

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

Example:

| | | |
|--|---|---------------------------|
| Input | | Output |
| <code>displaySS([(2,8), (3,7)], 10)</code> | → | <code>"..((...))."</code> |

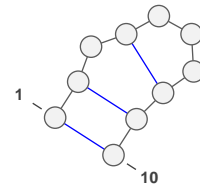
Parsing secondary structures

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 `parseSS(struct)` function, which takes a string `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:

| | | |
|------------------------------------|---|--|
| Input | | Output |
| <code>parseSS("((.(...)))")</code> | → | <code>([(0,9), (1,8), (3,7)], 10)</code> |



The corection of this function will be verified using its *inverse* function `parseSS`, defined above. Namely, for any well-parenthesized sequence `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 simplicity¹), into a function `countSS`, which will return the number of secondary structures compatible with a given sequence. To that purpose, it is sufficient to replace any occurrences of `min` and `+` in the dynamic programming equation, seen in the first lecture, with `+` and `*` respectively.

¹Remind that θ denotes the minimal distance of paired positions.

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

Example:

| Input | Output |
|--|--------|
| <code>countSS("A")</code> | → 1 |
| <code>countSS("CAG")</code> | → 2 |
| <code>countSS("CAGU")</code> | → 3 |
| <code>countSS("AAAA", True)</code> | → 4 |
| <code>countSS("AAAAAAAA", True)</code> | → 82 |

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

| len(r) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|--------|---|---|---|---|---|----|----|----|-----|-----|-----|------|
| S | 1 | 1 | 2 | 4 | 8 | 17 | 37 | 82 | 185 | 423 | 978 | 2283 |

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 `countSS` function, and adapt it into a function `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 θ between matching positions.

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

Example:

```
Entree : fillMatrix(CCCUUUUGGGG,3)
Sortie :
tab = [[ 0.  0.  0.  0.  0.  0.  0.  0. -1. -2. -3. -4. -5.]
       [ 0.  0.  0.  0.  0.  0.  0.  0. -1. -2. -3. -4. -4.]
       [ 0.  0.  0.  0.  0.  0.  0.  0. -1. -2. -3. -3. -4.]
       [ 0.  0.  0.  0.  0.  0.  0.  0. -1. -2. -2. -3. -3.]
       [ 0.  0.  0.  0.  0.  0.  0.  0. -1. -1. -2. -2. -3.]
       [ 0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1. -2. -2.]
       [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1. -2.]
       [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1.]
       [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
       [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

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:

| Input | Output |
|--|--|
| <code>traceback(tab)</code> | $\rightarrow [(0,12), (1,11), (2,10), (3,9), (4,8)]$ |
| <code>displaySS(traceback(tab))</code> | $\rightarrow "((((((...))))))"$ |

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 in Vienna, Austria. Please download and install the Vienna package from binaries, as described at:

<https://www.tbi.univie.ac.at/RNA>

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:

- `runRNAFold(seq)` takes a sequence `seq`, and return a pair (mfe, E) , where `mfe` is the MFE secondary structure for `seq`, given as a base-pair list, and `E` is its energy.
- `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:

<http://www.lix.polytechnique.fr/~ponty/enseignement/ViennaWrapperCheat.py>
<http://www.lix.polytechnique.fr/~ponty/enseignement/RNAfoldMathews.dat>

Model discrepancies

Implement a function `delta`, which takes an RNA sequence ω as input, and returns the difference of free-energies (as returned by `RNAEval`) between the secondary structures returned by your implementation of `nussinov` and `RNAFold` respectively.

Why this energy is always positive?

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 ω as input, along with its (assumed known) native structure S , and returns the proportion of base-pairs correctly predicted by the algorithms `nussinov` and `RNAFold` 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>