Appendix A

Nomenclature

A.0.1 Important Variables

All of these variables are stored as global variables. To modify a global variable called myvar, first declare `global myvar`, then future modification will happen to the variable declared in the global namespace.

- `L`

  A dictionary that contains, for each base in the sequence, a list of indices of bases it can pair with to the left.

- `N`

  The length of the sequence.

- `R`

  A dictionary that contains, for each base in the sequence, a list of indices of bases it can pair with to the right.

- `T`

  The temperature of experiment. For the tables used, this is 273.15 + 37 (K).

- `allowedPairs`

  A set of tuples representing the bases that can legally pair. Generally the WC pairs plus GU wobble pairs.

- `base_dict`

  A dictionary containing as keys all the leftmost paired bases, and as values their corresponding paired base.

- `beta`

  $\frac{1}{RT}$. With the usual values for R and T, this is 1.6225042730497252.

- `debug_energy`

  If true, prints out detailed energy information about every step of the deterministic traceback.

- `debug_structure`

  If true, prints out detailed energy information about every step of the deterministic traceback.

- `e_index`

  For a given d and j, corresponds to the largest possible index of R[d] such that R[d] ¡ j is still true.

- `l`

  30. The maximum number of bases we allow in an internal loop.

- `olds`

  The original sequence.

- `r`

  0.0019872036cal/mol·K. The gas constant.

- `s`

  The sequence, repeated twice. For example, if the original sequence was `"GAGACUCU"`, then we would have `s = "GAGACUCUGAGACUCU"`. We need s to be twice as long so we can properly handle the wraparound case.

- `scale`

  −0.34kcal/mol. Equal to the expected free energy bonus for each additional base.

- `zero`

  Stores whether or not all the energy tables should be set to value 0. If they *are* all set to 0, then each structure contributes 1 to the partition function, and the value of the overall partition function is equal to the number of distinct folds for that sequence.

A.0.2 Energy Tables

- `CLoops`

  A dictionary that maps loops made entirely of C's to the corresponding energy penalty.

- `dangle3Energy`

  A dictionary that maps 3' dangles to their energies.

- `dangle5Energy`

  A dictionary that maps 5' dangles to their energies.

- `ea`

  The energy penalty for initiating a multibranch.

- `eb`

  The energy penalty for adding an unpaired base to a multibranch.

- `ec`

  The energy bonus for pairing two bases in a multibranch.

- `endPenalty`

  A dictionary that maps a helical-stack-closing-pair to its corresponding energy penalty.

- `guClosureEnergy`

  A dictionary that maps a special hairpin-closing sequence to its corresponding energy penalty.

- `hairpinEnergy`

  A dictionary that maps size of a hairpin to its corresponding energy penalty.

- `int11`

  A dictionary that maps 1×1 loops to their corresponding energy penalty. Takes all six bases involved as key.

- `int21`

  A dictionary that maps 2×1 loops to their corresponding energy penalty. Takes all seven bases involved as key.

- `int22`

  A dictionary that maps 2×2 loops to their corresponding energy penalty. Takes all eight bases involved as key.

- `internalLoopEnergy`

  A dictionary that maps loop size to its corresponding energy penalty.

- `specialLoops`

  A dictionary that maps certain special hairpin loops to their additional energy bonus.

- `stackEnergy`

  A dictionary that maps the four bases involved in a stack to the corresponding energy bonus.

- `terminalMismatchEEnergy`

  If there is a base pair ij and it is the outermost loop, then we add on to the loop energy a penalty based on what i, $i+1$, $j-1$ and j are.

- `terminalMismatchHEnergy`

  If there is a base pair ij and it closes a hairpin, then we add on to the loop energy a penalty based on what i, $i+1$, $j-1$ and j are.

- `terminalMismatchMEnergy`

  If there is a base pair ij and it closes a hairpin, then we add on to the loop energy a penalty based on what i, $i+1$, $j-1$ and j are.

A.0.3 Functions and Flags

- `count_structures`

  Given a sequence, counts the number of microstates it can assume. Can choose to set `no_dangles` flag to prohibit terminal stacking.

- `exists`

  Given a string x, returns `True` if x is the name of a local or global variable, `False` otherwise.

- `fillAllowedPairsList`

  Fills two lists called `R` and `L` which hold the positions of bases each base as allowed to pair with. If given a list of lists of pairs as argument, then it uses that instead of loading all legal pairs.

- `fillEnergyTables`

  Fills all the loaded energy tables. Takes two parameters, `zero_energies` and `no_dangles`, both booleans, which can be used to set all the structure energies to 0 and/or to not allow terminal stacking. `force` tells the function to re-fill the tables.

- `fillMatrices1`

  Fills the top-right half of the `Zb`, `Z1`, `Z2` 2D arrays. The calculated points correspond to energies of states in the forward fill.

- `fillMatrices2`

  Fills the bottom-left half of the `Zb`, `Z1`, `Z2` 2D arrays. The calculated points correspond to energies of states in the wraparound.

- `fillZ3`

  Fills the `Z3` 1D array.

- `fillZ5`

  Fills the `Z5` 1D array.

- `filleIndex`

  Fills the `eIndex` 2D array. For any given `d` and a given `j`, `eIndex[d, j]` stores the index in `R[d]` of the allowed pair of `d` closest (while staying to the left of) to `j`. This array is used to speed up calculation of the internal loop energies.

- `folds_given_length`

  Given a length l and number of sequences to generate s, calculates and saves the number of folds each of the s sequences of length l has, in a Python list format stored at a designated filename.

- `generate_sequence`

  Given a length, generates a sequence of RNA from a uniform distribution of the bases `A`, `C`, `G`, `U`.

- `get_next_loop`

  Given the location of a base of the sequence, returns the location of the closest base to the right of it which starts a new pair.

- `get_num_structures`

  Given a base which closes a loop, calculates the number of structures (0 for hairpin, 1 for internal loop, 2+ for multibranch) nested inside the base and its partner.

- `get_structs`

  Given a sequence, returns, in exponential time, the number of different folds, the folds in dot-bracket notation, and the folds in `pair_dict` and `dangle_list` notation.

- `get_thermal_ensemble`

  Given a sequence, generates the thermal ensemble of that sequence by sampling from the partition function a specified number of times. Additionally, one can set the program to set all energies to zero, and/or to not allow terminal stacking. The output is saved in a user-specified filename.

- `loadAssymetry`

  Loads the asymmetry penalties for bulge loops into the `asymmetry` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadBulgeLoopEnergy`

  Loads the bulge loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadCLoops`

  Loads the Cloop energies into `CLoops` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadDangle3Energy`

  Loads the 3' dangle stacking into `dangle3Energy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadDangle5Energy`

  Loads the 5' dangle stacking into `dangle5Energy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadHairpinEnergy`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadInt11`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadInt21`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadInt22`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadInternalLoopEnergy`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadSequence`

  Loads the loop energies into `bulgeLoopEnergy` dictionary. Can set all energy values to zero with the `zero_energies` flag.

- `loadSpecialLoops`

  Loads the special lop energies into `specialLoops` dictionary. Currently limited to tetraloops. Can set all energy values to zero with the `zero_energies` flag.

- `loadStackEnergy`

  Can set all energy values to zero with the `zero_energies` flag.

- `loadTerminalMismatchEEnergy`

  Can set all energy values to zero with the `zero_energies` flag.

- `loadTerminalMismatchHEnergy`

  Can set all energy values to zero with the `zero_energies` flag.

- `loadTerminalMismatchMEnergy`

  Can set all energy values to zero with the `zero_energies` flag.

- `load_multibranch_penalties`

  Can set all energy values to zero with the `zero_energies` flag.

- `load_penalty_dicts`

  Can set all energy values to zero with the `zero_energies` flag.

- `lock_n_load`

  Given a sequence, calls `fillEnergyTables`, `fillAllowedPairsList`, `filleIndex`, `fillZ3`, `fillZ5`, `fillMatrices2`. Can be optionally set to not do last two calls, which are only necessary if we want to calculate pairing probabilities. Can also call with `zero_energies` and/or `no_dangles` flags to set energy tables to zero and/or not allow terminal stacking.

- `rand_sample`

  Generates a random fold from the partition function, using the globals `pair_dict` and `dangle_list`.

- `read_ct`

  Reads in a `.ct` file and stores the information in the globals `pair_dict` and `dangle_list`.

- `save_structure_count_list`

  Calls `count_structures` for a number of different lengths and saves the information to file.

- `traceZ1`

  Stochastic traceback for `Z1`.

- `traceZ1d`

  Deterministic traceback for `Z1`.

- `traceZ2`

  Stochastic traceback for `Z2`.

- `traceZ3`

  Stochastic traceback for `Z3`.

- `traceZ3d`

  Deterministic traceback for `Z3d`.

- `traceZb`

  Stochastic traceback for `Zb`.

- `traceZbd`

  Deterministic traceback for `Zbd`.

Tests

- `compare_Z`

  Generates a sequence or a length of sequence to generate, calculates and prints the partition function using MacroFold, UNAFold, and RNAFold.

- `compare_many_probs`

  Calls `compare_probs` for many different sequences of given length.

- `compare_probs`

  Compares frequency of every possible pair in every possible structure, calculated two ways: first, exhaustive exponential enumeration; second, with the partition function with zero-energy tables. Useful for testing the wraparound code once one is certain the forward-fill works.

- `debug_mode`

  For debugging. Turns on verbose structural and energetic descriptions of the deterministic traceback.

- `debug_structure`

  Turns on verbose structural descriptions of the deterministic traceback.

- `find_mismatches`

  Used in testing. Given a number of sequences to try, and a length for those sequences to be, counts the number of distinct sequences found both by exponentially enumerating all possible sequences, and by calculating the partition function using the zero-energy tables. Raises an exception if the numbers are not equal.

- `get_microstates`

  Given a sequence, recursively generates all its possible folds. Then checks the energy of each fold, as calculated by MacroFold and by UNAFold, and prints any differences alongside the corresponding fold.

- `run_tests`

  Given a length l and a number of trails n, generates a random sequence of length l, randomly folds it with UNAFold, and then compares the energy of the microstate as calculated by MacroFold and by UNAFold. Raises exception if different. Repeats for n different sequences.

Matrices

And a list of the matrices:

- `Z1`

  A 2D NumPy array. Z1[i, j] contains the partition function of all structures between i and j such that there is at least one piece of structure between i and j. Z1[j, i] contains the partition function of all structures outside i and j such that there is at least one piece of structure outside i and j.

- `Z2`

  A 2D NumPy array. Z2[i, j] contains the partition function of all structures between i and j such that there are at least two pieces of structure between i and j. Z2[j, i] contains the partition function of all structures outside i and j such that there are at least two pieces of structure outside i and j.

- `Z3`

  A 1D NumPy array. Z3[i] contains the partition function of all structures between i and the end of the sequence, inclusive.

- `Z5`

  A 1D NumPy array. Z5[j] contains the partition function of all structures between the start of the sequence and j, inclusive.

- `Zb`

  A 2D NumPy array. Zb[i, j] contains the partition function of all structures inside and including an ij pair.

A.0.4 General Procedure

Given a sequence $w = a_0 a_1 \dots a_{n-1}$, running `lock_n_load(w)` will do, in the following order:

- Set `olds` equal to `w`
- Set `N` equal to the length of `w`
- Set `s` equal to `w` repeated twice
- Initialize the matrices
- Fill all the energy dictionaries if they are not already
- Fill the `R` and `L` dictionaries
- Fill the `e_index` for each base
- Fill the matrices

Most desirable quantities are derivable from the information stored in the matrices.

Appendix B

Recursion relations

B.1 Principal Recursion Relations

B.1.1 Notes

- All arrays are 0-indexed.
- $R[i]$ is a list of indices of potential pairs to the right of i starting at position $min(i{+}4, N)$ and working up to position $i + N - 4$.
- $L[i]$ is a list of indices of potential pairs to the left of i starting at position $i - 4$ and working down to position 0

B.1.2 Z_3

We loop from $i = N - 5$ to $i = 0$.

$$\text{Base Cases: } Z_3(i) = \begin{cases} 0 & \text{when } i \geq N \\ 1 & \text{when } N-1 \geq i \geq N-4 \end{cases}$$

$$\begin{aligned} Z_3(i) = & \overbrace{Z_3(i+1)}^{\text{no pair}} \\ & + \overbrace{\sum_{k \in R[i]}^{N-3} Z^b(i,k)\ Z_3(k+1)}^{\text{no dangle}} \\ & + \overbrace{\sum_{k \in R[i]}^{N-3} Z^b(i,k)\ Z_{3'D}(i,k)\ Z_3(k+2)}^{\text{3' (right) dangle}} \\ & + \overbrace{\sum_{k \in R[i+1]}^{N-3} Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ Z_3(k+1)}^{\text{5' (left) dangle}} \\ & + \overbrace{\sum_{k \in R[i+1]}^{N-3} Z^b(i+1,k)\ Z_{ts}(i+1,k)\ Z_3(k+2)}^{\text{terminal stack}} \\ & + \overbrace{Z^b(i,N-2) + Z^b(i,N-2) Z_{3'D}(i,N-2)}^{k=N-2} \\ & + \overbrace{Z^b(i+1,N-2) Z_{5'D}(i+1,N-2) + Z^b(i+1,N-2) Z_{ts}(i+1,N-2)}^{k=N-2} \\ & + \overbrace{Z^b(i,N-1) + Z^b(i+1,N-1) Z_{5'D}(i+1,N-1)}^{k=N-1} \end{aligned}$$

B.1.3 Z_5

We loop from $j = 4$ to $j = N - 1$.

$$\text{Base Cases: } Z_5(j) = \begin{cases} 0 & \text{when } j < 0 \\ 1 & \text{when } 0 \leq j \leq 3 \end{cases}$$

$$\begin{aligned} Z_5(j) = & \overbrace{Z_5(j-1)}^{\text{no pair}} \\ & + \overbrace{\sum_{k \in L[j]}^{2} Z^b(k,j)\ Z_5(k-1)}^{\text{no dangle}} \\ & + \overbrace{\sum_{k \in L[j-1]}^{2} Z^b(k,j-1)\ Z_{3'D}(k,j-1)\ Z_5(k-1)}^{\text{3' (right) dangle}} \\ & + \overbrace{\sum_{k \in L[j]}^{2} Z^b(k,j)\ Z_{5'D}(k,j)\ Z_5(k-2)}^{\text{5' (left) dangle}} \\ & + \overbrace{\sum_{k \in L[j-1]}^{2} Z^b(k,j-1)\ Z_{ts}(k,j-1)\ Z_5(k-2)}^{\text{terminal stack}} \\ & + \overbrace{Z^b(1,j) + Z^b(1,j-1)Z_{3'D}(1,j-1) + Z^b(1,j)Z_{5'D}(1,j) + Z^b(1,j-1)Z_{ts}(1,j-1)}^{k=1} \\ & + \overbrace{Z^b(0,j) + Z^b(0,j-1)Z_{3'D}(0,j-1)}^{k=0} \end{aligned}$$

B.2 Forward Fill

B.2.1 Notes

- All arrays are 0-indexed.
- i and j range so $0 \leq i \leq N-5$ and $i+4 \leq j \leq N-1$.
- $R[i]$ is a list of indices of potential pairs to the right of i starting at position $min(i+4, N)$ and working up to position $i+N-4$.
- $L[i]$ is a list of indices of potential pairs to the left of i starting at position $i-4$ and working down to position 0

B.2.2 Z^b

Base Case: $Z^b(i,j) = 0$ when $j - i \leq 3$

$$
\begin{aligned}
Z^b(i,j) = & \overbrace{e^{-\beta E_h(i,j)}}^{\text{close hairpin}} \\
& + \overbrace{e^{-\beta E_s(i,j)}\ Z^b(i+1, j-1)}^{\text{stack}} \\
& + \overbrace{\sum_{\substack{i<d<e<j \\ 2<j-e+d-i<L}} e^{-\beta E_I(i,d,e,j)}\ Z^b(d,e)^1}^{\text{close internal loop}} \\
& + \overbrace{e^{-\beta(a+c)}\ Z^2(i+1, j-1)}^{\text{close multibranch loop, no dangle}} \\
& + \overbrace{e^{-\beta(a+b+c)}\ Z_{3'D}(j,i)\ Z^2(i+2, j-1)}^{\text{close multibranch loop, 3' dangle}} \\
& + \overbrace{e^{-\beta(a+b+c)}\ Z_{5'D}(j,i)\ Z^2(i+1, j-2)}^{\text{close multibranch loop, 5' dangle}} \\
& + \overbrace{e^{-\beta(a+2b+c)}\ Z_{ts}(j,i)\ Z^2(i+2, j-2)}^{\text{close multibranch loop, terminal stack}}
\end{aligned}
$$

[1]Alternatively, we can think of the loop sum as the nested sum

$$
\sum_{d=i+1}^{min(j-2, i+1+L)} \sum_{\substack{e \in R[d] \\ (d-i-1)+(j-e-1) \leq L}}^{j-1} e^{-\beta E_I(i,d,e,j)}\ Z^b(d,e)
$$

.

B.2.3 Z^1

Base Case: $Z^1(i,j) = 0$ when $j - i \leq 3$

$$
\begin{aligned}
Z^1(i,j) = {} & \overbrace{e^{-\beta b}\ Z^1(i+1,j)}^{i \text{ unpaired}} \\
& + \overbrace{\sum_{k \in R[i]}^{j} e^{-\beta c}\ Z^b(i,k)\ e^{-\beta b(j-k)}}^{(i,k) \text{ pair, first stem, no stacking}} \\
& + \overbrace{\sum_{k \in R[i]}^{j-1} e^{-\beta c}\ Z^b(i,k)\ Z_{3'D}(i,k)\ e^{-\beta b(j-k)}}^{(i,k) \text{ pair, first stem, 3' stacking}} \\
& + \overbrace{\sum_{k \in R[i+1]}^{j} e^{-\beta c}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ e^{-\beta b(j-k+1)}}^{(i+1,k) \text{ pair, first stem, 5' stacking}} \\
& + \overbrace{\sum_{k \in R[i+1]}^{j-1} e^{-\beta c}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ e^{-\beta b(j-k+1)}}^{(i+1,k) \text{ pair, first stem, terminal stacking}} \\
& + \overbrace{\sum_{k \in R[i]}^{j-5} e^{-\beta c}\ Z^b(i,k)\ Z^1(k+1,j)}^{(i,k) \text{ pair, add stem, no stacking}} \\
& + \overbrace{\sum_{k \in R[i]}^{j-6} e^{-\beta(b+c)}\ Z^b(i,k)\ Z_{3'D}(i,k)\ Z^1(k+2,j)}^{(i,k) \text{ pair, add stem, 3' stacking}} \\
& + \overbrace{\sum_{k \in R[i+1]}^{j-5} e^{-\beta(b+c)}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ Z^1(k+1,j)}^{(i+1,k) \text{ pair, add stem, 5' stacking}} \\
& + \overbrace{\sum_{k \in R[i+1]}^{j-6} e^{-\beta(2b+c)}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ Z^1(k+2,j)}^{(i+1,k) \text{ pair, add stem, terminal stacking}}
\end{aligned}
$$

B.2.4 Z^2

Base Case: $Z^2(i,j) = 0$ when $j - i \leq 3$

$$Z^2(i,j) = \overbrace{e^{-\beta b}\ Z^2(i+1,j)}^{i \text{ unpaired}}$$
$$+ \overbrace{\sum_{k \in R[i]}^{j-5} e^{-\beta c}\ Z^b(i,k)\ Z^1(k+1,j)}^{(i,k) \text{ pair, no stacking}}$$
$$+ \overbrace{\sum_{k \in R[i]}^{j-6} e^{-\beta(b+c)}\ Z^b(i,k)\ Z_{3'D}(i,k)\ Z^1(k+2,j)}^{(i,k) \text{ pair, add stem, 3' stacking}}$$
$$+ \overbrace{\sum_{k \in R[i+1]}^{j-5} e^{-\beta(b+c)}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ Z^1(k+1,j)}^{(i+1,k) \text{ pair, add stem, 5' stacking}}$$
$$+ \overbrace{\sum_{k \in R[i+1]}^{j-6} e^{-\beta(2b+c)}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ Z^1(k+2,j)}^{(i+1,k) \text{ pair, add stem, terminal stacking}}$$

B.3 Wraparound

B.3.1 Notes

- All arrays are 0-indexed.
- i and j are allowed to range so $(j - N) \leq i < N$ and $N \leq j < (2N - 1)$
- $R[i]$ is a list of indices of potential pairs to the right of i starting at position $min(i + 4, N)$ and working up to position $i + N - 4$.
- $L[i]$ is a list of indices of potential pairs to the left of i starting at position $i - 4$ and working down to position 0

B.3.2 Z^b

Base Cases:

$$
\begin{aligned}
Z^b(i,j) &= 0 \text{ when } i-(j-N-1)<3 \\
Z^b(N-1,N) &= 1 \\
Z^b(N-1,N+1) &= 1+Z_{5'D}(1,N-1) \\
Z^b(N-1,j) &= Z_5(j-N-1)+Z_5(j-N-2)Z_{5'D}(j-N,N-1) \\
Z^b(N-2,N) &= 1+Z_{3'D}(0,N-2) \\
Z^b(i,N) &= Z_3(i+1)+Z_3(i+2)Z_{3'D}(0,i) \\
Z^b(N-2,j) &= e^{-\beta E_s(i,j)}\ Z^b(i+1,j-1) \\
&\quad + \sum_{\substack{i<d<e<j \\ 2<j-e+d-i<L \\ d\le N<e}} e^{-\beta E_I}(i,d,e,j)\ Z^b(d,e) \\
&\quad + e^{-\beta(a+c)}\ Z^2(i+1,j-1) \\
&\quad + e^{-\beta(a+b+c)}\ Z_{3'D}(j,i)\ Z^2(i+2,j-1) \\
&\quad + e^{-\beta(a+b+c)}\ Z_{5'D}(j,i)\ Z^2(i+1,j-2) \\
&\quad + e^{-\beta(a+2b+c)}\ Z_{ts}(j,i)\ Z^2(i+2,j-2) \\
&\quad + Z_5(j-N-1) \\
&\quad + Z_5(j-N-1)\ Z_{3'D}(j-N,i) \\
&\quad + Z_5(j-N-2)\ Z_{5'D}(j-N,i) \\
&\quad + Z_5(j-N-2)\ Z_{ts}(j-N,i) \\
Z^b(i,1) &= e^{-\beta E_s(i,j)}\ Z^b(i+1,j-1) \\
&\quad + \sum_{\substack{i<d<e<j \\ 2<j-e+d-i<L \\ d\le N<e}} e^{-\beta E_I}(i,d,e,j)\ Z^b(d,e) \\
&\quad + e^{-\beta(a+c)}\ Z^2(i+1,j-1) \\
&\quad + e^{-\beta(a+b+c)}\ Z_{3'D}(j,i)\ Z^2(i+2,j-1) \\
&\quad + e^{-\beta(a+b+c)}\ Z_{5'D}(j,i)\ Z^2(i+1,j-2) \\
&\quad + e^{-\beta(a+2b+c)}\ Z_{ts}(j,i)\ Z^2(i+2,j-2) \\
&\quad + Z_3(i+1) \\
&\quad + Z_3(i+2)\ Z_{3'D}(j-N,i) \\
&\quad + Z_3(i+1)\ Z_{5'D}(j-N,i) \\
&\quad + Z_3(i+2)\ Z_{ts}(j-N,i)
\end{aligned}
$$

$$
\begin{aligned}
Z^b(i,j) = {} & \overbrace{e^{-\beta E_s(i,j)}\ Z^b(i+1,j-1)}^{\text{stack}} \\
& + \overbrace{\sum_{\substack{i<d<e<j \\ 2<j-e+d-i<L \\ d\le N<e}} e^{-\beta E_I}(i,d,e,j)\ Z^b(d,e)^2}^{\text{close internal loop}} \\
& + \overbrace{e^{-\beta(a+c)}\ Z^2(i+1,j-1)}^{\text{close multibranch loop, no dangle}} \\
& + \overbrace{e^{-\beta(a+b+c)}\ Z_{3'D}(j,i)\ Z^2(i+2,j-1)}^{\text{close multibranch loop, 3' (right) dangle}} \\
& + \overbrace{e^{-\beta(a+b+c)}\ Z_{5'D}(j,i)\ Z^2(i+1,j-2)}^{\text{close multibranch loop, 5' (left) dangle}} \\
& + \overbrace{e^{-\beta(a+2b+c)}\ Z_{ts}(j,i)\ Z^2(i+2,j-2)}^{\text{close multibranch loop, terminal stack}} \\
& + \overbrace{Z_3(i+1)\ Z_5(j-N-1)}^{\text{external loop, no dangle}} \\
& + \overbrace{Z_3(i+2)\ Z_5(j-N-1)\ Z_{3'D}(j-N,i)}^{\text{external loop, 3' dangle}} \\
& + \overbrace{Z_3(i+1)\ Z_5(j-N-2)\ Z_{5'D}(j-N,i)}^{\text{external loop, 5' dangle}} \\
& + \overbrace{Z_3(i+2)\ Z_5(j-N-2)\ Z_{ts}(j-N,i)}^{\text{external loop, terminal stack}}
\end{aligned}
$$

[1]Alternatively, we can think of the loop sum as the nested sum

$$
\sum_{d=i+1}^{min(N-1,i+1+L)} \quad \sum_{\substack{e\in R[d] \\ max((j-i-1)+(d-1)-L,N)}}^{j-1} e^{-\beta E_I(i,d,e,j)}\ Z^b(d,e)
$$

.

B.3.3 Z^1

Base Case: $Z^1(N, j) = 0$

$$
\begin{aligned}
Z^1(i,j) = {} & \overbrace{e^{-\beta b}\ Z^1(i+1,j)}^{i \text{ unpaired}} \\
& + \overbrace{\sum_{\substack{k \in R[i] \\ k \geq N}}^{j} e^{-\beta c}\ Z^b(i,k)\ e^{-\beta b(j-k)}}^{(i,k) \text{ pair, first stem, no stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i] \\ k \geq N}}^{j-1} e^{-\beta c}\ Z^b(i,k)\ Z_{3'D}(i,k)\ e^{-\beta b(j-k)}}^{(i,k) \text{ pair, first stem, 3' (right) stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i+1] \\ k \geq N}}^{j} e^{-\beta c}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ e^{-\beta b(j-k+1)}}^{(i+1,k) \text{ pair, first stem, 5' (left) stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i+1] \\ k \geq N}}^{j-1} e^{-\beta c}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ e^{-\beta b(j-k+1)}}^{(i+1,k) \text{ pair, first stem, terminal stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i] \\ k \neq N-1}}^{j-5} e^{-\beta c}\ Z^b(i,k)\ Z^1(k+1,j)}^{(i,k) \text{ pair, add stem, no stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i] \\ k \neq N-1 \\ k \neq N-2}}^{j-6} e^{-\beta(b+c)}\ Z^b(i,k)\ Z_{3'D}(i,k)\ Z^1(k+2,j)}^{(i,k) \text{ pair, add stem, 3' (right) stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i+1] \\ k \neq N-1}}^{j-5} e^{-\beta(b+c)}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ Z^1(k+1,j)}^{(i+1,k) \text{ pair, add stem, 5' (left) stacking}} \\
& + \overbrace{\sum_{\substack{k \in R[i+1] \\ k \neq N-1 \\ k \neq N-2}}^{j-6} e^{-\beta(2b+c)}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ Z^1(k+2,j)}^{(i+1,k) \text{ pair, add stem, terminal stacking}}
\end{aligned}
$$

B.3.4 Z^2

Base Case: $Z^2(N, j) = 0$

$$Z^2(i,j) = \overbrace{e^{-\beta b}\ Z^2(i+1,j)}^{i \text{ unpaired}}$$
$$+ \overbrace{\sum_{\substack{k \in R[i] \\ k \neq N-1}}^{j-5} e^{-\beta c}\ Z^b(i,k)\ Z^1(k+1,j)}^{(i,k) \text{ pair, no stacking}}$$
$$+ \overbrace{\sum_{\substack{k \in R[i+1] \\ k \neq N-1}}^{j-6} e^{-\beta(b+c)}\ Z^b(i,k)\ Z_{3'D}(i,k)\ Z^1(k+2,j)}^{(i,k) \text{ pair, add stem, 3' (right) stacking}}$$
$$+ \overbrace{\sum_{\substack{k \in R[i] \\ k \neq N-1}}^{j-5} e^{-\beta(b+c)}\ Z^b(i+1,k)\ Z_{5'D}(i+1,k)\ Z^1(k+1,j)}^{(i+1,k) \text{ pair, add stem, 5' (left) stacking}}$$
$$+ \overbrace{\sum_{\substack{k \in R[i+1] \\ k \neq N-1}}^{j-6} e^{-\beta(2b+c)}\ Z^b(i+1,k)\ Z_{ts}(i+1,k)\ Z^1(k+2,j)}^{(i+1,k) \text{ pair, add stem, terminal stacking}}$$