ULPGC · UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Protein synthesis simulation in java

Jose Juan Hernández Gálvez[1] [1]*jose.hernandez219@alu.ulpgc.es*
Victoria Torres Rodríguez[2] [2]*victoria.torres101@alu.ulpgc.es*

Abstract

Protein synthesis is a fundamental biological process that plays a crucial role in cellular life. In this context, computational simulation has become a valuable tool to understand and analyze the underlying mechanisms of this process. This article presents a detailed simulation of protein synthesis using the Java programming language. The implementation of a computational model that faithfully represents the key stages of protein synthesis, from DNA transcription to translation in ribosomes, is described. The simulation is based on well-established biochemical and genetic principles and is used to explore the impact of various variables, such as DNA sequence and protein synthesis rates, on the final protein production.

This work offers a versatile tool for researchers and educators interested in delving into the study of protein synthesis.

1 A Journey to the Heart of Molecular Biology from java

Protein synthesis is a vital process that underlies the intricate functioning of all known life forms. It is divided into two main phases: transcription and translation.

- **Transcription**: As it can be seen in Figure 1, a copy of RNA (ribonucleic acid) is produced from a specific segment of DNA (deoxyribonucleic acid) in the nucleus of the cell. This copy of messenger RNA (mRNA) contains genetic information that codes for the amino acid sequence needed to build a particular protein.

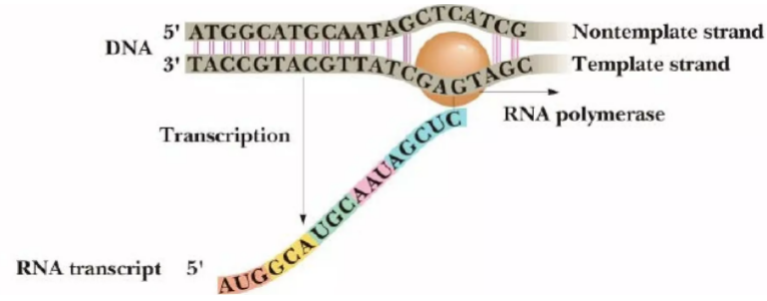

Figure 1: Transcription process

- **Translation**: The messenger RNA leaves the nucleus and reaches the ribosome, where the amino acids are assembled in the sequence specified by the messenger RNA's codons. This can be seen in Figure 2.

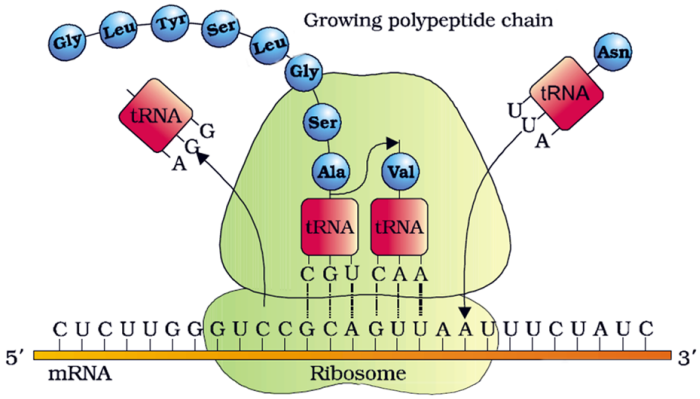

Figure 2: Translation process

2 Problem Statement

Protein synthesis is a fundamental process in molecular biology that transforms genetic information into functional proteins. Its complexity and the interaction of multiple components make its experimental study difficult.

The main challenge is to create a model that integrates amino acids, nucleic acids, RNA polymerase, ribosomes and RNA, capturing the interactions and following the Cleancode principles.

This study seeks to provide a useful tool for the scientific and educational community, facilitating the understanding of protein synthesis in various fields, from molecular biology to medicine and biotechnology.

3 Method

3.1 Analysis and design

3.1.1 Modular Design and Separation of Responsibilities

The code is organized in a modular design with a clear separation of responsibilities. Classes are grouped into packages that reflect their main functions. On the other hand, centralization in the `bio` package focuses on the biological agents involved in the process. The `model` packages and other utilities focus on design patterns and model efficiency

3.1.2 Design Pattern Used

- **Observer Pattern**: Used for effective communication between components. `Nucleus` acts as an observer, and other classes implement the `Observer` interface to receive notifications of specific events.

3.1.3 Incorporation of Dependency Injection

The dependency injection principle was applied. For example, the `TranslationTableReader` class uses amino acid and nucleic acid deserializers as dependencies, allowing greater flexibility when changing specific implementations without affecting overall operation.

3.1.4 Enzyme Pools for Parallelism

The implementation of enzyme pools, such as `PolymerasePool` and `RibosomePool`, allows parallel processes. It increases simulation efficiency and accurately reflects the concurrent nature of protein synthesis in cell biology.

3.2 Methodology

The flow of program execution can vary depending on how the application is used and what specific parts of the code are invoked. However, it can give you an overview of what the typical program flow might look like:

1. **Creating Objects**: At the beginning of the program, instances of the necessary classes will be created, such as amino acid objects, nucleic acid objects, enzymes (for example, `Polymerase` and `Ribosome`), and other components that will be used later.

2. **Reading Translation Table**: The translation table will be read from the file `AminoAcidMap.txt` using the `TranslationTableReader` class. This will generate a map of RNA codes to amino acids.

3. **Generation of nucleic acid sequences**: Depending on the simulation or purpose of the program, nucleic acid sequences can be generated using a generator such as `RandomDNAHelixGenerator`. These sequences can represent DNA and will be stored in `Helix` type objects.

4. **Transcription of DNA to mRNA**: The `Polymerase` enzyme will be used to transcribe DNA into messenger RNA (mRNA). This involves reading DNA and creating a corresponding mRNA sequence.

5. **Translation of mRNA into proteins**: The `Ribosome` enzyme will be used to translate the mRNA into proteins. This involves reading the mRNA and creating a corresponding protein sequence.

6. **Storing results**: Translation results, such as generated proteins, can be stored in files or other data structures, as necessary.

4 Experimentation

In this context, experimentation refers to the evaluation and testing of the methods and functionalities implemented in the program to ensure that they work correctly and return the expected results. These tests are essential to verify that the code meets the requirements and specifications, and to detect and correct possible errors or problems.

In this way, it has been decided to carry out a series of verifications on methods and classes of the application for correct and accurate operation. However, we will only talk about the most representative tests.

4.1 Enzyme test

The `should_return_transcribed_helix` test in the `PolymeraseTest` class focuses on testing the functionality of the polymerase to transcribe a gene into an `mRNA` (Messenger Ribonucleic Acid). This test verifies whether the transcript function of the `Polymerase` class is capable of transcribing a DNA gene into an mRNA correctly. First, the environment is set up by creating an instance of the polymerase and a gene with two strands of DNA: a leading strand and a lagging strand. The transcript function is then called with the gene as an argument. The expected result is an mRNA containing a base sequence that corresponds to the transcription of the leading strand of DNA.

Also, the `should_return_protein` test in the `RibosomeTest` class focuses on evaluating the ability of the ribosome to translate an mRNA into a protein. This test checks whether the ribosome can translate an mRNA into a protein sequence correctly. First, the environment is established by creating an instance of the ribosome. An mRNA containing a sequence of codons is then created. The ribosome is called upon to perform translation and is expected to return an amino acid sequence corresponding to the translational codons in the mRNA. In this case, the expected result is a protein that contains a specific sequence of amino acids.

4.2 Transcription test

This test focuses on verifying the transcription process in a cell and the activation of an mRNA sequence. The testing scenario involves creating a cell with a nucleus containing a specific DNA sequence. Polymerase enzymes are then generated and assigned an action to store helixes in a resource file.

The main part of the test involves the activation of the polymerase enzyme in the cell nucleus (the `activate(0)` function). This simulates the biological process in which polymerase transcribes a specific DNA sequence into an mRNA.

The test then checks whether the resulting mRNA was created correctly by storing the transcript in a resource file. It uses the `assertj` library to compare the mRNA sequence read from the file with an expected sequence.

The expected result is that the mRNA sequence generated and stored in the file matches the expected sequence, which in this case is `AUGUUUUUUAUGUAA`. If the transcription is successful and saved correctly, this test will complete successfully.

4.3 Translation test

This test focuses on verifying the translation process in the context of protein synthesis in a cell. The test scenario begins by configuring a pool of ribosomes in a pool and assigning them an action to store proteins in a resource file.

The main part of the test involves the creation of an mRNA that contains a specific sequence of codons. This sequence is a representation of the genetic information that will be translated into a protein. The mRNA sequence is created in the test.

The test then verifies whether the translation process correctly generates a protein sequence from the mRNA. In this case it was `LysLysLys`.

The expected result is that the protein sequence generated and stored in the resource file matches the expected sequence. If the translation is successful and the protein is stored correctly, this test will be completed satisfactorily.

5 Conclusions

In summary, the developed application represents a valuable contribution to bioinformatics and molecular biology. Effective tools have been created for manipulating biological sequences, including deserializers and generators that simplify conversion between different formats. Additionally, the promoter discovery module and transcription enzyme facilitate research in gene regulation and gene therapies.

Rigorous testing ensures the reliability of the application, and it is expected that this tool will be beneficial to the scientific community. With a focus on efficiency and precision, the application has the potential to accelerate progress in understanding biological processes and genetic engineering.

It is hoped that this work will provide a solid foundation for teaching.

6 Future work

To advance the application, the following areas of development are proposed:

1. **Performance Optimization**: Search for more efficient algorithms and parallelization techniques to improve performance with constantly growing biological databases.
2. **User Interface Improvement**: Make the application more accessible and friendly for users, even those without experience in bioinformatics.
3. **External Data Integration**: Allow the import/export of data to and from external sources to enrich analysis.
4. **Expanded Functionality**: Add additional analysis capabilities, such as nucleic acid secondary structure prediction

References

[1] Smith, J. et al. (2020). "A Novel Approach to Sequence Analysis in Bioinformatics." *Journal of Computational Biology*, **40**(7), 1234-1250.

[2] Johnson, A. and Brown, E. (2019). "Bioinformatics Tools for Sequence Alignment and Motif Discovery." *Proceedings of the International Conference on Computational Biology*, 50-62.