a solution with a score of 6 with the initial spacer as for the exhaustive search. This solution was found after only 113 iterations, but obviously depends on the random choice during the mutations. This shows nevertheless that this strategy allows to reach one of the optimal solutions (among the 9 possible ones) rather quickly and that an optimization of the number of iteration could be considered.

The solution obtained by the simulated annealing is among the optimal solutions. Other tests have been performed on other proteins (data not shown) with identical results, each time the simulated annealing allowed to find a solution among the optimal solutions. This does not mean that this will always be the case, but the results obtained tend to show that this strategy allows to reach a solution in a reasonable time without having to test all the possibilities. It also allows to reach the global minimal structure score contrary to the selective replacement strategy (whatever the alternative of the latter: with or without codon replacement beyond the first 10).

| Sequence | Host | CAI | Δ%MinMax | correlation %MinMax |
|---|---|---|---|---|
| Wild type | *Homo sapiens* | 0.728 | 0.0 | 1.0 |
| Wild type (full) | *E coli* | 0.588 | 29.417 | 0.268 |
| Optimized (full) | *E coli* | 0.567 | 9.301 | 0.869 |
| Wild type (after 48 nucleotides) | *E coli* | NC | 29.275 | 0.234 |
| Optimized (after 48 nucleotides) | *E coli* | NC | 8.019 | 0.935 |

Table 4.2: Comparison of the wild type sequence of the orf2 protein expressed in *Homo sapiens* (the origin host) and *E coli* and the optimized version for *E coli* using the full optimization workflow (NC: Not computed).

## 4.7 A full optimization workflow

In order to combine all these solutions a complete workflow on the same ORF2 protein will be proposed in this section. It will consist in transposing this protein from its original host, *Homo Sapiens*, to an expression host, *E coli*. The different steps applied will be the following:

1. Load the codon usage tables of both hosts into instances of the CodonUsage class.
2. Loading the DNA sequence of the protein of interest in an instance of the Protein class.
3. Replacement of the first 16 codons with rare codons using the codon usage table of the destination host.
4. Harmonization of the rest of the sequence (after the first 16 codons) using the codon usage tables of the natural host and the destination host by the Charming method which uses %MinMax as an estimator of the codon usage bias, and a window size of 10 and the Rodriguez initialization.
5. Recombination of these two pieces of sequence into a single protein.
6. Elimination of alternative start-up sites.
7. Reduction of the structure of the beginning of the sequence to which is attached an SD sequence and a spacer of height elements "A" using the "simulated annealing" strategy and a maximum number of iterations of 1000.

Beyond the algorithms themselves, the order of their execution is also important. All changes on the sequence beyond the first codons take place before optimizing the structure. Indeed, since the composition of codons directly influences the structure, it is the last step to be performed, any subsequent modification at step 7 could recreate nucleotide pairings with the elements of the beginning of the sequence.

Once these steps are completed, the algorithm returns an optimized version of the protein to be expressed in *E coli*. A graphical representation of the difference between the %MinMax profile of the reference and the wild type and optimized version is proposed in figure 4.4. The graph helps to better visualize the difference against the reference. The schematic view of the secondary structure of the mRNA of the native sequence (figure 4.2) and after optimizations steps (figure 4.3) also helps to visualize the differences.

As expected, the codon usage profile of the optimized version mimics the one of the wild type expressed in the native host except for the beginning of the sequence to reduce the codon usage as well as the secondary structure.

Table 4.2 gives a summary of the values for CAI, Δ%MinMax and correlation %MinMax. However, as the sequence was not harmonized along its entire length, the delta and correlation calculations were performed considering the entire sequence or only the harmonized part. This nevertheless reveals that, for the optimized version, these metrics are better than for the wild type version expressed in the destination host and this even if the whole sequence (thus with the non-harmonized part) is included. This reveals that the algorithms is still harmonizing the profiles as expected.

The structure score which was initially 11 (meaning that there were as many elements at the beginning of the sequence that were involved in a secondary structure) could not be reduced to 0, but the optimum found using simulated annealing is 7 after 99 iterations which is a significant reduction of the elements matching another at the beginning of the sequence. The dot-parenthesis representation of the region including the SD sequence, the spacer and the first 10 codons of the coding sequence shown below makes it easier to identify the positions of the problematic elements.

```
....((((..........))))...................(((
```

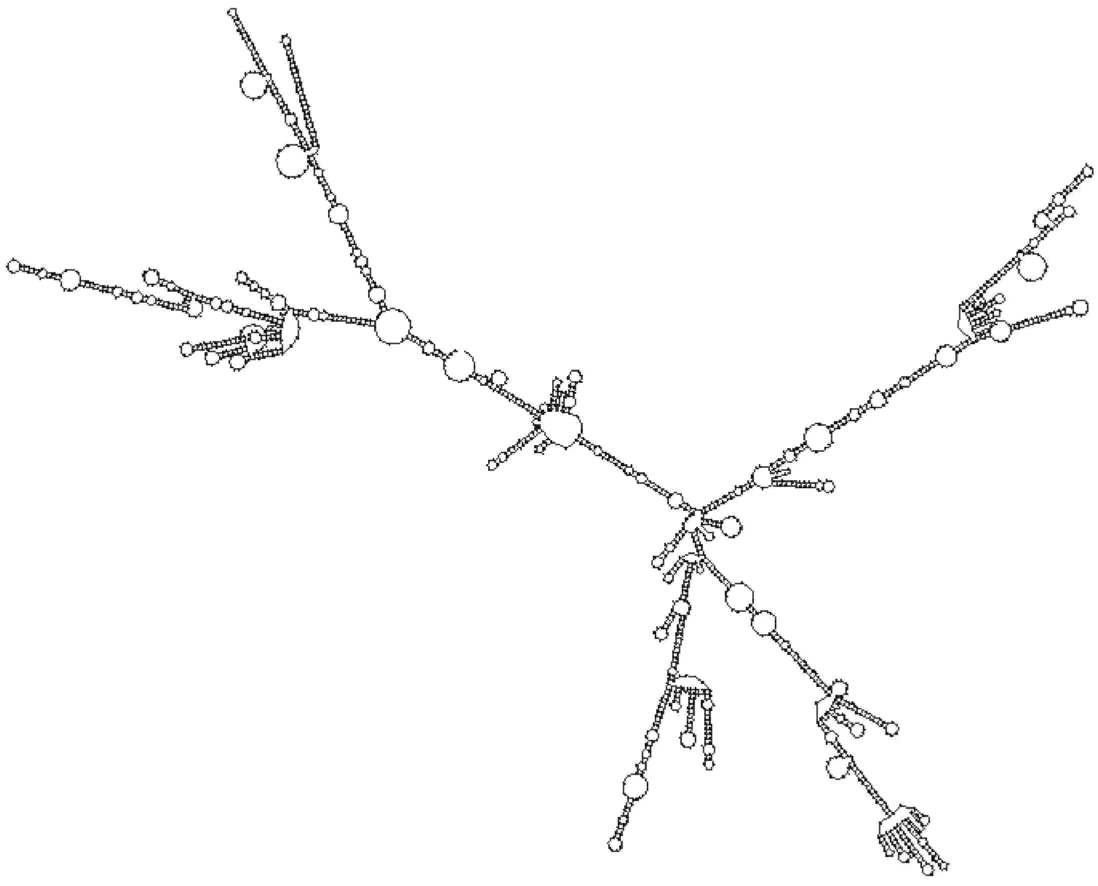


Figure 4.2: A visual representation of the RNA folding of the orf2 protein wild type sequence (the SD, the spacer and the coding sequence)

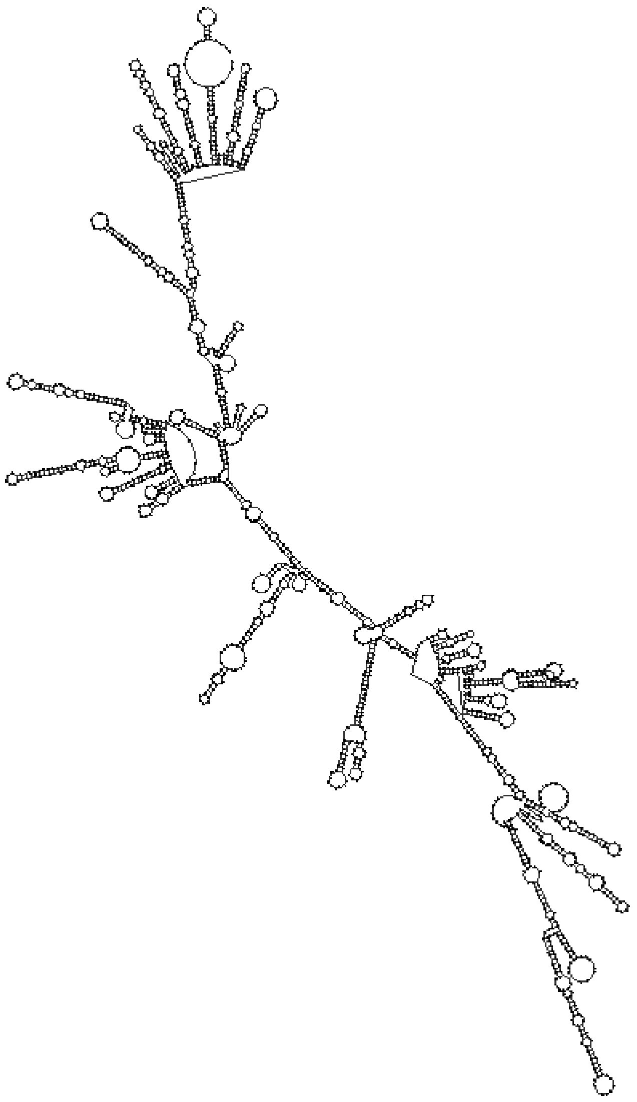


Figure 4.3: A visual representation of the RNA folding of the orf2 protein optimized sequence (the SD, the spacer and the coding sequence)

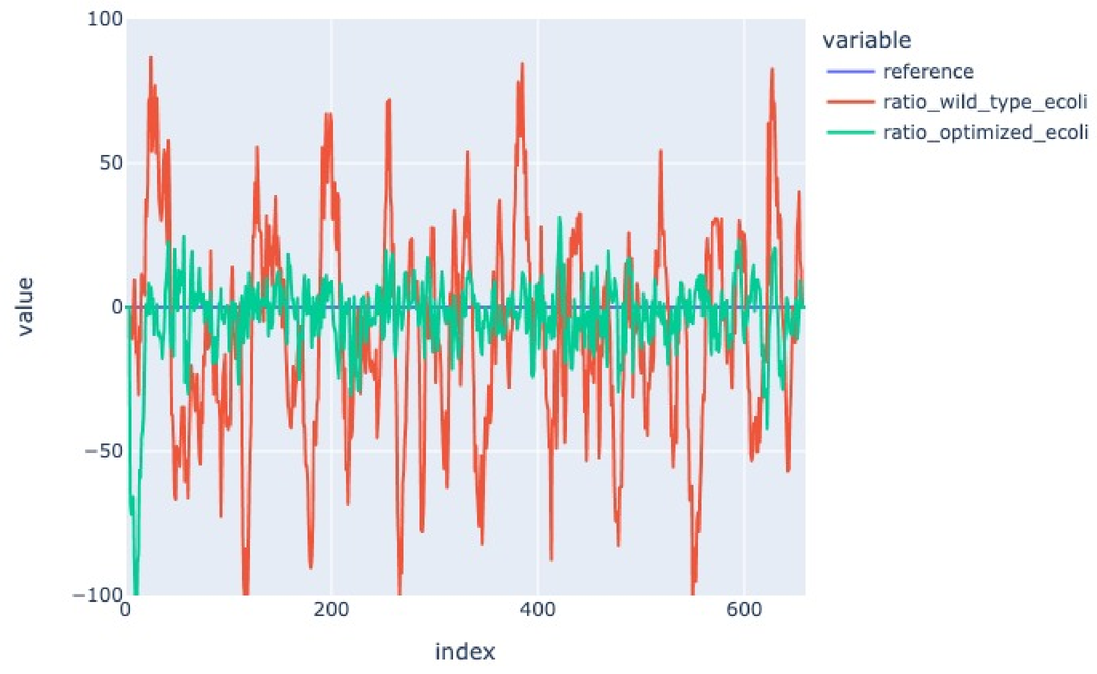


Figure 4.4: A visual representation of the %MinMax profile of the orf2 sequence (SD and spacer excluded). The beginning of the optimized sequence for *E coli* does not follow the wild type sequence profile expressed in human to enable a lower structure and a lower codon usage in this region. The rest of the optimized sequence (after the first 48 nucleotides) mimics the codon usage profile of the native host as opposed to the wild type sequence expressed in the destination host

## 4.8 Source code availability and usage

A GitHub repository that includes all the algorithms presented can be found under the following link: `https://github.com/ekMi/rna_design`. After recovery of the source code, all the functions described above can be called from the command line, it is nevertheless necessary to respect the following prerequisites in order to use them:

- Install the python 3.9 interpreter (later versions are not compatible with the python interface of the viennaRNA package). It is not necessary to install a C compiler since the ViennaRNA package uses wheel to install precompiled files.
- Install the python libraries listed in the requirements.txt file in a new virtual environment with the following command: pip install -r requirements.txt.
- Install Ghostscript which is necessary to transform the postscript file generated by the ViennaRNA package into a jpeg image.

All these requirements can be tedious and depend on the operating system used. An alternative solution is to get the Docker container that contains all the dependencies. To use this program, Docker need to be installed. Docker can be downloaded for free from the official website: https://www.docker.com/get-started Once Docker is installed, the program can be run by opening a terminal window and executing the following command:

```
docker run -it ekmi/rna_optimization_workflow:latest bin/bash
```

This will download the latest version of the image from Docker Hub (if it's not already present on your system) and run it inside a container. Then it will launch a shell interpreter to interact with the different files. Please note that the image is quite large (over 1 GB) due to the dependencies required by the program. Therefore, downloading and building the image may take some time depending on the internet connection speed.

No matter how the program is installed, it contains a folder named tutorial which contains the following files:

- Two codon bias tables (one of the initial host and one of the destination host) in two *.txt files.
- A file test_protein.fasta which contains a short nucleotide sequence.

These files allows to test the complete workflow on the command line using the following command (on one single line):

```
    python3 optimization_workflow.py <path_to_cu_table_origin>
        <path_to_cu_table_dest> <path_to_protein_fasta_file> <SD_sequence>
            <spacer_sequence> <output_name>
```

Where:

- path_to_cu_table_origin is the path to the codon usage table for the origin host.
- path_to_cu_table_dest is the path to the codon usage table for the destination host
- path_to_protein_fasta_file is the path to the FASTA file containing the protein sequence(s)
- SD_sequence is the Shine-Dalgarno sequence to use for translation initiation
- spacer_sequence is the spacer sequence to use between the Shine-Dalgarno sequence and the start codon
- output_name is the name to use for the output files

To launch the execution of the program on the files of the tutorial folder, the following command can for example be used with a SD sequence "AGGAGG" and a spacer of 4 nucelotides. The name of the destination files will start with "test".

```
    python3 optimization_workflow.py tutorial_files/CUB_origin.txt
        tutorial_files/CUB_destination.txt tutorial_files/test_protein.fasta
            "AGGAGG" "AAAA" "test"
```

The previously exposed steps are executed sequentially without any necessary input from the user and no intermediate output is generated.
The program generates the following final outputs:
Files:

- 'test.ps': a PostScript file of the RNA secondary structure
- 'test.png': a PNG image of the RNA secondary structure
- a chart showing the %MinMax profile for the wild-type sequence (in the origin host), the wild-type sequence (in the destination host), and the optimized sequence (in the destination host)

In the standard output of the terminal:

- The best sequence
- The best spacer
- The structure in dot parenthesis notation of the best sequence
- Then for the wild-type sequence in the original and destination host, and the optimized sequence:
  - the CAI
  - the delta MinMax on the full sequence
  - the delta MinMax on the harmonized part only
  - the correlation MinMax on the full sequence
  - the correlation MinMax on the harmonized part only

## 4.9 Conclusion

The chapter presents the implementation of various algorithms in the Python programming language to manipulate sequencing data and create a codon usage table, which is necessary for subsequent algorithms. The codon usage bias CAI and %MinMax were briefly discussed, and optimizations were made to %MinMax to reduce its computation time. The chapter then describes algorithms to modify the provided sequence to achieve certain characteristics. The first is Charming, which harmonizes the codon usage bias of a destination host with that of the starting host. The code was modified to improve execution speed and readability. The next algorithm allows rarer codons to be used at the beginning of the sequence, followed by the detection and modification of possible alternative starting sites. These two methods were created entirely. Finally, the most important test and implementation work was done for the development of different strategies to reduce the secondary structure at the beginning of the sequence. This involved either testing all possible sequences, making targeted modifications, or using the simulated annealing technique. The latter allows to reduce the secondary structure at the beginning of the sequence and to obtain a score as good as with the exhaustive search for the different cases tested. Its advantage is that it calls the scoring function (which is very penalizing from a computation time point of view) a lower number of times than with an exhaustive search. The advantage of the realized pipeline is that the number of alternative sequences is already reduced since only codons having a frequency of use lower than 0.4 can be selected because of the constraint concerning the beginning of the sequence. This is what "save" the simulated annealing algorithm, since if the search space is too large, it may have difficulties to cover it in all its extent. It is also clear that the parameter of the number of iterations is important because if it is too small compared to the search space, it will be more difficult to find an optimal solution. Another criticism is related to the randomness of the sequences that are generated, it is indeed possible that the algorithm tests the same sequence several times, an alternative could be to use a tabu algorithm that saves the tested sequences progressively to avoid retesting them.

This algorithm based on simulated annealing has nevertheless been kept for the implementation of the complete workflow. This one sequentially performed the following steps using the default values for the different parameters (threshold, number of iterations, ...): use of less used codons at the beginning of the sequence and harmonization of the rest with the initial host, elimination of the alternative start sites, reduction of the secondary structure at the beginning of the sequence with the help of the simulated annealing which uses the RNAFold program for its scoring function. This workflow uses mainly Charming which does a remarkable job to harmonize the codon usage bias which seems to be one of the main determinants of a heterologous protein synthesis in soluble form. But we wanted to go a little further by taking into account other optimization criteria described in the literature. Indeed, beyond solubility, the translation initiation phase is crucial and it is necessary to make the beginning of the sequence easily accessible to ribosomes. Then, the fact of starting slowly the reading allows a good pairing of the ribosome and avoids the jamming further on the strand which can lead to the detachment of the ribosomes or to the degradation of the RNA itself. It was not possible to reduce the secondary structure to 0 within the various constraints. Perhaps by relaxing some of them this objective could be achieved. Nevertheless, the reduction proposed by this algorithm remains an advance compared to other sequence optimization algorithms which do not take it into account.

Finally, installation instructions for this package were presented according to two alternatives: the recovery of the source code on a GitHub repository or the installation via a docker container.

# Chapter 5

# Conclusions

This work presents a pipeline of different algorithms to optimize a messenger RNA sequence for translation into a soluble protein in a heterologous host. Different constraints have been identified in the literature such as the need to reproduce the codon usage bias not globally but locally. This has the effect of generating regions or clusters where less common codons are used in order to favor a pause in translation that allows co-translational folds at the protein level. A recently published algorithm was used to achieve this goal: Charming. The results of the harmonizations it proposes allow to reach a high level of correlation between the optimized sequence and the native sequence expressed in the original host. This high correlation is a sign that local fluctuations in codon usage bias are respected, which should allow the protein to be expressed in its soluble form. But this tool omits other factors that can also prevent the production of a protein with high yield. These are the presence of alternative translation start sites, the use of rare codons at the beginning of the sequence to start translation slowly and avoid ribosomal jams, and the reduction of the secondary structure of the RNA strand at the beginning of the sequence in order to favor a good ribosome pairing. For each of these elements, an individual alogithmic solution is proposed while respecting the constraints of the others. Alternative start sites are detected by a regular and expression and alternative rare codons respecting the amino acid they code for have been selected at the beginning of the sequence. Finally, the most complex solution to implement was the creation of an algorithm to reduce the secondary structure at the beginning of the sequence. The secondary structure is predicted using a software package (ViennaRNA) and modified using a simulated annealing algorithm. This approach allows to explore the wide search space in a random way while accepting less good solutions at the beginning in order to avoid falling in a local minimum. Tests performed on different sequences have shown that this approach allows to reach one of the optimal solutions among those discovered by an exhaustive search. This approach was used to reduce the number of calls to the RNAFold structure prediction algorithm due to its high processing time.

As the example shows, it is not always possible to meet all the constraints and in particular the secondary structure. It is necessary to test other parameters, but in some cases, due to the nature of the amino acids to be coded, it is not possible to obtain a reduced structure. A constraint that could be relaxed would be the threshold to be used to select low expressed codons at the beginning of the sequence. Indeed, if the threshold is higher, the number of possible codons for the same amino acid is artificially increased. By offering more possibilities, the simulated annealing algorithm has more probability to find a better solution.

Other optimizations could be considered. The maximum number of iterations for the simulated annealing for example. This is fixed at 1000 by default (but can be changed by the user), but it could be appropriate to take into account the number of possibilities by using an exhaustive search to determine this maximum number of iterations rather then choosing an arbitrary number. The scoring function is also quite simple and does not take into account the location of the elements that are involved in a secondary structure at the beginning of the sequence. It could be useful to modify it in order to penalize even more the elements whose position is lower since it correspond to the ribosome binding site.

But all these elements are dependent on the previous steps where some optimizations are performed randomly which can lead to a different initialization of the simulated annealing. It should also be remembered that these are simulations and that the most successful prediction algorithm for secondary structure to date only achieves a performance score of 76%.

Finally, from a processing speed and user experience point of view, the following evolutions could be explored.

To speed up the processing (and thus potentially increase the number of iterations while keeping the same execution time or simply to reduce the execution time) it would also be possible to parallelize the simulated annealing. This would have the advantage of using several threads to compute the scoring function with the ViennaRNA package. However, it would be necessary to take care to add locks on some data to avoid concurrent access to the variable containing the best score for example. Another approach to parallelization would be to partion the search space and perform the processing in these subspaces and finally compare the optimum found in them. As previously explained, a tabu search could also be considered.

It should also be admitted that the use of this pipeline is not easily parameterized nor accessible by a non computer scientist. The classical solution to make such an algorithm available would be to create a graphical interface in the form of a web application. The difficulty here lies in the fact that the processing is long (several hours) and that in the case of such a solution, it is necessary to be able to provide an answer to the user in a reasonable time. This could be achieved by setting up an asynchronous processing where the user would submit a request via a form, this one would be taken care of by the server which would call an asynchronous processing service and would send back an answer to the user by providing him with a url where he could consult the result once the processing is finished.

In the field of biological sciences, several alternatives are generally possible since nature tends to accept certain errors and compensate for them. This field of research in computer science is very useful because it allows to orientate the alternatives in functions of rational choice on search spaces which would be impossible to evaluate“manuall”. But it is still necessary to test these solutions in a wet laboratory. There is still a long way to go before we master all the elements that will allow us to transpose a sequence from one animal species to another, but this pipeline tries to optimize it in order to respect the constraints mentioned above. Nevertheless, it is important to remain humble and to accept that during evolution, the animal kingdom has taken divergent paths and that it is probably not possible to express any protein in any organism.

# Appendix

# Appendix A

# Hepatitis E virus clone p6, complete genome

```
>JQ679013.1:5359-7341 Hepatitis E virus clone p6, complete genome
ATGTGCCCTAGGGTTGTTCTGCTGCTGTTCTTCGTGTTTCTGCCTATGCTGCCCGCGCCACCGGCCGGCC
AGCCGTCTGGCCGTCGTCGTGGGCGGCGCAGCGGCGGTGCCGGCGGTGGTTTCTGGGGTGACAGGGTTGA
TTCTCAGCCCTTCGCCCTCCCCTATATTCATCCAACCAACCCCTTCGCCGCCGATATCGTTTCACAATCC
GGGGCTGGAACTCGCCCTCGGCAGCCGCCCCGCCCCCTTGGCTCCGCTTGGCGTGACCAGTCCCAGCGCC
CCTCCGCTGCCCCCCGCCGTCGATCTGCCCCAGCTGGGGCTGCGCCGTTGACTGCTGTATCACCAGCCCC
TGACACAGCCCCTGTACCTGATGTTGATTCACGTGGTGCTATTCTGCGTCGGCAGTATAATTTGTCCACG
TCCCCGCTCACGTCATCTGTTGCTTCGGGTACCAATTTGGTTCTCTACGCTGCCCCGCTAAATCCCCTCT
TGCCCCTCCAGGATGGCACCAACACCCATATCATGGCTACTGAGGCATCCAACTATGCTCAGTACCGGGT
CGTTCGAGCTACGATCCGCTACCGCCCGCTGGTGCCGAATGCTGTTGGTGGTTATGCTATTTCTATTTCT
TTTTGGCCTCAAACTACAACTACCCCTACTTCTGTTGATATGAATTCTATTACTTCCACTGATGTTAGGA
TTTTGGTCCAGCCCGGTATTGCCTCCGAGTTAGTCATCCCTAGTGAGCGCCTTCATTATCGCAATCAAGG
CTGGCGCTCTGTTGAGACCACAGGTGTGGCTGAGGAGGAGGCTACCTCCGGTCTGGTAATGCTTTGCATT
CATGGCTCTCCTGTTAACTCTTATACTAATACACCTTACACTGGTGCGTTGGGGCTCCTTGATTTTGCAC
TAGAGCTTGAATTCAGGAATTTGACACCCGGGAACACCAACACCCGTGTTTCCCGGTATACCAGCACAGC
CCGTCATCGGTTGCGTCGCGGTGCTGATGGGACCGCTGAGCTTACTACCACAGCAGCCACACGATTTATG
AAGGATCTGCATTTCACTGGCACTAATGGCGTTGGTGAGGTGGGTCGCGGTATCGCCCTGACACTGTTCA
ATCTTGCTGATACGCTTCTAGGTGGTTTACCGACAGAATTGATTTCGTCGGCTGGGGGTCAGTTGTTCTA
CTCCCGCCCTGTTGTCTCGGCCAATGGCGAGCCGACAGTGAAGTTATACACATCTGTGGAGAATGCGCAG
CAAGACAAGGGCATTACCATCCCACACGATATAGATTTGGGTGACTCCCGTGTGGTTATTCAGGATTATG
ATAATCAGCACGAGCAAGACCGACCCACGCCGTCACCTGCCCCCTCACGCCCTTTCTCAGTCCTTCGCGC
TAACGATGTTTTGTGGCTCTCCCTCACTGCCGCTGAGTACGATCAGGCTACGTATGGGTCGTCTACCAAC
CCTATGTATGTCTCTGATACAGTTACCTTTGTCAATGTGGCCACTGGTGCTCAGGCTGTTGCCCGCTCTC
TTGATTGGTCTAAAGTTACTTTGGATGGTCGCCCCCTTACTACCATTCAGCAGTATTCTAAGACATTTTA
TGTTCTCCCGCTCCGCGGGAAGCTGTCCTTTTGGGAGGCTGGCACAACTAGGGCCGGCTACCCATATAAC
TATAACACCACTGCTAGTGATCAAATTCTGATTGAGAATGCGGCCGGCCATCGTGTCGCTATCTCCACCT
ACACTACCAGCCTGGGTGCCGGCCCTGCCTCGATCTCCGCGGTGGGTGTATTAGCCCCACACTCGGCCCT
TGCTGTTCTTGAGGACACTGTTGATTACCCTGCTCGTGCTCACACTTTTGATGATTTCTGCCCGGAGTGT
CGTACCCTAGGTTTGCAGGGTTGTGCATTCCAGTCCACTATTGCTGAGCTTCAGCGCCTTAAAACGGAGG
TAGGCAAAACCCGGGAGTCTTAA
```

# Bibliography

[1] B. Greenwood. “The contribution of vaccination to global health: past, present and future”. In: *Philos Trans R Soc Lond B Biol Sci* 369.1645 (2014), p. 20130433.

[2] E. De Gregorio and R. Rappuoli. “From empiricism to rational design: a personal perspective of the evolution of vaccine development”. In: *Nat Rev Immunol* 14.7 (July 2014), pp. 505–514.

[3] S. Plotkin. “History of vaccination”. In: *Proc Natl Acad Sci U S A* 111.34 (Aug. 2014), pp. 12283–12287.

[4] R. Cid and J. Bolívar. “Platforms for Production of Protein-Based Vaccines: From Classical to Next-Generation Strategies”. In: *Biomolecules* 11.8 (July 2021).

[5] M. J. Francis. “Recent Advances in Vaccine Technologies”. In: *Vet Clin North Am Small Anim Pract* 48.2 (Mar. 2018), pp. 231–241.

[6] C. M. O’Connor and J. U. Adams. “Essentials of Cell Biology”. In: *Cambridge, MA: NPG Education* (2010). URL: `https://www.nature.com/scitable/ebooks/essentials-of-cell-biology-14749010`.

[7] J. D. Watson and F. H. C. Crick. “Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid”. In: *Nature* 171.4356 (Apr. 1953), pp. 737–738. ISSN: 1476-4687. DOI: `10.1038/171737a0`. URL: `https://doi.org/10.1038/171737a0`.

[8] Pray L. “Discovery of DNA structure and function: Watson and Crick.” In: *Nature Education* (2008). URL: `https://www.nature.com/scitable/topicpage/discovery-of-dna-structure-and-function-watson-397`.

[9] P. Richard and J. L. Manley. “Transcription termination by nuclear RNA polymerases”. In: *Genes Dev* 23.11 (June 2009), pp. 1247–1269.

[10] J. Shine and L. Dalgarno. “The 3’-terminal sequence of Escherichia coli 16S ribosomal RNA: complementarity to nonsense triplets and ribosome binding sites”. In: *Proc Natl Acad Sci U S A* 71.4 (Apr. 1974), pp. 1342–1346.

[11] J. Ma, A. Campbell, and S. Karlin. “Correlations between Shine-Dalgarno sequences and gene features such as predicted expression levels and operon structures”. In: *J Bacteriol* 184.20 (Oct. 2002), pp. 5733–5745.

[12] I. Gusarov and E. Nudler. “The mechanism of intrinsic transcription termination”. In: *Mol Cell* 3.4 (Apr. 1999), pp. 495–504.

[13] A. Hecht, J. Glasgow, P. R. Jaschke, L. A. Bawazer, M. S. Munson, J. R. Cochran, D. Endy, and M. Salit. “Measurements of translation initiation from all 64 codons in E. coli”. In: *Nucleic Acids Res* 45.7 (Apr. 2017), pp. 3615–3626.

[14] C. B. Anfinsen. “Principles that govern the folding of protein chains”. In: *Science* 181.4096 (July 1973), pp. 223–230.

[15] P. D. Sun, C. E. Foster, and J. C. Boyington. “Overview of protein structural and functional folds”. In: *Curr Protoc Protein Sci* Chapter 17 (May 2004), Unit 17.1.

[16] Alberts B, Johnson A, Lewis J, and et al. *Molecular Biology of the Cell. 4th edition.* New York: Garland Science, 2002.

[17] Levinthal, Cyrus. “Are there pathways for protein folding?” In: *J. Chim. Phys.* 65 (1968), pp. 44–45. DOI: `10.1051/jcp/1968650044`. URL: `https://doi.org/10.1051/jcp/1968650044`.

[18] J. Pereira, A. J. Simpkin, M. D. Hartmann, D. J. Rigden, R. M. Keegan, and A. N. Lupas. “High-accuracy protein structure prediction in CASP14”. In: *Proteins* 89.12 (Dec. 2021), pp. 1687–1699.

[19] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021), pp. 583–589.

[20] L. Duan, X. Guo, Y. Cong, G. Feng, Y. Li, and J. Z. H. Zhang. "Accelerated Molecular Dynamics Simulation for Helical Proteins Folding in Explicit Water". In: *Front Chem* 7 (2019), p. 540.

[21] N. Gregersen, P. Bross, S. Vang, and J. H. Christensen. "Protein misfolding and human disease". In: *Annu Rev Genomics Hum Genet* 7 (2006), pp. 103–124.

[22] J. G. Thomas and F. Baneyx. "Protein misfolding and inclusion body formation in recombinant Escherichia coli cells overexpressing Heat-shock proteins". In: *J Biol Chem* 271.19 (May 1996), pp. 11141–11147.

[23] F. Baneyx. "Recombinant protein expression in Escherichia coli". In: *Curr Opin Biotechnol* 10.5 (Oct. 1999), pp. 411–421.

[24] G. L. Rosano and E. A. Ceccarelli. "Recombinant protein expression in Escherichia coli: advances and challenges". In: *Front Microbiol* 5 (2014), p. 172.

[25] F. H. Crick. "Codon–anticodon pairing: the wobble hypothesis". In: *J Mol Biol* 19.2 (Aug. 1966), pp. 548–555.

[26] C. Gustafsson, S. Govindarajan, and J. Minshull. "Codon bias and heterologous protein expression". In: *Trends Biotechnol* 22.7 (July 2004), pp. 346–353.

[27] J. B. Plotkin and G. Kudla. "Synonymous but not the same: the causes and consequences of codon bias". In: *Nat Rev Genet* 12.1 (Jan. 2011), pp. 32–42.

[28] D. M. Mauger, B. J. Cabral, V. Presnyak, S. V. Su, D. W. Reid, B. Goodman, K. Link, N. Khatwani, J. Reynders, M. J. Moore, and I. J. McFadyen. "mRNA structure regulates protein expression through changes in functional half-life". In: *Proc Natl Acad Sci U S A* 116.48 (Nov. 2019), pp. 24075–24083.

[29] T. Tuller, A. Carmi, K. Vestsigian, S. Navon, Y. Dorfan, J. Zaborske, T. Pan, O. Dahan, I. Furman, and Y. Pilpel. "An evolutionarily conserved mechanism for controlling the efficiency of protein translation". In: *Cell* 141.2 (Apr. 2010), pp. 344–354.

[30] K. Itakura, T. Hirose, R. Crea, A. D. Riggs, H. L. Heyneker, F. Bolivar, and H. W. Boyer. "Expression in Escherichia coli of a chemically synthesized gene for the hormone somatostatin". In: *Science* 198.4321 (Dec. 1977), pp. 1056–1063.

[31] J. Konczal, J. Bower, and C. H. Gray. "Re-introducing non-optimal synonymous codons into codon-optimized constructs enhances soluble recovery of recombinant proteins from Escherichia coli". In: *PLoS One* 14.4 (2019), e0215892.

[32] N. Behloul, W. Wei, S. Baha, Z. Liu, J. Wen, and J. Meng. "Effects of mRNA secondary structure on the expression of HEV ORF2 proteins in Escherichia coli". In: *Microb Cell Fact* 16.1 (Nov. 2017), p. 200.

[33] Y. Y. Waldman, T. Tuller, T. Shlomi, R. Sharan, and E. Ruppin. "Translation efficiency in humans: tissue specificity, global optimization and differences between developmental stages". In: *Nucleic Acids Res* 38.9 (May 2010), pp. 2964–2974.

[34] S. Bahiri-Elitzur and T. Tuller. "Codon-based indices for modeling gene expression and transcript evolution". In: *Comput Struct Biotechnol J* 19 (2021), pp. 2646–2663.

[35] M. Gouy and C. Gautier. "Codon usage in bacteria: correlation with gene expressivity". In: *Nucleic Acids Res* 10.22 (Nov. 1982), pp. 7055–7074.

[36] M. J. Ranaghan, J. J. Li, D. M. Laprise, and C. W. Garvie. "Assessing optimal: inequalities in codon optimization algorithms". In: *BMC Biol* 19.1 (Feb. 2021), p. 36.

[37] P. M. Sharp and W. H. Li. "The codon Adaptation Index–a measure of directional synonymous codon usage bias, and its potential applications". In: *Nucleic Acids Res* 15.3 (Feb. 1987), pp. 1281–1295.

[38] C. Mignon, N. Mariano, G. Stadthagen, A. Lugari, P. Lagoutte, S. Donnat, S. Chenavas, C. Perot, R. Sodoyer, and B. Werle. "Codon harmonization - going beyond the speed limit for protein expression". In: *FEBS Lett* 592.9 (May 2018), pp. 1554–1564.

[39] G. Wright, A. Rodriguez, J. Li, T. Milenkovic, S. J. Emrich, and P. L. Clark. "CHARMING: Harmonizing synonymous codon usage to replicate a desired codon usage pattern". In: *Protein Sci* 31.1 (Jan. 2022), pp. 221–231.

[40] T. F. Clarke and P. L. Clark. “Rare codons cluster”. In: *PLoS One* 3.10 (2008), e3412.

[41] A. Rodriguez, G. Wright, S. Emrich, and P. L. Clark. “%MinMax: A versatile tool for calculating and comparing synonymous codon usage and its impact on protein folding”. In: *Protein Sci* 27.1 (Jan. 2018), pp. 356–362.

[42] T. F. Clarke and P. L. Clark. “Increased incidence of rare codon clusters at 5’ and 3’ gene termini: implications for function”. In: *BMC Genomics* 11 (Feb. 2010), p. 118.

[43] L. Pellizza, C. Smal, G. Rodrigo, and M. Arán. “Codon usage clusters correlation: towards protein solubility prediction in heterologous expression systems in E. coli”. In: *Sci Rep* 8.1 (Feb. 2018), p. 10618.

[44] G. Wright, A. Rodriguez, J. Li, P. L. Clark, T. ć, and S. J. Emrich. “Analysis of computational codon usage models and their association with translationally slow codons”. In: *PLoS One* 15.4 (2020), e0232003.

[45] H. G. Menzella. “Comparison of two codon optimization strategies to enhance recombinant protein production in Escherichia coli”. In: *Microb Cell Fact* 10 (Mar. 2011), p. 15.

[46] S. Jayaraj, R. Reid, and D. V. Santi. “GeMS: an advanced software package for designing synthetic genes”. In: *Nucleic Acids Res* 33.9 (2005), pp. 3011–3016.

[47] C. Withers-Martinez, E. P. Carpenter, F. Hackett, B. Ely, M. Sajid, M. Grainger, and M. J. Blackman. “PCR-based gene synthesis as an efficient approach for expression of the A+T-rich malaria genome”. In: *Protein Eng* 12.12 (Dec. 1999), pp. 1113–1120.

[48] R. S. Hale and G. Thompson. “Codon optimization of the gene encoding a domain from human type 1 neurofibromin protein results in a threefold improvement in expression level in Escherichia coli”. In: *Protein Expr Purif* 12.2 (Mar. 1998), pp. 185–188.

[49] E. Angov, C. J. Hillier, R. L. Kincaid, and J. A. Lyon. “Heterologous protein expression is enhanced by harmonizing the codon usage frequencies of the target gene with those of the expression host”. In: *PLoS One* 3.5 (May 2008), e2189.

[50] T. A. Thanaraj and P. Argos. “Ribosome-mediated translational pause and protein domain organization”. In: *Protein Sci* 5.8 (Aug. 1996), pp. 1594–1612.

[51] J. Zhang, Y. Fei, L. Sun, and Q. C. Zhang. “Advances and opportunities in RNA structure experimental determination and computational modeling”. In: *Nat Methods* 19.10 (Oct. 2022), pp. 1193–1207.

[52] K. Sato, M. Akiyama, and Y. Sakakibara. “RNA secondary structure prediction using deep learning with thermodynamic integration”. In: *Nat Commun* 12.1 (Feb. 2021), p. 941.

[53] Q. Zhao, Z. Zhao, X. Fan, Z. Yuan, Q. Mao, and Y. Yao. “Review of machine learning methods for RNA secondary structure prediction”. In: *PLoS Comput Biol* 17.8 (Aug. 2021), e1009291.

[54] I. Tinoco and C. Bustamante. “How RNA folds”. In: *J Mol Biol* 293.2 (Oct. 1999), pp. 271–281.

[55] R. Nussinov and A. B. Jacobson. “Fast algorithm for predicting the secondary structure of single-stranded RNA”. In: *Proc Natl Acad Sci U S A* 77.11 (Nov. 1980), pp. 6309–6313.

[56] S. R. Eddy. “How do RNA folding algorithms work?” In: *Nat Biotechnol* 22.11 (Nov. 2004), pp. 1457–1458.

[57] M. Zuker. “Mfold web server for nucleic acid folding and hybridization prediction”. In: *Nucleic Acids Res* 31.13 (July 2003), pp. 3406–3415.

[58] N. R. Markham and M. Zuker. “UNAFold: software for nucleic acid folding and hybridization”. In: *Methods Mol Biol* 453 (2008), pp. 3–31.

[59] R. Lorenz, S. H. Bernhart, C. ner Zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker. “ViennaRNA Package 2.0”. In: *Algorithms Mol Biol* 6 (Nov. 2011), p. 26.

[60] P. Gaspar, G. Moura, M. A. Santos, and J. L. Oliveira. “mRNA secondary structure optimization using a correlated stem-loop prediction”. In: *Nucleic Acids Res* 41.6 (Apr. 2013), e73.

[61] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by simulated annealing”. In: *Science* 220.4598 (May 1983), pp. 671–680.

[62] Jeff Gauthier, Antony T Vincent, Steve J Charette, and Nicolas Derome. “A brief history of bioinformatics”. In: *Briefings in Bioinformatics* 20.6 (Aug. 2018), pp. 1981–1996. ISSN: 1477-4054. DOI: `10.1093/bib/bby063`. eprint: `https://academic.oup.com/bib/article-pdf/20/6/1981/31789310/bby063.pdf`. URL: `https://doi.org/10.1093/bib/bby063`.

[63] Y. Nakamura, T. Gojobori, and T. Ikemura. “Codon usage tabulated from international DNA sequence databases: status for the year 2000”. In: *Nucleic Acids Res* 28.1 (Jan. 2000), p. 292.

[64] J. Athey, A. Alexaki, E. Osipova, A. Rostovtsev, L. V. Santana-Quintero, U. Katneni, V. Simonyan, and C. Kimchi-Sarfaty. “A new and updated resource for codon usage tables”. In: *BMC Bioinformatics* 18.1 (Sept. 2017), p. 391.