Assessing the predictive capacity of the Nussinov model

Warm-up

A secondary structure can be given as a list of base-pairs \( l \) over a sequence of length \( n \). This first assignment consists in printing the structure back as a well-parenthesized expression.

Implement a function \( \text{displaySecStr} \), which takes as input a pair \((l, n)\), and returns the well-parenthesized expression. **Remark:** Bases will be numbered starting from 0.

**Example:**

\[
\text{displaySS}([((2,8),(3,7)],10) \rightarrow "..((...)\)."
\]

Parsing secondary structures

We wish to extract a list of base-pairs from the well-parenthesized notation used in the Vienna package. In this notation, two positions are involved in a base-pair if and only if they correspond to matched opening (and closing) parentheses. Unpaired positions are denoted by dots.

Implement a \( \text{parseSS} \) function, which takes a string \( \text{struct} \) as input, and returns a pair \((l, n)\), where \( l \) is the list of base-pairs and \( n \) is the length of the string.

**Example:**

\[
\text{parseSS}("((.(.)))") \rightarrow ([0,9),(1,8),(3,7]),10)
\]

The correctness of this function will be verified using its inverse \( \text{parseSS} \), defined above. Namely, for any well-parenthesized sequence \( \text{seq} \), one should have:

\[
\text{displaySS}([\text{parseSS}(\text{seq}),\text{len}(\text{seq})] = \text{seq}
\]

Counting compatible structures

Implement a counting variant of the Nussinov algorithm (setting \( \theta = 1 \) for the sake of simplicity\(^1\)), into a function \( \text{countSS} \), which will return the number of secondary structures compatible with a given sequence. To that purpose, it is sufficient to replace any

\(^1\)Remind that \( \theta \) denotes the minimal distance of paired positions.
occurrences of \texttt{min} and \texttt{+} in the dynamic programming equation, seen in the first lecture, with \texttt{+} and \texttt{*} respectively.

The function takes as input an RNA sequence \(r\) and a boolean \texttt{debug}. In the normal mode \texttt{debug==False}, the function returns the number of secondary structures which only induces base-pairs of the type \texttt{GC}, \texttt{AU} or \texttt{GU} on \(r\). When \texttt{debug==True}, it relaxes the base-pairing condition, and returns the total number of secondary structures of length \(\text{len}(r)\).

**Example:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{countSS(&quot;A&quot;)}</td>
<td>1</td>
</tr>
<tr>
<td>\texttt{countSS(&quot;CAG&quot;)}</td>
<td>2</td>
</tr>
<tr>
<td>\texttt{countSS(&quot;CAGU&quot;)}</td>
<td>3</td>
</tr>
<tr>
<td>\texttt{countSS(&quot;AAAA&quot;,True)}</td>
<td>4</td>
</tr>
<tr>
<td>\texttt{countSS(&quot;AAAAAAA&quot;,True)}</td>
<td>82</td>
</tr>
</tbody>
</table>

You may use the number of secondary structures for sequence lengths up to 12:

<table>
<thead>
<tr>
<th>len(r)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>17</td>
<td>37</td>
<td>82</td>
<td>185</td>
<td>423</td>
<td>978</td>
<td>2283</td>
</tr>
</tbody>
</table>

to check your implementation.

**Nussinov**

Now, we move on to the implementation of our variant of the Nussinov algorithm. To that purpose, one must duplicate the code of the \texttt{countSS} function, and adapt it into a function \texttt{fillMatrix}, which precomputes the energy of the minimal free-energy structure with respect to a Nussinov energy model \(\Delta G(\text{AU}) = \Delta G(\text{GC}) = \Delta G(\text{GU}) = -1\). Additionally, the code will be modified to account for general minimal distances \(\theta\) between matching positions.

The \texttt{fillMatrix} function takes a sequence \(r\) and a \texttt{theta} integer value as input, and return a filled dynamic programming matrix \(\text{tab}\) such that \(\text{tab}[i][j]\) is the MFE for any structure compatible with the interval \([i,j]\).

**Example:**

Entree : \texttt{fillMatrix(CCCCUUUUGGGGG,3)}

<table>
<thead>
<tr>
<th>\text{tab} =</th>
<th>0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1. -2. -3. -4. -5.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1. -2. -3. -4. -4.</td>
</tr>
<tr>
<td></td>
<td>0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1. -2. -3. -3. -4.</td>
</tr>
<tr>
<td></td>
<td>0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1. -2. -2. -3. -3.</td>
</tr>
<tr>
<td></td>
<td>0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1. -1. -2. -2. -3.</td>
</tr>
</tbody>
</table>

Page 2
Once tested, the matrix-filling function will be supplemented by a traceback (possibly recursive) function, which builds (one of) the MFE structure(s). This function takes a DP matrix tab produced by fillMatrix as input, and returns a set of non-crossing base-pairs associated with the minimal free-energy.

Example:

\[
\text{Input:} \quad \text{traceback(tab)} \quad \rightarrow \quad \{(0,12),(1,11),(2,10),(3,9),(4,8)\}
\]
\[
\text{Output:} \quad \text{displaySS(traceback(tab))} \quad \rightarrow \quad "((((...))))"
\]

Finally, combine these two functions into a nussinov function which takes an RNA sequence as input, and returns a minimal free-energy structure in the Nussinov model.

**Half-time summary**

The time has now come to compare the predictive capacities of our – minimally simple – RNA folding software with state-of-the-art tools. To that end, we will use the Vienna package, a suite of tools maintained by Ronny Lorenz at the Theoretical Biochemistry Institute of Vienna. The package includes RNAEval, which computes the free-energy of a given secondary structure, and RNAFold which uses dynamic programming to compute the MFE structure for a given RNA sequence, both with respect to the latest version of the Turner energy model.

We implemented two Python wrappers for the runRNAEval and runRNAFold tools through the following functions:

- \text{runRNAFold(seq)} takes a sequence seq, and return a pair (\text{mfe},E), where mfe is the MFE secondary structure for \text{seq}, given as a base-pair list, and E is its energy.
- \text{runRNAEval(seq,str)} takes a sequence seq and a secondary structure str formatted as a well-parenthesized expression, and returns the associated energy.

You should start by downloading the wrappers (+ data) at:

Model discrepancies

Firstly, implement a function `compareSS` which takes as input two structures $S$ et $S'$ (represented as base-pair lists), and returns the number of common base-pairs $|S \cap S'|$. We recommend using Python sets for a single-line implementation.

Predictive performances

Use the `compareSS` function to implement a benchmark function, which takes an RNA sequence $\omega$ as input, along with its (assumed known) native structure $S$, and returns the proportion of base-pairs correctly predicted by the algorithms `nussinov` and `RN AFold` on a reference set of sequence/structures. The dataset was gathered by D. H. Mathews, and can be downloaded from:

http://www.lix.polytechnique.fr/~ponty/enseignement/MathewsRNASorted.faa