**Department of Electrical & Computer Engineering (ECE), Concordia University**

**APPLIED MACHINE LEARNING & EVOLUTIONARY ALGORITHMS**

**COEN 432 (6321): Fall 2025**

**Assignment 1: Inverse RNA Folding Using an Evolutionary Algorithm**

**(due date: Fri 11 Oct @ 23hr55, via Moodle)**

#### 1. Problem Description:

The challenge of inverse RNA folding is to design an RNA sequence that folds (with high probability) into a specific, predetermined secondary structure. This is a fundamental problem in RNA science and engineering. In this assignment, you will develop a program based on an Evolutionary Algorithm (EA) to tackle this problem.

Your primary task is to create an EA that can discover a diverse set of RNA sequences that **perfectly satisfy the given sequence constraints** while folding into a structure that is **as close as possible to the target structural constraint**.

Crucially, while the **sequence constraints must never be violated**, the **structural constraint acts as a target** for your fitness function. The fitness (or loss) of any given candidate sequence is purely a function of how different its actual folded structure is from the required one. The definition of **sequence constraints** and **structural constraints** are given below:

1. **Sequence Constraints:** Restrictions on which nucleotide base (A, U, C, G) can appear at certain positions in the sequence. These will be provided using the standard IUPAC notation (see section 3.1 in the sequel).
2. **Structural Constraints:** The required secondary structure that the sequence must fold into. This will be provided using the dot-bracket notation (also, see 3.1).

The goal of your algorithm is not just to find a single valid sequence, but to generate a diverse set of five unique sequences for each problem instance, all of which satisfy the given constraints.

To evaluate the fitness of candidate sequences (i.e., to check if a sequence folds into the target structure), you will use an external, pre-existing folding software. You will be given the necessary code wrapper or function call to interact with this software. This means you do not need to implement the RNA folding prediction yourself. **Your focus should be entirely on the design and implementation of your Evolutionary Algorithm**, including key components such as:

* Individual representation
* Population initialization
* Fitness evaluation
* Selection, crossover, and mutation operators

#### 2. Program Requirements

To ensure smooth automated testing and grading, your program must strictly adhere to the following requirements:

* **Programming Language:** Your program must be written in **Python**.
* **User Input:** Create a file named config.py in the root directory of your project. This file will serve as the central control panel for your EA. Below is a structured template demonstrating the types of parameters you should include. You must organize your parameters in a similar fashion, using clear variable names and comments.
* Example config.py Template:

|  |
| --- |
| Python TARGET\_STRUCTURE = "((((...))))..." IUPAC\_CONSTRAINT = "NNNNNNNNNNNNN..." IPKNOT\_PATH = "/path/to/your/ipknot" POPULATION\_SIZE = 100 GENERATIONS = 500 etc........ |

* **Code Identification:** You must include a comment at the very top of your main Python file (the first line) that lists the full names and student IDs of all team members.
* Example:

|  |
| --- |
| Python *# John Smith (12345678), Jane Doe (87654321)* import numpy as np |

#### 3. Data Representation

Your program must be able to process a CSV (Comma-Separated Values) input file containing multiple problem instances and produce a corresponding CSV output file with your results.

**3.1 Input**

You will be provided with a single CSV file (e.g., EA\_Assignment\_1\_constraints.csv) that contains all the problem instances to be solved. Your program must read this file and process each row as a separate, independent task.

* **Sequence Constraint (IUPAC Notation):**
  + A string representing the constraints on the nucleotide sequence. It uses IUPAC nucleotide codes to specify which bases are allowed at each position.
  + For example, the string NNNNAGCUWSSNN means:
    - N: Any base (A, U, C, G) is allowed.
    - A, G, C, U: Only the specified base is allowed.
    - W: "Weak" interaction, either A or U is allowed.
    - S: "Strong" interaction, either C or G is allowed.
  + Your program should correctly interpret these codes to ensure that any generated sequence respects these positional constraints.
  + If you wish to access more information, you can find detailed IUPAC codes in this website: <https://www.bioinformatics.org/sms/iupac.html>
* **Structure Constraint (Dot-Bracket Notation):**
  + A string of the same length as the sequence constraint, composed of parentheses(), {}, [] and dots ..
  + This notation defines the target secondary structure:
    - An open bracket such as (, { or [: A base at this position must form a pair with a base at a corresponding closing bracket.
    - A closing bracket such as ), } or ]: A base that is paired with an open bracket (above).
    - A dot.: A base at this position is not paired with any other base.
  + For example, the structure (((((...))))) represents a simple hairpin loop with a stem of length 5 (pairs of brackets) and an end-loop of size 3 (dots). To be clear, the first five bases pair with the last five bases, and the middle three bases are unpaired.

**3.2 Output**

After running your Evolutionary Algorithm, an output file from a successful run might contain the following sequences. These sequences **perfectly match the sequence constraint** (e.g.  **NNSNNS**) and have been evolved to have a **very high fitness score**, meaning their predicted secondary structure is very close to (or identical to) the target structure (....). Your program must generate an output .csv file that **exactly matches the format** of the template file provided on Moodle (EA\_Assignment\_1\_output.csv). This file will contain the sequences discovered by your algorithm. Please note that RNA sequences have a direction, which is signified by a 5’ sign on one side (conventionally, the start) and a 3’ sign at the end. If you follow convention, there is no need to output the 5’ sign.

#### 3.3 A Complete Example

To help you better understand the task requirements, here is a complete example that goes from input to output and includes a diversity analysis.

#### Input

Assume your program receives the following two constraints as input:

|  |
| --- |
| .csv  id,2d\_structure,IUPAC  3.1,(....),NNSNNS |

These constraints imply that:

1. The sequence length is 6.
2. The bases at position 1 and position 6 must form a pair (e.g., A-U, G-C).
3. The bases at positions 3 and 6 must be "strong" (G or C).
4. The bases at all other positions (2, 4, 5) can be any nucleotide.

Your EA will then run to find sequences that meet these criteria. Let's assume it finds many (more than 20) valid solutions, such as CGGAAG, CGCACG, GGGAAC, AUCGAU, and GCCGGC. Your diversity-maintenance mechanism should help you select five that are substantially different from each other.

#### Output

The corresponding output for id "3.1" would be a single row in your results file:

|  |
| --- |
| id,result\_1,result\_2,result\_3,result\_4,result\_5 3.1,CGGAAG,CGCACG,GGGAAC,AUCGAU,GCCGGC |

#### 4. Submission Instructions

Please adhere strictly to the following submission guidelines. Any deviation may result in a score of zero from the automated grading script.

You must submit a **single ZIP file** named exactly as **Assignment1.zip**. Each team should only submit one file.

The Assignment1.zip file must contain the following items at its top level:

1. **Output File:** A csv file named exactly EA\_Assignment\_1\_output.csv. This file should contain the best set of valid RNA sequences discovered by your algorithm using the parameters you found to be most effective.
2. **Source Code Folder:** A folder containing all your Python (.py) source code.
3. **README File:** A file named README.md or README.txt. This file must provide clear, step-by-step instructions on how to run your program from the command line. If your program has any external dependencies (libraries that are not standard in Python), you must list them and provide the pip install commands.
4. **Report:** A report in PDF format, named report.pdf, with a maximum length of two pages.

**Late Submission Policy:**

* Up to 24 hours of delay in submission leads to a **20% deduction** from your final mark.
* If the delay is greater than 24 hours, your assignment may not be marked at all (resulting in a score of 0).

#### 5. Report Guidelines

Your report (maximum 2 pages) is a critical part of this assignment and should professionally document your work. It must include the following sections:

* **EA Design and Implementation:**
  + Clearly describe the EA components you chose and implemented. Which selection strategy did you use (e.g., tournament, roulette wheel)? What were your crossover and mutation operators? Did you incorporate techniques like elitism?
* **Experimentation and Strategy:**
  + Show how you experimented to improve your results. Discuss the different parameters (e.g., population size, generation count, mutation/crossover rates) you tested and how they affected the outcome. If you tried different algorithmic strategies, describe them and explain why you ultimately chose your final approach. You may use tables or plots to illustrate your findings.
* **Analysis of Results:**
  + Discuss the final results contained in your EA\_Assignment\_1\_output.csv. For each problem instance, how effectively was your algorithm able to find a set of five diverse and valid solutions? You can also discuss the size of the total valid solution pool your EA discovered before selecting the final five, as this can indicate the exploratory power of your algorithm.

#### 6. Grading Criteria

This assignment will be graded out of 100 points, based on the four criteria detailed below. Note that both your code and your submitted results will be evaluated.

|  |  |  |
| --- | --- | --- |
| Category | Weight | Description |
| Efficiency | 30% | This criterion measures the computational efficiency of your algorithm in finding high-quality solutions. Your algorithm's performance will be compared relatively to the performance of other submissions. |
| In-line Documentation | 20% | This assesses the quality and clarity of the comments within your source code. Good documentation should explain the logic behind your implementation, making it easy for others to understand your design choices. |
| Correctness | 25% | This evaluates the quality and validity of the final sequences your program produces. The score is based on:  1. **Strict adherence to Sequence Constraints:** All submitted sequences **must** perfectly satisfy the IUPAC constraints. Any sequence that violates this is invalid.  2. **Structural Fitness:** The primary measure is how closely your final sequences fold into the target dot-bracket structure. Higher fitness scores will result in a higher mark.  3. **Completion**: The ability to generate a full set of five valid, high-fitness sequences for each problem instance defined in the input file. |
| Diversity | 25% | This assesses the variety among the valid sequences you discovered. The goal is to find sequences that are significantly different from one another, demonstrating that your algorithm has effectively explored the solution space. An algorithm that finds many similar sequences will score lower than an algorithm that finds an equal number of highly distinct sequences. |

#### Measuring Diversity

In your report, you will need to analyze and demonstrate the diversity of the sequence set you have found. A good way to quantify this is by calculating the average normalized Hamming distance between the sequences.

The Hamming distance between two strings of equal length is the number of positions at which the corresponding characters are different. For example, the Hamming distance between CGGAAC and CGCACA is 3 (because the 3rd, 5th, and 6th positions are different).

The normalized Hamming distance is simply the Hamming distance divided by the total length of the sequence. This constrains the diversity score to a value between 0 and 1.

Example Calculation: Let's use the first three sequences from the output above as an example:

* Seq A: CGGAAC
* Seq B: CGCACA
* Seq C: GGGAAC

1. Calculate the pairwise normalized Hamming distance (sequence length is 6):
   * Distance(A, B) = 3 / 6 = 0.5
   * Distance(A, C) = 1 / 6 ≈ 0.167
   * Distance(B, C) = 4 / 6 ≈ 0.667
2. Calculate the average diversity for the set:
   * Average Diversity = (0.5 + 0.167 + 0.667) / 3 ≈ 0.445

In your report, you can use this method to quantify and compare the diversity of results produced by different algorithms or parameter settings, thereby demonstrating the effectiveness of your chosen strategy.

#### 7. Questions and Clarifications

For clarifications of the content of the assignment or submission procedure, please e-mail the TA: Kaiyu Nie (kaiyu.nie@mail.concordia.ca).