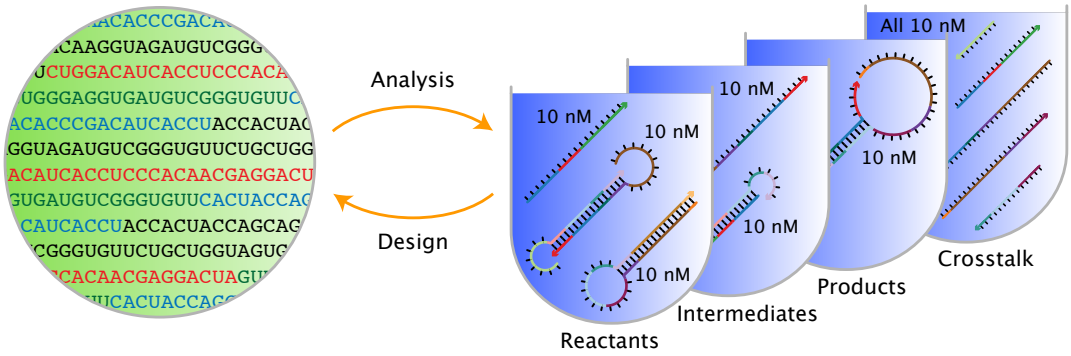
NUPACK 3.2 User Guide

Analysis and Design of Nucleic Acid Structures, Devices, and Systems

Nicholas J. Porubsky, Brian R. Wolfe, Justin S. Bois, and Niles A. Pierce

California Institute of Technology

March 7, 2017



Contents

List of Figures

List of Tables

List of Examples

1 Overview

NUPACK is a growing software suite for the analysis and design of nucleic acid structures, devices, and systems serving the needs of researchers in the fields of nucleic acid nanotechnology, molecular programming, synthetic biology, and across the life sciences more broadly. Most of this software may be conveniently run using the NUPACK web application at nupack.org (Zadeh et al., 2011a).

When finishing a project that has benefited from NUPACK calculations, please remember to cite the NUPACK web application and algorithms appropriately; citations are an important component in helping to secure funding for NUPACK development and maintenance. Please email us with questions, comments, feature requests, and bug reports at support@nupack.org.

– The NUPACK Team

1.1 Terminology, notation, and physical model

NUPACK algorithms are formulated in terms of nucleic acid secondary structure. In most cases, pseudoknots are excluded from the structural ensemble. The sequence, ϕ, of one or more interacting RNA strands is specified as a list of bases $\phi^a \in \{\texttt{A},\texttt{C},\texttt{G},\texttt{U}\}$ for $a = 1,\ldots,|\phi|$ (`T` replaces `U` for DNA). A *secondary structure*, s, of one or more interacting RNA strands is defined by a set of base pairs (each a Watson–Crick pair [`A`·`U` or `C`·`G`] or a wobble pair [`G`·`U`]). A *polymer graph* representation of a secondary structure is constructed by ordering the strands around a circle, drawing the backbones in succession from 5′ to 3′ around the circumference with a *nick* between each strand, and drawing straight lines connecting paired bases. A secondary structure is *unpseudoknotted* if there exists a strand ordering for which the polymer graph has no crossing lines. A secondary structure is *connected* if no subset of the strands is free of the others. A *complex* of L interacting strands with strand ordering, π, has *structural ensemble* containing all connected polymer graphs with no crossing lines (Dirks et al., 2007). (We dispense with our prior convention (Dirks et al., 2007; Zadeh et al., 2011a,b) of calling this entity an *ordered complex*.)

If a complex contains multiple strands with the same sequence, subtleties arise in the definition of the structural ensemble and in the calculation of experimental observables (Dirks et al., 2007). Let Γ denote the structural ensemble in which each strand is treated as distinct (i.e., each strand has a unique identifier in $\{1,\ldots,L\}$) and let Γ' denote the ensemble in which strands with the same sequence are treated as indistinguishable. Two secondary structures are indistinguishable if their polymer graphs can be rotated so that all strands are mapped onto indistinguishable strands, all base pairs are mapped onto base pairs, and all unpaired bases are mapped onto unpaired bases; otherwise the structures are distinct (Dirks et al., 2007). The ensemble $\Gamma' \subseteq \Gamma$ is a maximal subset of distinct secondary structures for strand ordering π.

A *test tube* may contain an arbitrary number of strand species interacting to form an arbitrary number of complex species in a dilute solution. Let Ψ^0 denote the set of strand species that interact in a test tube to form the set of complex species Ψ. It is often convenient to define Ψ to contain all complexes of up to some size $L_{\max}$. Each complex $j \in \Psi$ corresponds to a distinct strand ordering π_j of L strands for $L \in \{1,\ldots,L_{\max}\}$. L distinct strands can be ordered around a circle in $(L-1)!$ distinct ways (e.g., strands A, B, and C can be ordered ABC and ACB). If some of the L strands are of the same species, there will be fewer than $(L-1)!$ distinct strand orderings (e.g., strands A, A, and B can only be ordered AAB). For a given set of L strands, each unpseudoknotted connected secondary structure is found in the structural ensemble, Γ_j, corresponding to exactly one strand ordering, π_j (i.e., exactly one complex $j \in \Psi$) (Dirks et al., 2007).

For sequence ϕ and secondary structure, s, the *free energy*, $\Delta G(\phi, s)$, is calculated using nearest-neighbor empirical parameters for RNA (Serra and Turner, 1995; Mathews et al., 1999; Zuker, 2003) in 1M Na^+ or for DNA (SantaLucia, 1998; Zuker, 2003) in user-specified concentrations of Na^+ and Mg^{++} (SantaLucia and Hicks, 2004; Koehler and Peyret, 2005). Additional parameters are employed for pseudoknotted secondary structures (Dirks and Pierce,

2003), which may be included in the structural ensemble only when analyzing a single RNA strand. The zero free energy reference state for all calculations is a system where all relevant strands are present with no base pairs (Dirks et al., 2007).

1.2 Conventions

- Sequences are listed $5'$ to $3'$. The bases in a complex are indexed starting with 1 at the $5'$-most base of the first strand and ending at the $3'$-most base of the last strand. For example, if a complex has three strands of length 15, 20, and 13, respectively, the fifth base of the third strand has index 40.
- Valid bases are `A`, `C`, `G`, `T`, and `U`. For RNA calculations, `T` is automatically converted to `U`, and vice versa for DNA calculations.
- Secondary structures are specified in one of three ways:
  - *dot-parens-plus notation*: each unpaired base is represented by a dot, each base pair by matching parentheses, and each nick between strands by a plus (Zadeh et al., 2011a). For example, `((...))` specifies that bases 1 and 2 are paired to bases 7 and 6, respectively, while bases 3, 4, and 5 are unpaired. `((+...))` specifies that bases 1 and 2 of strand 1 are paired to bases 5 and 4 of strand 2. Four types of "parentheses" are accepted: `()`, `[]`, {}, and <>. Within a specified structure, each type of parentheses must satisfy a nesting property but different types need not be nested, allowing specification of pseudoknotted structures (though highly nested pseudoknots may not be specifiable with only four types of parentheses).
  - *DU+ notation*: Using DU+ notation, a duplex of length $\underline{x}$ base pairs is represented by D$\underline{x}$ and an unpaired region of length $\underline{x}$ nucleotides is represented by U$\underline{x}$ (Zadeh, 2010). Each duplex is followed immediately by the substructure (specified in DU+ notation) that is 'enclosed' by the duplex. If this substructure includes more than one element, parentheses are used to denote scope. A nick between strands is specified by a '+'. See Figure 1 and Table 1 for examples.
  - *pair list notation*: each line consists of two whitespace-separated integers [i j], $i < j$, specifying that base i is paired to base j. Any secondary structure, including highly-nested pseudoknots, may be specified in this way.

Table 1: Examples of dot-parens-plus and DU+ notation.

Dot-parens-plus notation	DU+ notation
`((((((((((((.........))))))))))))`	`D12 U10`
`((((((((((((+))))))))))))........`	`D12 + U10`
`((((((((((((+.........))))))))))))`	`D12 (+ U10)`

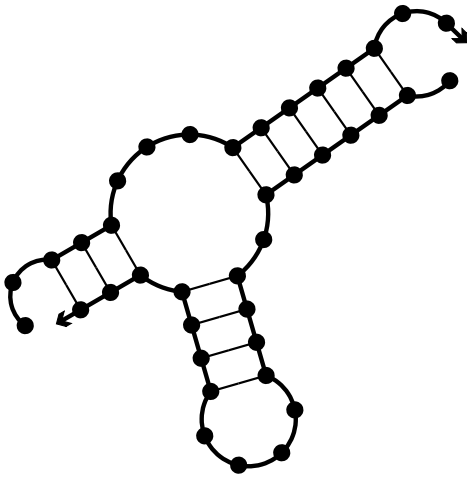
The following option flags are recognized by multiple NUPACK executables:

`-material` `parameters`
The parameter files defining the nucleic acid material are specified via the argument `parameters` which represents either a filename prefix or a shorthand identifier for an included parameter set. If the filename does not contain a relative or absolute path, then the program will look for the files first in the current directory, and then in the directory `$NUPACKHOME/parameters`. Available filename prefixes currently include:

- `rna1995` (default; shorthand: `rna`)
  Parameter files `*.dG` and `*.dH` for RNA allowing calculations at different temperatures (Serra and Turner, 1995; Zuker, 2003); includes pseudoknot parameters from (Dirks and Pierce, 2003).

Figure 1: Comparison of dot-parens-plus and DU+ notation.

Secondary structure:



Dot-parens-plus notation: `..(((...(((((((..+.)))))))..((((....))))))))`

DU+ notation: `U2 D3 (U3 D6 (U2 + U1) U1 D4 (U4))`

- `dna1998` (shorthand: `dna`)
  Parameter files `*.dG` and `*.dH` for DNA allowing calculations at different temperatures (SantaLucia, 1998; Zuker, 2003); there are no pseudoknot parameters.
- `rna1999`
  Parameter file `*.dG` for RNA for calculations at 37 °C (Mathews et al., 1999; Zuker, 2003); includes pseudoknot parameters from (Dirks and Pierce, 2003).
- `custom`
  Custom parameter files `*.dG` and `*.dH` allowing calculations at different temperatures; or custom parameter file `*.dG` allowing calculations at one temperature. Custom parameter files must be placed in the same location as the default parameter files (`/urs/local/share` for a default installation).

DNA/RNA hybrids are not allowed.

`-sodium concentration`
The Na^+ concentration of the solution in units of molar (default: 1.0, range: [0.05,1.1]) is specified by `concentration` (SantaLucia and Hicks, 2004). This flag is only valid when the `-material dna` is also selected because no RNA salt correction parameters are available.

`-magnesium concentration`
The Mg^{++} concentration of the solution in units of molar (default: 0.0, range: [0.0,0.2]) is specified by `concentration` (Koehler and Peyret, 2005). This flag is only valid when the `-material dna` is also selected.

`-dangles treatment`
The way in which dangle energies are incorporated is specified by `treatment`, which may have the following values:

- `none`: No dangle energies are incorporated.
- `some`: (default) A dangle energy is incorporated for each unpaired base flanking a duplex (a base flanking two duplexes contributes only the minimum of the two possible dangle energies).
- `all`: A dangle energy is incorporated for each base flanking a duplex regardless of whether it is paired.

`-T temperature`
Temperature specified in °C (default: 37).

`-multi`
Specify a calculation involving complexes of multiple interacting strands.

`-pseudo`
Augment the structural ensemble Γ with a class of pseudoknots (Dirks and Pierce, 2003, 2004). This option is only available for single-stranded RNA calculations. An error message is returned if `-pseudo` is specified in combination with either `-multi` or `-material dna`.

1.3 Versions

- NUPACK 3.0:
  - Features:
    - complex analysis
    - complex design
    - test tube analysis
  - Executables:
    - `pfunc`, `pairs`, `mfe`, `subopt`, `count`, `energy`, `prob`, `pairs`, `defect`, `complexes`, `concentrations`, `distributions`, `design`
    - These executables read input files containing comment lines preceded by `%`; blank lines are not permitted.
  - Terminology and notation:
    - details in (Dirks et al., 2007)
- NUPACK 3.1:
  - New features:
    - test tube design
  - New executables:
    - `tubedesign` and `tubedefect`
    - These executables read `*.np` script files written in v1 of the NUPACK scripting language (see "Future Version: Script files" below).
    - In `*.np` script files, a comment begins with `#` and continues for the rest of the line; blank lines are permitted.
  - Changes to existing executables:
    - Name of executable `design` changed to `complexdesign`.
    - Name of executable `defect` changed to `complexdefect`.
    - Updates to the default options and output file formats for executables `complexes`, `concentrations`, and `distributions`. Use option `-v3.0` to revert to NUPACK 3.0 behavior using NUPACK 3.1.
  - Terminology and notation:
    - details in Section 1.1
- NUPACK 3.2:
  - New features:

    - constrained multistate test tube design
  - New executables:
    - `multitubedesign` and `multitubedefect`
    - These executables read `*.np` script files written in v2 of the NUPACK scripting language.
    - In `*.np` script files, a comment begins with `#` and continues for the rest of the line; blank lines are permitted.
  - Terminology and notation:
    - details in Section 1.1

- Future Version:
  - Script files:
    - In a future major release, all NUPACK jobs will be specified using a NUPACK scripting language in `*.np` script files. This approach will be more flexible and robust than the current approach of using a tailored input file format for each NUPACK executable.
  - Output files:
    - In a future major release, the format of NUPACK output files will be changed so that they can be loaded using a generic parser. This approach will be more convenient and robust than the current approach of using a tailored output file format for each NUPACK executable.

1.4 License

2 Analysis

Analyze the equilibrium properties over one of two ensembles:

- Complex Analysis: analyze the equilibrium base-pairing properties of a complex of interacting nucleic acid strands (Dirks et al., 2007).
- Test Tube Analysis: analyze the equilibrium base-pairing properties and concentrations for a test tube of interacting nucleic acid strands (Dirks et al., 2007).

2.1 Complex Analysis

Analyze the equilibrium base-pairing properties of a complex of interacting nucleic acid strands.

2.1.1 `pfunc`: calculate the partition function

Command: `pfunc [-T temperature] [-multi] [-pseudo] [-material parameters] [-dangles treatment] prefix`

Description: Computes the partition function, $Q(\phi)$, over the ensemble Γ'.

Input: Input is read from the file `prefix.in`, where `prefix` is a command line argument. Comment lines are preceded by `%` and may be interspersed with data in input files; blank lines are not permitted. For single-stranded calculations, the input file contains the strand sequence specified on a single line. If `-multi` is specified, the input file must contain the following entries on separate lines:

- The number of distinct strand species, $|\Psi^0|$.
- The sequence for each distinct strand species (each on a separate line). Note that strand species defined on different lines are treated as distinct even if they have the same sequence.
- L integers from the range 1 to $|\Psi^0|$ representing the strand ordering π of the L strands in the complex.

Output:
Following header comments, the free energy of the complex ($\Delta G \equiv -kT \log Q$) and the partition function are written to the screen.

Example 1: Partition function for a single strand including pseudoknots. Calculate the partition function for a single RNA strand at 37 °C including a class of pseudoknots.

Input file contents:

```
GGGCUGUUUUUCUCGCUGACUUUCAGCCCCAAACAAAAAAUGUCAGCA
```

Command: `pfunc -pseudo`
`$NUPACKHOME/doc/examples/complex-analysis/pseudoknot/input/telomerase`

Example 2: Partition function for a complex. Calculate the partition function for a complex of four DNA strands at 23 °C, two of which are indistinguishable.

Input file contents:

```
3
AGTCTAGGATTCGGCGTGGGTTAA
TTAACCCACGCCGAATCCTAGACTCAAAGTAGTCTAGGATTCGGCGTG
AGTCTAGGATTCGGCGTGGGTTAACACGCCGAATCCTAGACTACTTTG
1 2 2 3
```

Command: `pfunc -T 23 -multi -material dna`
`$NUPACKHOME/doc/examples/complex-analysis/advanced/input/hcr`

2.1.2 `pairs`: calculate base-pairing observables

Command: `pairs [-T temperature] [-multi] [-pseudo] [-material parameters] [-dangles treatment] [-cutoff cutoffvalue] prefix`

Description: Computes *pair probabilities* $p(i_n \cdot j_m)$ for the complex corresponding to the specified strand ordering π. When `-multi` is selected, also computes the *expected number of base pairs* $E(i_{\{A\}} \cdot j_{\{B\}})$.

Additional option:

`-cutoff cutoffvalue`
Only probabilities and expected values at or above `cutoffvalue` (default: 0.001) are saved in the output file(s).

Input: Same format as for the executable `pfunc`.

Output: The output is written to the files:

- `prefix.ppairs`
  Contains the probability of each type of base pair in the complex. The relevant quantities are $p(i_n \cdot j_m)$, the probability that base i of strand n is paired to base j of strand m in the complex corresponding to strand ordering π. All strands in the complex are considered to be distinct. For example, the two strands labeled 2 in Example 2 are considered distinct. One might think of them as strand $2a$ and $2b$, and a given base of strand $2a$ may have different pair probabilities than the corresponding one in strand $2b$. The total number of bases in the complex is $N = \sum_{l=1}^{L} N_l$, so indexing bases from 1 to N, the pair probabilities can be stored in a symmetric $N \times N$ matrix. Augmentation by an $N+1$st column containing the probability that each base is unpaired causes the rows to sum to unity.

  By default, the file is formatted as follows. Following header comments, the first entry is the integer N. The remaining entries come in triplets of the form [i j p], where $1 \leq i \leq N$ and $1 \leq j \leq N+1$ are base numbers and p is the probability of the corresponding pair. Values corresponding to $j = N+1$ represent the probability that base i is unpaired. If `-pseudo` is selected, each row is augmented by two additional columns. The first is the probability that bases i and j form a nested pair and the second is the probability that bases i and j form a non-nested pair. In the case of $j = N+1$, these additional columns store the probability that bases i and j do not form a nested pair and the probability that they do not form a non-nested pair, respectively.

- `prefix.epairs`
  Generated when `-multi` is selected. Similar to `prefix.ppairs` except strands of the same species are considered to be indistinguishable. The relevant quantities are $E(i_{\{A\}} \cdot j_{\{B\}})$, the expected number of base i of strand species A that are paired to base j of strand species B in the complex corresponding to strand ordering

π. The number of distinct bases in the complex is $N_{\text{distinct}} \equiv \sum_{k \in \Psi^0} N_k$, representing the total number of bases in all $|\Psi^0|$ strand species. Numbering the distinct bases from 1 to N_{distinct}, the distinct base pairs may be represented as a symmetric $N_{\text{distinct}} \times N_{\text{distinct}}$ matrix; by augmenting the matrix with an extra column that contains the expected number of base i of strand species A that are unpaired, each row sums to the number of base i of strand species A in the complex. Note that this numbering system is used even if some sequences listed in the input file are absent from the specified complex.

The file is formatted as follows. Following header comments, the first entry is the integer N_{distinct}, and the remaining entries come in triplets of the form $[i \;\; j \;\; E]$, analogously to the `.ppairs` file, except E is the expected number of the corresponding pair. Information is stored only for bases included in the specified complex.

2.1.3 `mfe`: find the minimum free energy (MFE) secondary structure(s)

Command: `mfe [-T temperature] [-multi] [-pseudo] [-material parameters] [-dangles treatment] [-degenerate] prefix`

Description: Determine the minimum free energy secondary structure(s), $s^{\text{MFE}}(\phi)$, of sequence ϕ over the ensemble of the complex, Γ. If the `-degenerate` flag is selected, all secondary structures with the minimum free energy are determined; otherwise only one MFE structure is returned.

Input: Same format as for the executable `pfunc`.

Output: Output is written to the file `prefix.mfe`. After header comments, each entry describes one of the possibly many degenerate MFE structures. The entries are separated by comment lines (repeated `%` signs). The first line in each entry is the number of bases in the ordered complex. The second line is the minimum free energy. The third line is the dot-parens-plus representation of the MFE structure. Subsequent lines contain the MFE structure in pair list notation.

2.1.4 `subopt`: find all secondary structures within a specified free energy gap of the MFE

Command: `subopt [-T temperature] [-multi] [-pseudo] [-material parameters] [-dangles treatment] prefix`

Description: Similar to `mfe` except that all secondary structures in Γ with free energies within the specified (non-negative) free energy gap of the MFE are calculated and stored. This can be very slow and the output very large if the specified gap is too large. The output is sorted by increasing free energy.

Input: Same format as for the executable `pfunc`, plus one additional row containing the energy gap.

Output: Output is written to the file `prefix.subopt` with the same format as for the executable `mfe`.

2.1.5 `count`: count the number of secondary structures in the ensemble

Command: `count [-multi] [-pseudo] prefix`

Description: Calculates the number of secondary structures, $|\Gamma|$, in the ensemble of the complex, treating all strands as distinct.

Input: Same format as for the executable `pfunc`.

Output: The number of secondary structures is written to the screen, preceded by header comments.

2.1.6 `energy`: calculate the free energy of a secondary structure

Command: `energy [-T temperature] [-pseudo] [-multi] [-material parameters] [-dangles treatment] prefix`

Description: Calculate the free energy, $\Delta G(\phi, s)$, of sequence ϕ in secondary structure s. See Figure 2 for a cautionary tale.

Input: Same format as for executable `pfunc`, plus one additional row specifying the secondary structure in dot-parens-plus notation. Alternatively, the structure may be represented in pair list notation.

Output: The free energy is written to the screen, preceded by header comments.

2.1.7 `prob`: calculate the equilibrium probability of a secondary structure

Command: `prob [-T temperature] [-pseudo] [-multi] [-material parameters] [-dangles treatment] prefix`

Description: Calculates the equilibrium probability, $p(\phi, s)$, that sequence ϕ adopts secondary structure s within the ensemble of the complex, Γ'.

Input: Same format as for the executable `energy`.

Output: The probability is written to the screen, preceded by header comments.

2.2 Test Tube Analysis

Analyze the equilibrium base-pairing properties and concentrations for a test tube of interacting nucleic acid strands.

2.2.1 `complexes`: calculate the partition function and equilibrium base-pairing properties for each complex in a test tube

Command: `complexes [-T temperature] [-material parameters] [-pairs] [-mfe] [-degenerate] [-dangles treatment] [-timeonly] [-v3.0] [-quiet] prefix`

Description: For the set of strands Ψ^0, calculate the partition function, Q_j, for each complex $j \in \Psi$, corresponding to all complexes of up to $L_{\max}$ strands. Significant additional functionality can be specified via command line flags. The output of `complexes` can be used as the input to the executables `concentrations` and `distributions`.

Additional options:

`-pairs`
Calculate base-pairing observables as for the `pairs` executable.

`-cutoff cutoffvalue`
Only probabilities and expected values at or above `cutoffvalue` (default: 0.001) are saved in the output file(s) generated when the `-pairs` flag is selected.

`-mfe`
Calculate all MFE structures for each complex as for the `mfe` executable. The `-degenerate` flag is only applicable in conjunction with the `-mfe` flag.

`-timeonly`
After generating the strand orderings for all complexes in Ψ, estimate the time it would take to compute all of the partition functions. The partition function calculations are not performed, the time estimate is written to the screen, and no output files are generated.

`-v3.0`
Revert to NUPACK 3.0 behavior (see changes).

`-quiet`
Suppress output to the screen.

Input: Input is read from the file `prefix.in`, where `prefix` is a command line argument. Comment lines are preceded by `%` and may be interspersed with data in input files; blank lines are not permitted. The input file must contain the following entries on separate lines:

- The number of distinct strand species, $|\Psi^0|$.
- The sequence for each distinct strand species (each on a separate line).
- The maximum complex size, $L_{\max}$.

In addition to considering all complexes up to size $L_{\max}$, the optional file `prefix.list` can be used to manually specify complexes with more than $L_{\max}$ strands. Each ordered complex of size $L > L_{\max}$ is specified on a separate line by:

- A list of L integers from the range 1 to $|\Psi^0|$ representing the strand ordering for the complex.

Output: Unless the `-quiet` flag is selected, `complexes` reports progress to the screen. By default there are two output files:

- `prefix.ocx`
  Contains the strand composition and free energy of each complex $j \in \Psi$. The first and second columns are integer strand composition and strand ordering identifiers, respectively, the next $|\Psi^0|$ columns are $A_{1,j}\ \ A_{2,j}\ \ \ldots\ \ A_{|\Psi^0|,j}$ defining the number of each strand type in complex j, and the final column is ΔG_j for complex j.

- `prefix.ocx-key`
  Contains the strand ordering π_j for each complex $j \in \Psi$ containing L_j strands. The first and second columns are integer strand composition and strand ordering identifiers, respectively, and the remaining L_j columns are integers from the range 1 to $|\Psi^0|$. Note that the value of L_j may be different for each complex $j \in \Psi$.

Depending on the command line options, the following output files may also be written:

- `prefix.ocx-epairs`
  Generated if `-pairs` is selected. Contains the base-pairing expectation values for each type of distinct base pair in each complex. The relevant quantities are $E(i_{\{A\}} \cdot j_{\{B\}})$, the expected number of base i of strand species A that are paired to base j of strand species B in the complex. The entries are separated by comment lines (repeated `%` symbols), and each entry begins with a comment line containing the strand composition identifier `idcomp` and strand ordering identifier `idorder` expressed as "`% compositionidcomp orderingidorder`".

- `prefix.ocx-ppairs`
  Generated if `-pairs` is selected. Similar to `.epairs` except that all strands in the complex are assumed to be distinct. The data in each entry are the same as those in the `.ppairs` file produced by the executable `pairs`.

- prefix.ocx-mfe
  Generated if -mfe is selected. Contains the minimum free energy and MFE structure(s) for each complex. Each entry is formatted the same as the output for the mfe executable. The entries are separated by comment lines (repeated % symbols), and each entry begins with a comment line containing the strand composition identifier idcomp and strand ordering identifier idorder expressed as "% compositionidcomp orderingidorder". If the -degenerate flag is selected, the degenerate MFE structures for a given entry are separated by a comment line of repeated % symbols.

Changes: Relative to NUPACK 3.0, the following changes were introduced to the complexes executable:

- -ordered is on by default
- output files .cx and .cx-epairs are not written
- the comment lines in .ocx-epairs and .ocx-mfe employ updated terminology

Use the -v3.0 option to revert to NUPACK 3.0 behavior.

2.2.2 concentrations: calculate the equilibrium concentration for each complex in a test tube

Command: concentrations [-pairs] [-sort method] [-v3.0] [-quiet] prefix

Description: Given a user-specified concentration for each strand species, calculates the equilibrium concentration of each complex species or base pair in a dilute solution (e.g., a test tube) (Dirks et al., 2007). Partition function information is read in from output files generated with the executable complexes.

Additional options:

-pairs
Compute base-pairing information for the entire solution using results from prefix.ocx-epairs as output by the executable complexes.

-cutoff cutoffvalue
Only ensemble pair fractions at or above cutoffvalue (default: 0.001) are saved in the output file prefix.fpairs generated when the -pairs flag is selected. Note that cutoffvalue should not be less than that used with complexes to generate the input files.

-sort method
The argument method is one of the following integers:

- 0: Output is listed in the same order as in the input file.
- 1: Output is sorted by the concentration of each complex (default).
- 2: Output is sorted first by the sum of the concentrations of all complexes with each strand composition and then by the concentration of each complex with that strand composition.
- 3: Output is sorted first by the strand composition identifier and then by the strand ordering identifier.
- 4: Output is sorted first by the number of strands in each complex, then by the integers $A_{1,j}$ $A_{2,j}$ ... $A_{|\Psi^0|,j}$ defining the number of each strand type in complex j (with $A_{1,j}$ having the highest precedence, followed by $A_{2,j}$, and so on), and finally by the strand ordering identifier.

-v3.0
Revert to NUPACK 3.0 behavior (see changes).

-quiet
Suppress output to the screen.

Input: Input is read from the file `prefix.ocx` output from the executable `complexes`. The temperature at which the calculation is done is read from a line in the comments of the `.ocx` input file that reads "`% T = temperature`", where `temperature` is the temperature in °C. This line is automatically included in all output files of the executable `complexes`.
The input file `prefix.con` specifies the total molar concentration of each of $|\Psi^0|$ strand species on a separate line. The concentration may be in scientific notation (e.g., `1e-6` for a strand species at 1 μM concentration).

Output: Unless `-quiet` is selected, the following information is written to the screen:

- The error in conservation of mass for each strand species in molar.
- The free energy of the entire solution in kcal/L.
- The wall clock time for the calculation.

The output is written to the files:

- `prefix.eq`
  The content is the same as the input file (except resorted, depending on the `-sort` option) with an extra column containing the concentration of the species in molar inserted after the free energy column.
- `prefix.fpairs`
  Generated if `-pairs` is selected. Reports the fraction of each distinct base that is paired to each of the other distinct bases in solution. The relevant quantity is $f_A(i_A \cdot j_B)$, the expected fraction of strands of species A for which base i is paired to base j of strand species B (Dirks et al., 2007). The number of distinct bases in the dilute solution is $N_{\text{distinct}} \equiv \sum_{k=1}^{|\Psi^0|} N_k$, representing the total number of bases in all $|\Psi^0|$ strand species. Numbering the distinct bases from 1 to N_{distinct}, the quantity $f_A(i_A \cdot j_B)$ may be stored as an (asymmetric) $N_{\text{distinct}} \times N_{\text{distinct}}$ matrix; by augmenting the matrix with an extra column that contains the expected fraction of base i of strand species A that are unpaired, each row sums to unity. The file is formatted as follows. Following header comments, the first entry is the integer N_{distinct}. The remaining entries come in triplets of the form $[i \;\; j \;\; f]$, where $1 \leq i \leq N_{\text{distinct}}$ and $1 \leq j \leq N_{\text{distinct}} + 1$ are base numbers and f is the corresponding fraction from the augmented matrix.

Changes: Relative to NUPACK 3.0, the following change was introduced to the `concentrations` executable: the `-ordered` option is on by default. Use the `-v3.0` option to revert to NUPACK 3.0 behavior.

Example 3: Test tube analysis. Calculate the partition function, equilibrium pair probabilities, MFE structure(s), and equilibrium concentration for each complex in a test tube containing three DNA strand species that interact to form all complex species of up to four strands, plus additional larger complexes specified in a `.list` file.

Input file contents:

```
3
AGTCTAGGATTCGGCGTGGGTTAA
TTAACCCACGCCGAATCCTAGACTCAAAGTAGTCTAGGATTCGGCGTG
AGTCTAGGATTCGGCGTGGGTTAACACGCCGAATCCTAGACTACTTTG
4
```

List file contents:

```
1 2 2 3 3
1 2 3 2 3
2 3 2 3 2
1 2 2 2 3 3
```

Commands: `complexes -T 23 -material dna -pairs -mfe -degenerate $NUPACKHOME/doc/examples/tube-analysis/advanced/input/hcr concentrations -pairs $NUPACKHOME/doc/examples/tube-analysis/advanced/input/hcr`

Figure 2: A cautionary tale: free energies and concentrations. NUPACK calculates free energies and equilibrium concentrations of complexes as described in (Dirks et al., 2007) (in particular, if you plan to calculate equilibrium concentrations by hand, see endnote 13 regarding strand association penalties). For example, the following holds at equilibrium for a dilute solution containing strands A and B that can interact to form complex AB:

$$\begin{aligned}\frac{x_{\mathrm{AB}}}{x_{\mathrm{A}}\,x_{\mathrm{B}}} &= \exp\left\{-\frac{\Delta G_{\mathrm{AB}}-\Delta G_{\mathrm{A}}-\Delta G_{\mathrm{B}}}{kT}\right\},\\ &= \frac{[\mathrm{AB}]/\rho_{\mathrm{H_2O}}}{([\mathrm{A}]/\rho_{\mathrm{H_2O}})\,([\mathrm{B}]/\rho_{\mathrm{H_2O}})},\\ &= \frac{[\mathrm{AB}]\,\rho_{\mathrm{H_2O}}}{[\mathrm{A}][\mathrm{B}]},\end{aligned}$$

where for each complex, i, x_i is the *mole fraction*, $[i]$ is the concentration (e.g. in units of mol/L), ΔG_i is the free energy as reported by NUPACK, and $\rho_{\mathrm{H_2O}}$ (≈ 55.14 mol/L at 37°C) is the concentration of water.

Consider duplex formation for two RNA strands, A = `GCGCG` and B = `CGCGC`, present at concentrations of $[\mathrm{A}]_0$ and $[\mathrm{B}]_0$, respectively, in 1 M Na^+ at 37°C. The free energies given by NUPACK are

$$\Delta G_{\mathrm{A}} = 0.00 \text{ kcal/mol}, \quad \Delta G_{\mathrm{B}} = 0.00 \text{ kcal/mol}, \quad \Delta G_{\mathrm{AB}} = -9.62 \text{ kcal/mol}.$$

If only these three complexes are considered, the concentration of AB is determined by finding the appropriate root of

$$\frac{[\mathrm{AB}]\,\rho_{\mathrm{H_2O}}}{([\mathrm{A}]_0-[\mathrm{AB}])\,([\mathrm{B}]_0-[\mathrm{AB}])} = \exp\left\{-\frac{\Delta G_{\mathrm{AB}}-\Delta G_{\mathrm{A}}-\Delta G_{\mathrm{B}}}{kT}\right\}.$$

For $[\mathrm{A}]_0 = [\mathrm{B}]_0 = 1\ \mu$M, we get

$$[\mathrm{A}] = [\mathrm{B}] = 0.91\mu\mathrm{M}, \quad [\mathrm{AB}] = 0.09\mu\mathrm{M}.$$

A common mistake is to forget to include the $\rho_{\mathrm{H_2O}}$ in the calculation. Doing so would give the erroneous result of $[\mathrm{AB}] = 0.67\ \mu$M. It is important to remember that

$$\exp\left\{-\frac{\Delta G_{\mathrm{AB}}-\Delta G_{\mathrm{A}}-\Delta G_{\mathrm{B}}}{kT}\right\} \neq \frac{[\mathrm{AB}]}{[\mathrm{A}]\,[\mathrm{B}]}\,!$$

Finally, note that we have artificially stipulated that only three complexes are allowed. However, the sequences of A and B are such that they may form homodimers. If we consider this possibility, we are left with a system of coupled nonlinear algebraic equations that are difficult to solve. NUPACK performs such calculations, and the resulting concentrations are

$$[\mathrm{A}] = 0.686\ \mu\mathrm{M}, \quad [\mathrm{B}] = 0.925\ \mu\mathrm{M}, \quad [\mathrm{AB}] = 0.069\ \mu\mathrm{M}, \quad [\mathrm{AA}] = 0.123\ \mu\mathrm{M}, \quad [\mathrm{BB}] = 0.003\ \mu\mathrm{M},$$

significantly different from what we calculated neglecting the other complexes. Therefore, one must exercise caution when applying complex free energies to determination of equilibrium concentrations. It is best to directly use the `concentrations` executable or the NUPACK web application for these calculations.

2.2.3 `distributions`: calculate the equilibrium population distribution and expected value for a few complexes in a small box

Command: `distributions [-maxstates big] [-writestates] [-sort method] [-v3.0] [-quiet] prefix`

Description: The executable `distributions` calculates the partition function, Q_{box}, for a box containing a small number of strands, given user-defined populations for each strand species (Dirks et al., 2007). This is used to calculate the expected value and probability distribution of the population of each species of complex. Partition function information is read from output files generated with the executable `complexes`.

Additional options:

`-maxstates big`
The maximum number of states of the box to be enumerated (default: `1e7`). A segmentation fault will occur if the stack size on your machine is exceeded.

`-writestates`
Write a (typically large) output file describing properties for all population states of the system.

`-sort method`
The argument `method` is one of the following integers:

- `1`: (default) Output is sorted by the expected value of the population of each complex.
- `2`: Output is sorted first by the sum of the expected values of all complexes with each strand composition and then by the expected value of each complex with that strand composition.
- `3`: Output is sorted first by the complex composition identifier and then by the strand ordering identifier.
- `4`: Output is sorted first by the number of strands in each complex, then by the integers $A_{1,j}$ $A_{2,j}$ ... $A_{|\Psi^0|,j}$ defining the number of each strand type in complex j (with $A_{1,j}$ having the highest precedence, followed by $A_{2,j}$, and so on), and finally by the strand ordering identifier.

`-v3.0`
Revert to NUPACK 3.0 behavior (see changes).

`-quiet`
Suppress output to the screen.

Input: Same format as for the executable `concentrations`, except the file `prefix.con` file is replaced by `prefix.count`, which specifies the total strand population, m_i^0, for each strand species $i \in \Psi^0$ on a separate line. The last line of the file contains the volume of the box in liters. This may be entered in scientific notation (e.g., `1.4e-18`).

Output: Unless the `-quiet` flag is selected, the following information is written to the screen:

- The number of states of the box.
- The free energy of the entire box in units of kT and in units of kcal.
- The wall clock time for the calculation.

The output is written to the files:

- prefix.dist
  The content is the same as the input file (with rows sorted according to -sort) with extra columns after the free energy column. The first extra column (for complex j) is the expected value of the population $\langle m_j \rangle$. Subsequent columns are [$p_j(0)$ $p_j(1)$ ... $p_j(\max(m^0))$]. These represent the probability that complex j has population 0, 1, ..., $\max(m^0)$, at equilibrium.

- prefix.states
  Generated when -writestates is selected. Each row corresponds to a population vector, m, for the box. The first column is the probability that the population vector occurs at equilibrium. The remaining entries come in triples: strand composition identifier, strand ordering identifier, nonzero population. This pattern continues for all complexes with non-zero populations.

Changes: Relative to NUPACK 3.0, the following change was introduced to the distributions executable: the -ordered option is on by default. Use the -v3.0 option to revert to NUPACK 3.0 behavior.

3 Design

Design sequences over one of three design ensembles (Table 2):

- Complex Design: design the equilibrium base-pairing properties of a complex of (one or more) interacting nucleic acid strands (Zadeh et al., 2011b). Complex design is a special case of multistate test tube design, corresponding to a design ensemble comprising a single target test tube containing a single on-target complex and no off-target complexes.
- Test Tube Design: design the equilibrium base-pairing properties and concentrations of a test tube of interacting nucleic acid strands (Wolfe and Pierce, 2015). Test tube design is a special case of multistate test tube design, corresponding to a design ensemble comprising a single target test tube containing arbitrary numbers of on- and off-target complexes. Design jobs are specified using v1 of the NUPACK scripting language.
- Multistate Test Tube Design: design the sequences of multiple nucleic acid strands intended to hybridize in solution via a prescribed reaction pathway. Sequence design is formulated as a multistate optimization problem using a set of target test tubes to represent reactant, intermediate, and product states of the system, as well as to model crosstalk between components (Wolfe et al., 2017). Sequence design is performed subject to diverse user-specified sequence constraints. The multistate test tube design ensemble generalizes the complex design and test tube design ensembles, encompassing an arbitrary number of target test tubes, each containing arbitrary numbers of on- and off-target complexes. Design jobs are specified using v2 of the NUPACK scripting language.

For each design ensemble, the sequence is optimized by reducing a physically meaningful ensemble defect that quantifies design quality over the design ensemble. See (Wolfe et al., 2017) for a comparison of the three design problems, their ensemble defects, and a discussion of the positive and negative design paradigms implemented in each case. *We recommend using the multistate test tube design framework, which supports v2 of the NUPACK scripting language, and reduces to the two subsidiary design problems as special cases.*

Table 2: Comparison of nucleic acid sequence design ensembles

		Per target test tube	
Design problem	Target test tubes	On-target complexes	Off-target complexes
Complex design	1	1	0
Test tube design	1	Arbitrary	Arbitrary
Multistate test tube design	Arbitrary	Arbitrary	Arbitrary

3.1 Complex Design

Design the equilibrium base-pairing properties of a complex of interacting nucleic acid strands.

3.1.1 `complexdesign`: design the equilibrium base-pairing properties of a complex

Command: `complexdesign [-init initmode] [-loadinit] [-outputinit] [-loadseed] [-outputseed] [-fstop fstopvalue] [-prevent file] [-mleafopt mleafoptvalue] [-mreopt mreoptvalue] [-pairs] [-cutoff cutoffvalue] prefix`

Description: Perform sequence design over the ensemble of a complex of interacting nucleic acid strands. The user specifies a target secondary structure. Sequence design is formulated as an optimization problem with the goal of reducing the complex ensemble defect below a user-specified stop condition (Zadeh et al., 2011b).

Additional options:

`-init initmode`
The argument `initmode` selects the sequence initialization method from one of the following (using a sequence that satisfies the base-pairing requirements of the target secondary structure with Watson-Crick pairs):

`AU`: Initial sequences are randomly selected from `A` and `T/U` bases only.

`CG`: Initial sequences are randomly selected `C` and `G` bases only.

`RND`: (default) Initial sequences are randomly selected from `A,C,G,T/U`.

`SSM`: Initial sequences are generated using sequence symmetry minimization (Seeman, 1982; Dirks et al., 2004).

`-loadinit`
Initialize the sequence from the file `prefix.init`.

`-outputinit`
Output the initial sequence to `prefix.init`.

`-loadseed`
Initialize the random number generator with the seed specified in `prefix.seed`. This can be used to duplicate design execution.

`-outputseed`
Output the random number generator's seed to `prefix.seed`.

`-fstop fstopvalue`
Set the stop condition for the design algorithm to `fstopvalue` (default: 0.01). The design algorithm seeks to achieve $n(\phi, s) \leq$ `fstopvalue` $|\phi|$.

`-prevent preventfile`
The file `preventfile` contains patterns to be prevented from appearing in the sequence design.

`-mleafopt mleafoptvalue`
Leaf optimization is restarted from new initial conditions up to `leafoptvalue` times (default: 3) before terminating unsuccessfully (Zadeh et al., 2011b).

`-mreopt mreoptvalue`
The elimination of emergent defects in a parent node by defect-weighted child sampling and reoptimization is attempted up to `mreoptvalue` times (default: 10) (Zadeh et al., 2011b).

`-pairs`
Save the pair probabilities in a `.ppairs` file.

`-cutoff cutoffvalue`
Only probabilities at or above `cutoffvalue` are saved in the `.ppairs` file (default: 0.001).

Input: The target structure and sequence constraints are read from `prefix.fold`. The first line of the file is the target structure in dot-parens-plus notation. The second line of the file contains the sequence constraints (if any) specified using IUPAC nucleic acid codes (Table 3). If no sequence constraints are specified for a given base, it is assumed to be unconstrained. Comment lines are preceded by `%` and may be interspersed with data in input files; blank lines are not permitted.

Optional inputs are specified in the following files:

`prefix.init`: Used when `-loadinit` is specified. The first line in the file is the initial sequence.

`prefix.seed`: Used when `-loadseed` is specified. The first line is an integer random seed for the design algorithm (unique seeds are in the range $[0, 2^{32} - 1]$).

`preventfile`: Specifies patterns to be prevented from appearing in the designed sequences. The file must contain one pattern per line using standard nucleic acid codes. No design will be produced if the sequence constraints cannot be satisfied.

Output: Output is written to the files:

- `prefix.summary`
  The header of this file includes comments about thermodynamic and design parameters used in the design process. The first line below the header contains the strand sequences separated by + symbols.
- `prefix.init`
  Generated if `-outputinit` is specified. It contains the initial sequence on the first line of the file.
- `prefix.seed`
  Generated if `-outputseed` is specified. It contains the random seed on the first line of the file.
- `prefix.ppairs`
  Generated if `-pairs` is specified. This file specifies the base pairing probabilities for the complex in the same format as the file generated by the executable `pairs`.

Table 3: IUPAC degenerate nucleotide codes for RNA*

Code	Nucleotides
M	A or C
R	A or G
W	A or U
S	C or G
Y	C or U
K	G or U
V	A, C, or G
H	A, C, or U
D	A, G, or U
B	C, G, or U
N	A, C, G, or U

*T replaces U for DNA

3.1.2 `complexdefect`: calculate the complex ensemble defect

Command: `complexdefect [-T temperature] [-pseudo] [-multi] [-material parameters] [-dangles treatment] [-mfe] prefix`

Description: Calculate the complex ensemble defect, $n(\phi, s)$, representing the average number of incorrectly paired nucleotides at equilibrium evaluated over the ensemble of the complex, Γ (Dirks et al., 2004; Zadeh et al., 2011b). Here, ϕ is the sequence and s is the target secondary structure.

Additional option:

`-mfe`
Instead, calculate the complex MFE defect, $\mu(\phi, s)$, representing the number of incorrectly paired nucleotides in the MFE structure s^{MFE} (Zadeh et al., 2011b).

Example 4: Design a sequence for a complex of three DNA strands intended to adopt a target secondary structure at 23 °C. The first 24 nucleotides are constrained to nucleotide `H` (corresponding to a 3-letter alphabet) and the specified list of patterns are prevented throughout. A seed is set to make the design repeatable.

Input file contents:

```
((((((((((((((((((((((+(((((((((((((((((((((.....................
.+))))))))))))))))))))))))))))))))))))))))))))
HHHHHHHHHHHHHHHHHHHHHHH
```

Prevent file contents:

```
AAAA
CCCC
GGGG
UUUU
KKKKKK
MMMMMM
RRRRRR
SSSSSS
WWWWWW
YYYYYY
```

Seed file contents:

```
93
```

Command: `complexdesign -T 23 -material dna -pairs -loadseed -prevent $NUPACKHOME/doc/examples/complex-design/advanced/input/hcr-design.prevent $NUPACKHOME/doc/examples/complex-design/advanced/input/hcr-design`

Input: Same format as for the executable `energy`.

Output: Following header comments, the complex ensemble defect, $n(\phi, s)$, and the normalized complex ensemble defect, $n(\phi, s)/|\phi|$, are written to the screen. If the `-mfe` flag is selected, the complex MFE defect, $\mu(\phi, s)$, and the normalized complex MFE defect, $\mu(\phi, s)/|\phi|$, are written to the screen.

3.2 Test Tube Design

Design the equilibrium base-pairing properties and concentrations for of a test tube of interacting nucleic acid strands.

3.2.1 `tubedesign`: design the equilibrium base-pairing properties of a test tube

Command: `tubedesign prefix`

Description: Perform sequence design for a test tube of interacting nucleic acid strands. The user specifies a set of desired on-target complexes, each with a target secondary structure and target concentration, and a set of undesired off-target complexes, each with vanishing target concentration. The set of off-target complexes is specified to be all complexes up to a user-specified number of strands (excluding those complexes that are on-target complexes). Sequence design is formulated as an optimization problem with the goal of reducing the test tube ensemble defect below a user-specified stop condition (Wolfe and Pierce, 2015).

Input: The executable `tubedesign` reads a job description from file `prefix.np` written in v1 of the NUPACK scripting language. In a `.np` file, a comment begins with # and continues for the rest of the line; blank lines are permitted. Specification of a test tube design job is illustrated in Example 5.

Output: Output is written to commented output file `prefix.out`.
See examples in `$NUPACKHOME/doc/examples/tube-design/`

Optional commands: In a `.np` script file using v1 of the NUPACK scripting language, parameter values are set as follows (defaults shown):

```
# physical model parameters: see options for details
material = rna               # values: rna, dna, rna1995, rna1999, dna1998
temperature = 37.0           # °C
sodium = 1.0                 # in interval [0.05,1.1], molar
magnesium = 0.0              # in interval [0.0,0.2], molar
dangles = some               # values: none, some, all

# algorithm parameters: see (Wolfe and Pierce, 2015) for details
hsplit = 2                   # default: 2 for rna, 3 for dna
nsplit = 12                  #
fsplit = 0.99                # in interval (0,1)
fstringent = 0.99            # in interval (0,1)
dgclamp = -25.0              # kcal/mol
mbad = 300                   #
mreseed = 50                 #
mreopt = 3                   #
fpassive = 0.01              # in interval (0,1)
fredecomp = 0.03             # in interval (0,1)
frefocus = 0.03              # in interval (0,1)
allowwobble = false          # allow algorithm to introduce wobble pairs
initgc = 0.5                 # in interval [0,1], initial GC content
maxopttime = 31536000        # seconds
```

Example 5: Test tube design. Design a sequence for a target test tube at default temperature 37 °C. The target test tube contains 1 on-target dimer (with a target structure and target concentration) and all off-target complexes of up to 2 strands (each with vanishing target concentration).

Script file contents:

```
# set properties
material = dna
seed = 93 # set seed to make design repeatable

# define target structure for 1 on-target complex
structure legs = ((((((((((((((((((.....................+.............
........))))))))))))))))))

# define target test tube containing 1 on-target complex
tube walker = legs

# define target concentration for 1 on-target complex (molar)
# default: 1.0e-6
walker.legs.conc = 1.0e-6

# augment tube with all off-target complexes of up to 2 strands
# default: 0
walker.maxsize = 2
```

Command: `tubedesign`
`$NUPACKHOME/doc/examples/tube-design/simple/input/walker-design`

3.2.2 `tubedefect`: calculate the test tube ensemble defect

Command: `tubedefect` `prefix`

Description: Calculate the test tube ensemble defect, C, representing the concentration of incorrectly paired nucleotides evaluated over the ensemble of a target test tube (Wolfe and Pierce, 2015). The target test tube is specified as a set of desired on-target complexes, each with a target secondary structure and target concentration, and a set of undesired off-target complexes, each with vanishing target concentration. The set of off-target complexes is specified to be all complexes up to a user-specified number of strands (excluding those complexes that are on-target complexes).

Input: The executable `tubedefect` reads a job description from file `prefix.np` written in v1 of the NUPACK scripting language. In `.np` script files, a comment begins with # and continues for the rest of the line; blank lines are permitted. A sample script file is shown in Example 6. See executable `tubedesign` for additional details on v1 of the NUPACK scripting language.

Output: Following header comments, the test tube ensemble defect, C, and the normalized test tube ensemble defect, C/y^{nt}, are written to the screen (Wolfe and Pierce, 2015). Here, y^{nt} is the total concentration of nucleotides in the test tube.

Example 6: Test tube ensemble defect. Calculate the test tube ensemble defect for a specified set of sequences and target test tube at default temperature 37 °C. The target test tube contains 1 on-target dimer (with a target structure and target concentration) and all off-target complexes of up to 2 strands (each with vanishing target concentration).

Script file contents:

```
# set properties
material = dna

# define 2 sequence domains
domain a = CGTGAACATCGGCGTGGTCGACCAACCCCACACAAAAAACCTA
domain b = TTCCCTCTATATTTCTACACTCCCGACCACGCCGATGTTCACG

# define 2 strands
strand leg1 = a
strand leg2 = b

# define target structure for 1 on-target complex
structure legs = ((((((((((((((((((((.....................+.............
.......)))))))))))))))))))

# define strand ordering for 1 on-target complex
legs.seq = leg1 leg2

# define target test tube containing 1 on-target complex
tube walker = legs

# define target concentration for 1 on-target complex (molar)
# default:  1.0e-6
walker.legs.conc = 1.0e-6

# augment tube with all off-target complexes of up to 2 strands
# default:  0
walker.maxsize = 2
```

Command: `tubedefect`
`$NUPACKHOME/doc/examples/tube-design/simple/input/walker-defect`

3.3 Multistate Test Tube Design

Design the sequences of multiple nucleic acid strands intended to hybridize in solution via a prescribed reaction pathway (Wolfe et al., 2017).

3.3.1 `multitubedesign`: design the equilibrium base-pairing properties of a test tube

Command: `multitubedesign` `prefix`

Description: Sequence design is formulated as a multistate optimization problem using a set of target test tubes to represent reactant, intermediate, and product states of the system, as well as to model crosstalk between components. Each target test tube contains a set of desired on-target complexes, each with a target secondary structure and target concentration, and a set of undesired off-target complexes, each with vanishing target concentration. Design quality is quantified by the multistate test tube ensemble defect, $\mathcal{M}$, representing the average equilibrium fraction of incorrectly paired nucleotides evaluated over the design ensemble (Wolfe et al., 2017). Optimization of the sequences so as to reduce $\mathcal{M}$ below a user-specified stop condition implements both a positive design paradigm, explicitly designing for on-pathway elementary steps, and a negative design paradigm, explicitly designing against off-pathway crosstalk. Sequence design is performed subject to diverse user-specified sequence constraints including composition constraints, complementarity constraints, pattern prevention constraints, and biological constraints. The ensemble of complexes in the test tube can be specified explicitly or combinatorially.

Sequence constraints: The following types of sequence constraints can be imposed (Wolfe et al., 2017):

- *Assignment Constraint.* Nucleotide a is constrained to have a specified sequence (e.g., `A`, `C`, `G`, `U` or any of the IUPAC degenerate nucleotide codes; see Table 3).
- *Match Constraint.* Two nucleotides a and b are constrained to be identical (e.g., if a strand species appears in more than one on-target complex, corresponding nucleotides are constrained to have the same sequence in all complexes).
- *Watson–Crick Constraint.* Two nucleotides a and b are constrained to be Watson-Crick complements (by default, Watson–Crick constraints are implied for all base pairs present in on-target structures).
- *Complementarity Constraint.* Two nucleotides a and b are constrained to be Watson–Crick or wobble complements.
- *Composition Constraint.* Consecutive nucleotides $a, \ldots, b$ are constrained to have a sequence composition in a specified range (e.g., a desired `GC` content can be achieved by constraining the fraction of `S` nucleotides to fall in the range $[f^{\min}, f^{\max}]$).
- *Similarity Constraint.* Consecutive nucleotides $a, \ldots, b$ are constrained to be similar to a specified sequence of length $n = b - a + 1$ to a specified degree (e.g., the fraction of nucleotides matching an mRNA sequence can be constrained to fall in the range $[f^{\min}, f^{\max}]$).
- *Pattern Prevention Constraint.* Consecutive nucleotides $a, \ldots, b$ are constrained not to contain a specified subsequence of length $n \leq b - a + 1$ (e.g., prevention of `GGGG`, which is prone to forming `G`-quadruplexes (Saini et al., 2013) that are not accounted for in nearest-neighbor free energy models (Mathews et al., 1999; SantaLucia and Hicks, 2004)).
- *Library Constraint.* Consecutive nucleotides $a, \ldots, b$ are constrained to be selected from a specified library of m sequences of length $n = b - a + 1$ (e.g., a library of toehold sequences or a library of codons).
- *Window Constraint.* Consecutive nucleotides $a, \ldots, b$ are constrained to be a subsequence of a specified source sequence of length $n \geq b - a + 1$ (e.g., the source sequence is an mRNA), or more generally, a subsequence of one of multiple specified source sequences.

Each constraint is expressed as a constraint relation (Table 4). For some constraint relations, it is convenient to make

use of the sequence distance function,

$$d(\phi, q) \equiv \sum_{a \in 1,\ldots,|\phi|} \begin{cases} 0 : \phi^a \in q^a \\ 1 : \phi^a \notin q^a \end{cases},$$

between sequence ϕ and the constraint sequence q of equal length, which may contain degenerate IUPAC nucleotide codes (see Table 3). For example, $d(\texttt{ACGU}, \texttt{SSWW}) = 2$.

Table 4: Sequence constraints (Wolfe et al., 2017)

Constraint type	Constraint relation*	Nucleotides
Assignment	$(\phi^a) \in R_a^{\text{assignment}} \equiv \{(q^1)\}$	1
Match	$(\phi^a, \phi^b) \in R_{a,b}^{\text{match}} \equiv \{(\texttt{A},\texttt{A}), (\texttt{C},\texttt{C}), (\texttt{G},\texttt{G}), (\texttt{U},\texttt{U})\}$	2
Watson–Crick	$(\phi^a, \phi^b) \in R_{a,b}^{\text{WC}} \equiv \{(\texttt{A},\texttt{U}), (\texttt{C},\texttt{G}), (\texttt{G},\texttt{C}), (\texttt{U},\texttt{A})\}$	2
Complementarity	$(\phi^a, \phi^b) \in R_{a,b}^{\text{complement}} \equiv \{(\texttt{A},\texttt{U}), (\texttt{C},\texttt{G}), (\texttt{G},\texttt{C}), (\texttt{U},\texttt{A}), (\texttt{G},\texttt{U}), (\texttt{U},\texttt{G})\}$	2
Composition	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{composition}} \equiv \{(\phi^a, \ldots, \phi^b) \vert f^{\min} \leq \sum_{i=a,\ldots,b} d(\phi^i, q^1)/n \leq f^{\max}\}$	$b-a+1=n$
Similarity	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{similarity}} \equiv \{(\phi^a, \ldots, \phi^b) \vert f^{\min} \leq d((\phi^a, \ldots, \phi^b), (q^1, \ldots, q^n))/n \leq f^{\max}\}$	$b-a+1=n$
Pattern prevention	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{pattern}} \equiv \{(\phi^a, \ldots, \phi^b) \vert (q^1, \ldots, q^n)$ is not a subsequence of $(\phi^a, \ldots, \phi^b)\}$	$b-a+1 \geq n$
Library	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{library}} \equiv \{(q_1^1, \ldots, q_1^n), \ldots, (q_m^1, \ldots, q_m^n)\}$	$b-a+1=n$
Window	$(\phi^a, \ldots, \phi^b) \in R_{a,\ldots,b}^{\text{window}} \equiv \{(\phi^a, \ldots, \phi^b) \vert (\phi^a, \ldots, \phi^b)$ is a subsequence of $(q^1, \ldots, q^n)\}$	$b-a+1 \leq n$

*For user-specified $q^i \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}, \texttt{M}, \texttt{R}, \texttt{W}, \texttt{S}, \texttt{Y}, \texttt{K}, \texttt{V}, \texttt{H}, \texttt{D}, \texttt{B}, \texttt{N}\}$.

Input: The executable `multitubedesign` reads a job description from file `prefix.np` written in v2 of the NUPACK scripting language. In a `.np` file, a comment begins with # and continues for the rest of the line; blank lines are permitted. Specification of a multistate test tube design job employing sequence domains to enforce complementarity constraints is illustrated in Example 7.

Output: Output is written to commented output file `prefix_0.npo`.
See examples in `$NUPACKHOME/doc/examples/multitube-design/`

Figure 3: Reaction pathway schematic for Example 7. Conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs (scRNAs) (Hochrein et al., 2013). scRNA A·B detects input X (comprising sequence 'a-b-c'), leading to production of Dicer substrate B·C (targeting independent sequence 'w-x-y-z'). Step 1: X displaces A from B via toehold-mediated 3-way branch migration and spontaneous dissociation. Step 2: B assembles with C via loop/toehold nucleation and 3-way branch migration to form Dicer substrate B·C. See (Wolfe et al., 2017) for additional reaction pathway case studies from the molecular programming literature.

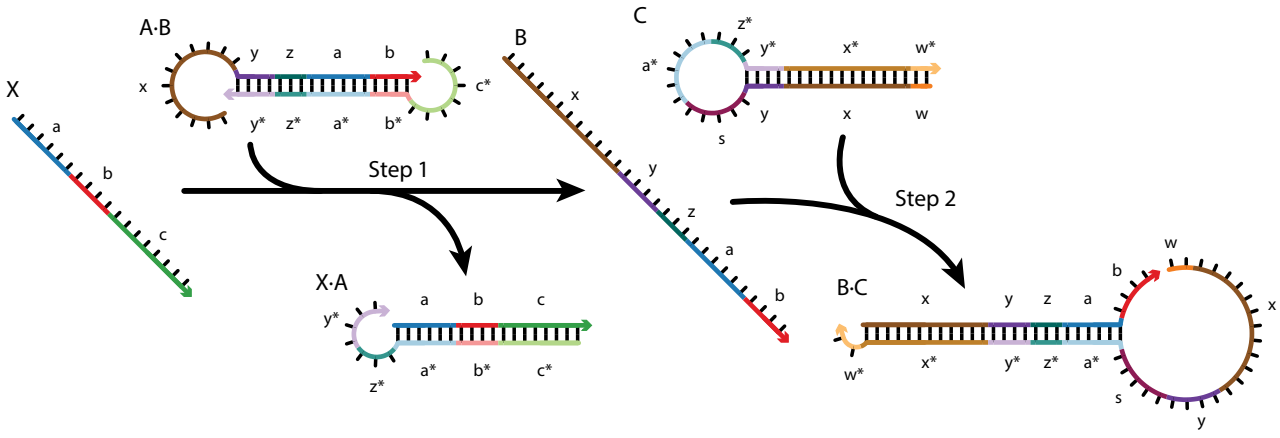
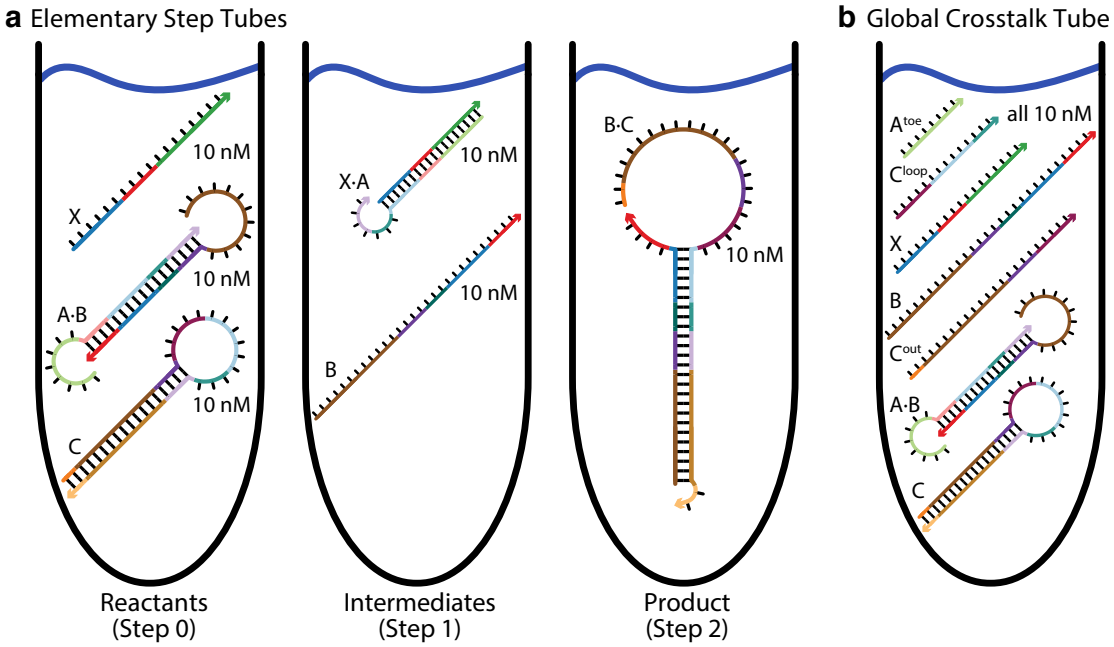


Figure 4: Target test tube specification for Example 7. Conditional Dicer substrate formation via shape and sequence transduction with scRNAs. (a) Elementary step tubes. Reactants tube (Step 0): target X and scRNAs A·B and C. Step 1 tube: X·A and B. Step 2 tube: Dicer substrate B·C. Each target test tube contains the depicted on-target complexes corresponding to the on-pathway products for a given step (each with the depicted target secondary structure and a target concentration of 10 nM) as well as off-target complexes (not depicted) corresponding to on-pathway reactants and off-pathway crosstalk for a given step. (b) Global crosstalk tube. Contains the depicted on-target complexes corresponding to reactive species generated during Steps 0, 1, 2 as well as off-target complexes (not depicted) corresponding to off-pathway interactions between these reactive species. Design conditions: RNA in 1 M Na^+ at 37 °C. See (Wolfe et al., 2017) for a description of how to specify target test tubes for a given reaction pathway.



Example 7: Reaction pathway engineering via constrained multistate test tube design. Design strands for conditional Dicer substrate formation via shape and sequence transduction with scRNAs. Impose sequence constraints: GC-content in range $[0.45, 0.55]$, prevent patterns $\{\text{AAAA}, \text{CCCC}, \text{GGGG}, \text{UUUU}\}$, and input and output targets are subsequences of biological sequences. See the reaction pathway of Figure 3 and the target test tubes of Figure 4 (Wolfe et al., 2017).

Script file contents:

```
# set physical parameters
temperature[C] = 37.0
material = rna1999
seed = 93 # set seed to make design repeatable

# define domains
domain a = N6
domain c = N8
domain b = N4
domain w = N2
domain y = N4
domain x = N12
domain z = N3
domain s = N5

# define strands from domains
strand Cout_s = w x y s
strand A_s = c* b* a* z* y*
strand A_toe_s = c*
strand C_s = w x y s a* z* y* x* w*
strand C_loop_s = s a* z*
strand B_s = x y z a b
strand Xs_s = a b c

# define complexes composed of one or more strands in a given order
complex C = C_s
complex B = B_s
complex C_loop = C_loop_s
complex A_B = A_s B_s
complex X = Xs_s
complex X_A = Xs_s A_s
complex C_out = Cout_s
complex B_C = B_s C_s
complex A_toe = A_toe_s

# define target structures for each complex
C.structure = D2 D12 D4( U5 U6 U3 )
B.structure = U12 U4 U3 U6 U4
C_loop.structure = U14
A_B.structure = U8 D4 D6 D3 D4(+ U12)
X.structure = U18
X_A.structure = D6 D4 D8(+) U3 U4
C_out.structure = U23
B_C.structure = D12 D4 D3 D6 (U4 + U2 U12 U4 U5) U2
A_toe.structure = U8
```

Example 7 continued...

```
# define elementary step tubes
tube Step_0 = C X A_B
Step_0.C.conc[M] = 1e-08
Step_0.X.conc[M] = 1e-08
Step_0.A_B.conc[M] = 1e-08
Step_0.offtargets = {maxsize = 2} + {A_s, B_s} - {X_A}

tube Step_1 = X_A B
Step_1.X_A.conc[M] = 1e-08
Step_1.B.conc[M] = 1e-08
Step_1.offtargets = {maxsize = 2} + {X, A_B}

tube Step_2 = B_C
Step_2.B_C.conc[M] = 1e-08
Step_2.offtargets = {maxsize = 2} + {B, C}

# define global orthogonality tube
tube Crosstalk = A_B C X B C_out C_loop A_toe
Crosstalk.offtargets = {maxsize = 2}\
                     - {X_A, B_C, Xs_s A_toe_s, B_s C_loop_s}
Crosstalk.A_B.conc[M] = 1e-08
Crosstalk.C.conc[M] = 1e-08
Crosstalk.X.conc[M] = 1e-08
Crosstalk.B.conc[M] = 1e-08
Crosstalk.C_out.conc[M] = 1e-08
Crosstalk.C_loop.conc[M] = 1e-08
Crosstalk.A_toe.conc[M] = 1e-08
Crosstalk.weight[frac] = 1

# GC content constraints
similarity Cout_s_m = S23
Cout_s_m.similarity[frac] Cout_s = [0.45, 0.55]
similarity A_s_m = S25
A_s_m.similarity[frac] A_s = [0.45, 0.55]
similarity C_s_m = S50
C_s_m.similarity[frac] C_s = [0.45, 0.55]
similarity C_loop_s_m = S14
C_loop_s_m.similarity[frac] C_loop_s = [0.45, 0.55]
similarity B_s_m = S29
B_s_m.similarity[frac] B_s = [0.45, 0.55]
similarity Xs_s_m = S18
Xs_s_m.similarity[frac] Xs_s = [0.45, 0.55]
```

Example 7 continued...

```
# sources lines
source tpm3 = gaacacuauuagcuauuuguaguacucuaaagaggacugcagaacgcaucgcaguagugg\
ugaaaagccgugcgugcgcgugaaacaucugauccucacguuacuuccacucgcucugcg\
uuugacuuguuggcggggcguuggugccuuggacuuuuuuuuccuccuucucuucuucgc\
ggcucgguccacuacgcugcucgagaggaaucugcuuuauucgaccacacuacuccuaaa\
guaacacauuaaaauggccggaucaaacagcaucgaugcaguuaagagaaaaaucaaagu\
uuuacaacagcaagcagaugaggcagaagaaagagccgagauuuugcagagacaggucga\
ggaggagaagcgugccagggagcaggcugaggcagagguggcuucucugaacaggcguau\
ccagcugguugaggaggaguuggaucgugcucaggagagacuggccacagcccugcaaaa\
gcuggaggaagccgagaaggccgcagaugagagcgagagagggaugaaggugauugagaa\
cagggcucugaaggaugaggagaagauggagcugcaggagauccagcuuaaggaggccaa\

window tpm3_window = a b c
tpm3_window.source = tpm3

source desm = cauuuacacagcguacaaacccaacaggcccagucaugagcacgaaauauucagccuccg\
ccgagucggcguccucuuaccgccgcaccuuuggcucagguuugggcuccucuauuuucg\
ccggccacgguuccucagguuccucuggcuccucaagacugaccuccagaguuuacgagg\
ugaccaagagcuccgcuucucccccauuuuuccagccaccgugcguccggcucuuucggag\
guggcucggugguccguuccuacgcuggccuuggugagaagcuggauuucaaucuggcug\
augccauaaaccaggacuuccucaacacgcguacuaaugagaaggccgagcuccagcacc\
ucaaugaccgcuucgccagcuacaucgagaaggugcgcuuccucgagcagcagaacucug\
cccugacgguggagauugagcgucugcggggucgcgagcccacccguauugcagagcugu\
acgaggaggagaugagagagcugcgcggacagguggaggcacugaccaaucagagauccc\
guguggagaucgagagggacaaccuagucgaugaccuacagaaacuaaagcucagacuuc\

window desm_window = w x y z
desm_window.source = desm

# global pattern prevention constraint
prevent = AAAA,CCCC,GGGG,UUUU

# stop condition
stop[%] = 5
```

Command: `multitubedesign`
`$NUPACKHOME/doc/examples/multitube-design/advanced/input/dicer-design`

3.3.2 `multitubedefect`: calculate the multistate test tube ensemble defect

Command: `multitubedefect` `prefix`

Description: Calculate the multistate test tube ensemble defect, $\mathcal{M}$, representing the average equilibrium fraction of incorrectly paired nucleotides evaluated over the ensemble of a set of target test tubes (Wolfe et al., 2017). Each target test tube is specified as a set of desired on-target complexes, each with a target secondary structure and target concentration, and a set of undesired off-target complexes, each with vanishing target concentration. The ensemble of complexes in the test tube can be specified explicitly or combinatorially.

Input: The executable `multitubedefect` reads a job description from file `prefix.np` written in v2 of the

NUPACK scripting language. In `.np` script files, a comment begins with # and continues for the rest of the line; blank lines are permitted. A sample script file is shown in Example 8.

Output: Output is written to commented output file `prefix_0.npo`.

Example 8: Multistate test tube ensemble defect. Calculate the multistate test tube ensemble defect, $\mathcal{M}$, for a specified target test tube at 23 °C. The target test tube contains 1 on-target tetramer (with a target structure and target concentration) and all off-target complexes of up to 3 strands (each with vanishing target concentration).

Script file contents:

```
# set physical properties
material = rna
temperature = 23.0

# define domains
domain a = ACCUCCAAGCACAACUGUGGCCCCAUA
domain b = GGGGCCGGAUUACAACUUUCCCUGUGAAC
domain c = AUCACAGACAGUUAACCACUUGAGG
domain d = AUCAAGUGGGCUUGGAGC

# define strands from domains
strand left = a
strand top = b
strand right = c
strand bottom = d

# define complex composed of strands in a given order
complex stickfigure = left top right bottom

# define target structure for complex
stickfigure.structure = U2D8(U2D6(D6(U3+)D3U9D6(U2+U1))U2D8(U2+U1))U1

# define tube
tube figuretube = stickfigure
figuretube.stickfigure.conc = 1.0e-6
figuretube.offtargets = {maxsize = 3}    # all complexes of up to 3 strands
```

Command: `multitubedefect $NUPACKHOME/doc/examples/multitube-design/simple/input/stickman_tube-defect`

3.3.3 Design script syntax (v2)

Comments, whitespace, and line continuation. Whitespace is required to separate reserved words from names, but is optional when unambiguous. A line may include a comment, which begins with the '#' character and continues for the rest of the line. Any statement may be extended across multiple lines by adding a '\' character followed by optional whitespace. The '\' and '#' characters may not be used on the same line. Blank lines are allowed between definitions.

Names. Naming of design elements (domains, strands, complexes, tubes, etc.) are case sensitive and follow the same rules as identifiers in Python (i.e., at least one alphabetical character or underscore followed by zero or more alphanumeric characters or underscores). *Names are global and type-independent.* Redundant definitions are ambiguous and produce an error message.

Numbers. Numbers are either integer or floating point. Integers are represented as a series of one or more digits starting with a nonzero digit (e.g., 10 not 010). Floating point numbers may be represented as integers, or with a decimal point, or using "e"-based scientific notation (e.g. 10, 10.0, 1e1, and 1.0e1 are all equivalent).

Nucleotide codes. Sequence constraints can be specified using degenerate nucleotide codes (Table 3) to indicate which nucleotides are allowed at a given position.

Reserved words. The following words are reserved by the scripting language and cannot be used to name design elements in a script:

```
seed, maxsize, material, sodium, magnesium, temperature, mbad, mreopt,
fsplit, nsplit, hsplit, dangles, structure, tube, domain, strand,
conc, stop, false, true, fstringent, fredecomp, frefocus, fpassive,
allowwobble, pairscutoff, trials, maxopttime, prevent, library, libseq,
source, window, similarity, complement, match, weight, complex,
offtargets, mreseed, dgclamp
```

Physical parameters. In v2 of the NUPACK scripting language, physical parameter values may be set as follows (defaults shown):

```
# physical model parameters: see options for details
material = rna          # values: rna, dna, rna1995, rna1999, dna1998, custom
temperature = 37.0      # specified in °C (default) or K
temperature[C] = 37.0
temperature[K] = 310.15
sodium = 1.0            # in interval [0.05,1.1], molar
sodium[M] = 1.0         # optional specification of molar units
sodium[mM] = 1000       # optional use of alternate units
magnesium = 0.0         # in interval [0.0,0.2], molar
dangles = some          # values: none, some, all
```

Specify a domain. A domain is a set of consecutive nucleotides that appear as a subsequence of one or more strands in the design. The sequence of a domain is specified 5' to 3' using degenerate nucleotide codes (Table 3). Consecutive repeats of a single code can be represented by the nucleotide code followed by the total number of repeats:

```
domain a = AAAA
domain b = A4 # equivalent specification

domain c = NNNNNNNNNN
domain d = N10 # equivalent specification

domain e = RRSSAAACCA
domain f = R2S2 A3C2 A # equivalent specification
```

Names of domains are referenced in other parts of the script. The reverse complement of domain 'a' is referred to as 'a*'.

Specify a strand. A strand is an oligonucleotide defined by a whitespace-separated list of previously defined domain names:

```
strand A = a b c
strand B = d e*       # 'e*' is reverse complement of 'e'
strand C = e a f
strand D = d d d
```

Sequence domains provide a convenient approach for specifying sequence constraints between different strands intended to interact via a prescribed reaction pathway.

Specify an on-target complex. An on-target complex is specified in two steps. First, the `complex` command is used to name the complex and specify a strand ordering as a whitespace-separated list of previously defined strand names. Second, a target structure is specified in dot-parens-plus notation or DU+ notation. The target structure for a complex is used in all target test tubes in which the complex appears as an on-target.

```
complex C1 = A B C
complex C2 = D D
complex C3 = B B B
complex C4 = B A B
complex C5 = B C

# specify a target structure for each on-target complex
C1.structure = \
........((((((((((+))))))))))((((((((((+))))))))))..............
C2.structure = D30 +
C3.structure = D10(D10 + D10 +)
C4.structure = D8(U12 +) D10(+) U10
C5.structure = U10 D10(+) U10
```

Specify a test tube. A test tube ensemble is defined in three steps. First, the `tube` command is used to name the tube and list the on-target complexes as a whitespace-separated list of previously defined complexes. Second, the target concentration for each on-target complex is specified in units of M (default), or optionally mM, uM (read as μM), nM, pM, fM, aM, zM, or yM; the default target concentration is 1e-6 M. Third, the off-target complexes in the test tube are specified by adding or subtracting sets of complexes (each set is delimited by curly braces and contains comma-separated complexes as illustrated below). Off-target complexes are specified in one of three ways: using a previously named on-target complex to indicate a strand ordering, specifying an unnamed strand ordering, or combinatorially (all off-target complexes up to a specified number of strands).

```
# specify on-target complexes in tube
tube T1 = C1
tube T2 = C2
tube T3 = C1 C2
tube T4 = C1 C3
tube T5 = C4 C5
tube T6 = C5

# note:  target structure previously specified for C1 and C2
# as part of complex specification

# specify target concentration for each on-target complex
T1.C1.conc = 1e-6      # using default units (M)
```

```
T2.C2.conc[uM] = 1     # using specified units of uM (micromolar)
T3.C1.conc = 0.000001
T3.C2.conc = 1e-3
T4.C1.conc = 2e-4
T4.C3.conc = 3e-5
T5.C4.conc = 4e-6
T5.C5.conc = 5e-7
T6.C5.conc = 6e-8

# specify named off-target complexes in tube
T1.offtargets = {C4, C5}

# specify unnamed off-targets each denoted by a strand ordering
T2.offtargets = {D D D, D D D D}

# specify combination of named and unnamed off-targets
T3.offtargets = {C3, A A B B, C, D D D D}

# specify off-targets combinatorially (default:  maxsize = 0)
T4.offtargets = {maxsize = 2} # all complexes of up to 2 strands that
                              # that are not on-targets in tube 'T4'

# specify off-targets as the sum of sets
T5.offtargets = {maxsize = 2} + {C3, B B B B}

# specify off-targets as the difference of sets
T6.offtargets = {maxsize = 3} - {C3, B B}
```

Note: any defined on-target complex that is not included in a tube is implicitly assigned to its own tube containing no off-targets (i.e. complex design), and this tube is included in the multistate test tube ensemble.

Specify sequence constraints.

- **Match constraint.** Equal length concatenations of one or more domains are constrained to be identical as follows:

```
domain a = N10
domain b = N4
domain c = H6
domain d = N6
domain e = S2

match c = b e*       # 'e*' is reverse complement of 'e'
match a b = d d e
```

- **Complementarity constraint.**

  Equal-length concatenations of domains can be constrained to be reverse complements using a complementarity constraint. By default, a complementarity constraint will impose Watson-Crick base-pairing (A· U or C·G for RNA, A·T or C·G for DNA). To permit wobble pairs in a design (G·U for RNA, G·T for DNA), set the global flag `allowwobble`.

```
# concatenation 'c-d-e' reverse complement of concatenation 'a-b'
complement a b = c d e


# allow wobble pairs
allowwobble = true      # values:  true or false (default)
```

It is also possible to force base pairs to be wobble pairs as illustrated below:

```
# force wobble pairs
domain e = S2
domain f = S2
allowwobble = true
complement e = f # both domains are all G or U and 'allowwobble' is true
```

- **Similarity constraint.** A similarity constraint is specified in two steps. First, specify the sequence to which similarity is required. Second, specify the degree of similarity as a fraction (default) or percent:

```
domain a = N10
similarity x = S5 N5


a.similarity x = [0.25,0.75]
a.similarity[%] x = [25,75] # equivalent specification
```

Composition constrains are specified using the similarity constraint syntax as follows:

```
domain b = N20
similarity GCcontent = S20


b.similarity GCcontent = [0.45,0.55] # enforce 45-55% GC content
```

- **Window constraint.** A window constraint is specified in three steps. First, define a source sequence (typically a long sequence) from which sequences constrained by window constraints will be selected. Second, define a window concatenated from domains and/or strands that will be drawn as a continguous subsequence of a source sequence. Third, constrain the window to be drawn from the source. More generally, a window can be drawn from any of set of sources, written as a whitespace-separated list.

```
# define source sequence; note line continuation syntax
source GFP = \
auggugagcaagggcgaggagcuguucaccggggugguugcccauccuggu\
cgagcuggacggcgacguaaacggccacaaguucagcguguccggcgagg\
gcgagggcgaugccaccuacggcaagcugacccugaaguucaucugcacc\
accggcaagcugcccgugcccuggcccacccucgugaccacccugaccua\
cggcgugcagugcuucagccgcuaccccgaccacaugaagcagcacgacu\
ucuucaaguccgccaugcccgaaggcuacguccaggagcgcaccaucuuc\
uucaaggacgacggcaacuacaag


source RFP = \
ccugcaggacggcgaguucaucuacaaggugaagcugcgcggcaccaacu\
uccccuccgacggccccguaaugcagaagaagaccaugggcugggaggcc\
uccuccgagcggauguaccccgaggacggcgcccugaagggcgagaucaa\
gcagaggcugaagcugaaggacggcggccacuacgacgcugaggucaaga\
ccaccuacaaggccaagaagcccgugcagcugcccggcgccuacaacguc\
```

```
aacaucaaguuggacaucaccucccacaacgaggacuacaccaucgugga\
acaguacgaacgcgccgagggccgccacuccaccggcggcauggacgagc\
uguacaaguaa

# define window in terms of domains
window X = a b*
window Y = c* e

# constrain window to be drawn from source
X.source = GFP
# OR constrain window to be drawn from more than once source
Y.source = GFP, RFP
```

- **Library constraint.** A library constraint is specified in two steps. First, specify a library of alternative sequences of uniform length. Second, constrain a sequence domain to be drawn from one or more libraries.

```
# define a library of sequences
library toeholds = CAGUGG, AGCUCG, CAGGGC

# define a library of codons for each amino acid
library aaI = AUU, AUC, AUA
library aaL = CUU, CUC, CUA, CUG, UUA, UUG
library aaV = GUU, GUC, GUA, GUG
library aaF = UUU, UUC
library aaM = AUG
library aaC = UGU, UGC
library aaA = GCU, GCC, GCA, GCG
library aaG = GGU, GGC, GGA, GGG
library aaP = CCU, CCC, CCA, CCG
library aaT = ACU, ACC, ACA, ACG
library aaS = UCU, UCC, UCA, UCG, AGU, AGC
library aaY = UAU, UAC
library aaW = UGG
library aaQ = CAA, CAG
library aaN = AAU, AAC
library aaH = CAU, CAC
library aaE = GAA, GAG
library aaD = GAU, GAC
library aaK = AAA, AAG
library aaR = CGU, CGC, CGA, CGG, AGA, AGG
library aaSTOP = UAA, UAG, UGA

# domain a is drawn from the 'toeholds' library
a.libseq = toeholds     # domain 'a' has 6 nt

# domain b is drawn from a concatenation of library sequences
# representing codons
b.libseq = aaI aaM aaC aaG     # domain 'b' has 12 nt
```

Pattern prevention. A pattern prevention constraint can be specified for a domain, a strand, a list of domains and strands, or globally:

```
domain a = N12
domain b = N12
strand A = a a*
strand B = b b*

# pattern prevention for a domain
prevent a = AAAA, UUUU

# pattern prevention for a strand
prevent B = AAAA, UUUU

# preventing the same patterns for strand 'A' and domain 'b'
prevent A, b = AAAAA,CCCCC,GGGGG,UUUUU

# global pattern prevention
prevent = AAAA,CCCC,GGGG,UUUU,MMMMMM,KKKKKK,WWWWWW,SSSSSS,RRRRRR,YYYYYY
prevent = A4,C4,G4,U4,M6,K6,W6,S6,R6,Y6 # equivalent specification
```

Specify the stop condition. The multistate test tube design algorithm seeks to reduce the multistate test tube ensemble defect, $\mathcal{M}$, representing the average equilibrium fraction of incorrectly paired nucleotides over the design ensemble, below a stop condition specified in fraction (default) or percent format (default value shown):

```
stop = 0.05             # in internal (0,1)
stop[frac] = 0.05       # optional specification of fraction format
stop[%] = 5             # in interval (0,100)
```

Defect weights. The user may wish to alter the relative weighting of defect contributions within the design objective function, $\mathcal{M}$, to prioritize or deprioritize design quality for a portion of the design ensemble. Custom defect weights can be defined for any level within the design ensemble (tube, complex, strand, domain), or for any combination of levels (specified coarser to finer with a period separating each level). Each weight takes a value in the interval $[0, \infty)$. By default, all weights are unity. Increasing the weight for a tube, complex, strand or domain will lead to a corresponding increase in the allocation of effort to designing this entity, typically leading to a corresponding reduction in the defect contribution of the entity. Likewise, decreasing the weight for a tube, complex, strand or domain will lead to a corresponding decrease in the allocation of effort to designing this entity, typically leading to a corresponding increase in the defect contribution of the entity. Weights specified at multiple levels within the ensemble are multiplicative (see Supplementary Information of (Wolfe et al., 2017) for details). With the default value of unity for all weights, $\mathcal{M}$ reduces to the multistate test tube ensemble defect, representing the average equilibrium fraction of incorrectly paired nucleotides over the design ensemble. With custom weights, the physical meaning of the objective function is distorted in the service of adjusting design priorities. The following script illustrates assignment of defect weights at different levels within the design ensemble:

```
# domains
domain a = N5
domain b = N5
domain c = N5
domain d = N5
```

```
# strands
strand A = a b
strand B = b c
strand C = c d
strand D = d a

# complexes
complex S1 = A B
complex S2 = B C
complex S3 = C D
complex S4 = D A

S1.structure = D20 +
S2.structure = D10 (U10+U10)
S3.structure = D20 +
S4.structure = D5 (U10 D5 + U10)

# tubes
tube T1 = S1 S2
tube T2 = S3 S4

T1.offtargets = {maxsize = 2}
T2.offtargets = {maxsize = 2}

T1.S1.conc[nM] = 1.0
T1.S2.conc[nM] = 1.0
T2.S3.conc[nM] = 1.0
T2.S4.conc[nM] = 1.0

# weights specified for a single granularity level
a.weight = 2 # weight for domain 'a'
S3.weight = 4 # weight for complex 'S3'
T2.weight = 0.5 # weight for tube 'T2'

# weights for combinations of adjacent granularity levels
T1.S1.weight = 5 # weight complex 'S1' in tube 'T1'
A.b.weight = 0.75 # weight for domain 'b' in strand 'A'
T2.S4.D.a.weight = 0.5 # weight for domain 'a' in strand 'D'
                       # in complex 'S4' in tube 'T2'

# weights for nonadjacent granularity levels
T2.d.weight = 3 # weight for domain 'd' in tube 'T2'
T2.C.weight = 3 # weight for strand 'C' in tube 'T2'
S4.b.weight = 0.1 # weight for domain 'b' in complex 'S4'

# weights for nonexistent granularity combinations are ignored
# T1.S3.weight = 0.5 # complex S3 is not present in tube T1
# D.b.weight = 10 # domain b is not present in strand D
```

Algorithm parameters. Algorithm parameters for constrained multistate test tube design are set as follows (defaults shown):

```
# algorithm parameters: see Supp Info of (Wolfe et al., 2017) for details
hsplit = 2              # default: 2 for rna, 3 for dna and custom
nsplit = 12             #
fsplit = 0.99           # in interval (0,1)
fstringent = 0.99       # in interval (0,1)
dgclamp = -20.0         # kcal/mol
mbad = 300              #
mreseed = 50            #
mreopt = 3              #
fpassive = 0.01         # in interval (0,1)
fredecomp = 0.03        # in interval (0,1)
frefocus = 0.03         # in interval (0,1)
allowwobble = false     # allow algorithm to introduce wobble pairs
maxopttime = 86000000   # seconds
pairscutoff = 0         # cutoff for pair probabilities output
```

4 Getting Started

4.1 Compilation and installation

- Required packages

  The following package must be installed to compile NUPACK3.1:

  > CMake (version 2.6.0+) - A cross-platform open-source build system

  This can be installed with a standard package manager (e.g., Homebrew) or downloaded from the software developer's website. Note: *You may need root privileges to perform default installations (e.g., precede the installation command with* `sudo` *and provide a password when prompted).*

- NUPACK root directory

  Unpack the file `nupack.tar.gz` and place the `nupack` root directory in a convenient location (e.g., `/usr/local/nupack` or `$HOME/nupack`). Set the environment variable `NUPACKHOME` to specify an absolute path to the `nupack` root directory, for example:

  ```
  export NUPACKHOME=/usr/local/nupack
  ```

- Compiling and installing NUPACK

  To compile NUPACK, type the following commands from within the `nupack` root directory:

  ```
  mkdir build
  cd build
  cmake ../
  make
  ```

  To install NUPACK, then type the following command:

  ```
  make install
  ```

  Note: *You may need root privileges to perform a default installation (e.g., type* `sudo make install` *and provide a password when prompted).* The default installation will put executables in `/usr/local/bin`, libraries in `/usr/local/lib`, headers in `/usr/local/include`, and parameters in `/usr/local/share`. You are now ready to run NUPACK executables. To start, you may want to run some of the examples in `$NUPACKHOME/doc/examples`.

- Custom builds

  If you wish to compile multiple versions of NUPACK, simply make additional build directories in `$NUPACKHOME`. Various build configuration options can be passed to `cmake`. For example, to specify different compilers, replace the default call to `cmake` with:

  ```
  cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_CXX_COMPILER=g++ ../
  ```

  See CMake documentation for configuration options.

- Custom installation

  To install NUPACK in a custom location, replace the default call to `cmake` with:

  ```
  cmake -DCMAKE_INSTALL_PREFIX=prefix ../
  ```

The install command will then put executables in prefix/bin, libraries in prefix/lib, headers in prefix/include, and parameters in prefix/share. For custom installations, set the NUPACKINSTALL environment variable:

```
export NUPACKINSTALL=prefix
```

so that NUPACK executables can find NUPACK parameter files. Also, you must add NUPACKINSTALL/bin to your path:

```
export PATH=$PATH:$NUPACKINSTALL/bin
```

so that the NUPACK executables can be run without specifying an absolute path.

4.2 Examples

Following the organization of the NUPACK3.2 User Guide, sample input and output files are provided in $NUPACKHOME/doc/examples with the following directory structure:

```
complex-analysis
   simple, advanced, pseudoknot
     runjobs, input, output, output.ref
tube-analysis
   simple, advanced
     runjobs, input, output, output.ref
complex-design
   simple, advanced
     runjobs, input, output, output.ref
tube-design
   simple, advanced
     runjobs, input, output, output.ref
multitube-design
   simple, advanced
     runjobs, input, output, output.ref
```

Five subdirectories (complex-analysis, tube-analysis, complex-design, tube-design, multitube-design) correspond to different problem classes. Within each of these, three subdirectories (simple, advanced, pseudoknot) or two subdirectories (simple, advanced) correspond to different sample calculations. Within each of these, there is a shell script (runjobs) and three subdirectories containing job files (input, output, output.ref).

Run the shell script runjobs to call multiple NUPACK executables, reading input files from the directory input and writing output files to the directory output. The generated results in directory output can be compared to the reference results in directory output.ref to check that your local installation of NUPACK3.2 is running properly.

Alternatively, in directory $NUPACKHOME/doc/examples, run the script runall to run all of the runjobs scripts (a total of 11 runjobs scripts). Then run the script diffall to compare the files in the output directories to the corresponding files in the output.ref directories.

5 Acknowledgments

We thank all the NUPACK users that have helped out as beta testers over the years, as well as the many NUPACK users that have emailed `support@nupack.org` to request features or report bugs. NUPACK is supported by the National Science Foundation via the Molecular Programming Project (NSF-CCF-1317694), by the Gordon and Betty Moore Foundation (GBMF2809), and by the Beckman Institute at Caltech (PMTC). NUPACK has previously been supported by the National Science Foundation (NSF-CCF-0832824, NSF-CHE-0533064, NSF-DMS-0506468, NSF-CCF-CAREER-0448835), by the John Simon Guggenheim Memorial Foundation, by the National Institutes of Health (P50 HG004071), by the Ralph M. Parsons Foundation, and by the Charles Lee Powell Foundation.

6 References

R. M. Dirks and N. A. Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *J. Comput. Chem.*, 24:1664–1677, 2003. (pdf)

R. M. Dirks and N. A. Pierce. An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots. *J. Comput. Chem.*, 25:1295–1304, 2004. (pdf)

R. M. Dirks, M. Lin, E. Winfree, and N. A. Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Res.*, 32(4):1392–1403, 2004. (pdf)

R. M. Dirks, J. S. Bois, J. M. Schaeffer, E. Winfree, and N. A. Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM Rev.*, 49(1):65–88, 2007. (pdf)

L. M. Hochrein, M. Schwarzkopf, M. Shahgholi, P. Yin, and N. A. Pierce. Conditional Dicer substrate formation via shape and sequence transduction with small conditional RNAs. *J. Am. Chem. Soc.*, 135(46):17322–17330, 2013.

R. T. Koehler and N. Peyret. Thermodynamic properties of DNA sequences: Characteristic values for the human genome. *Bioinformatics*, 21(16):3333–3339, 2005.

D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.*, 288:911–940, 1999.

N. Saini, Y. Zhang, K. Usdin, and K. S. Lobachev. When secondary comes first – the importance of non-canonical DNA structures. *Biochimie*, 95(2):117–123, 2013.

J. SantaLucia, Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. USA*, 95(4):1460–1465, 1998.

J. SantaLucia, Jr. and D. Hicks. The thermodynamics of DNA structural motifs. *Annu. Rev. Biophys. Biomol. Struct.*, 33:415–440, 2004.

N.C. Seeman. Nucleic acid junctions and lattices. *J. Theor. Biol.*, 99:237–247, 1982.

M. J. Serra and D. H. Turner. Predicting thermodynamic properties of RNA. *Methods Enzymol.*, 259:242–261, 1995.

B. R. Wolfe and N. A. Pierce. Nucleic acid sequence design for a test tube of interacting nucleic acid strands. *ACS Synth. Biol.*, 4:1086–1100, 2015. (pdf)

B. R. Wolfe, N. J. Porubsky, J. N. Zadeh, R. M. Dirks, and N. A. Pierce. Constrained multistate sequence design for nucleic acid reaction pathway engineering. *J. Am. Chem. Soc.*, 139:3134–3144, 2017. (pdf)

J. N. Zadeh. *Algorithms for Nucleic Acid Sequence Design*. PhD thesis, California Institute of Technology, 2010.

J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *J. Comput. Chem.*, 32(1):170–173, 2011a. (pdf)

J. N. Zadeh, B. R. Wolfe, and N. A. Pierce. Nucleic acid sequence design via efficient ensemble defect optimization. *J. Comput. Chem.*, 32:439–452, 2011b. (pdf)

M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, 31(13): 3406–3415, 2003.