CANADA
The Manual

Udo Feldkamp
Universität Dortmund
Chemistry Dept. / BCMT
44221 Dortmund
Germany

April 16, 2009

Contents

Chapter 1

Introduction

1.1 CANADA and this Manual

CANADA stands for COMPUTER-AIDED NUCLEIC ACID DESIGN PACKAGE. It is a collection of tools not only for designing nucleic acid sequences with certain properties, but also for analyzing and handling them.

The design of nucleic acid sequences is necessary for a wide range of applications, most of which belong to the fields of DNA-Computing or DNA-based nanotechnology [Feldkamp et al., 2003, Feldkamp and Niemeyer, 2006]. The most important requirement for nucleic acid molecules is specific hybridization, i.e. avoidance of any unintended and thus undesired binding between strands. Furthermore, a high and homogeneous hybridization efficiency of all oligonucleotides and their intended binding partner is desirable. And, depending on the application, additional restrictions, e.g. on the molecules' melting temperature, or the specification of fixed subsequences etc., may be necessary. This multitude of different requirements is joined by the huge search space of nucleic acid sequences, containing 4^n different sequences of length n, and if you regard sets of sequences, combinatorics makes things even worse. Thus, the aid of computer programs is indispensable. The design tools in CANADA have been developed to enable the user to find sequences that meet all desired requirements and restrictions. The other tools, DNA handling ones and others, were developed when necessary during the research on DNA sequence design, and may be useful to people working with DNA.

This manual only describes how to use the tools, i.e. it mainly specifies input and output of the tools. It does not describe algorithmic details. Please refer to [Feldkamp, 2000, Feldkamp et al., 2003] for more information on how the design tools work.

CANADA is written in C++. For the time being, all binaries are for the Windows command line. Future plans include compiling Linux binaries, publishing the source code of at least some of the tools and the DeLaNA parser with the object classes, polishing and extending the tools, adding some more small tools, and maybe even a GUI or two.

1.2 Tools in CANADA

This manual describes the tools in CANADA, version 2.0. This version of the package contains the following tools:

Tool	Version	Short description
		design tools
dsg	2.01	DNA sequence generator
dsc	3.09	DNA sequence compiler
		DNA tools
align	1.0	global alignment & duplex stability
clean_out	1.0	removes non-base characters
complement	1.0	puts out Watson-Crick complements
duplex2hairpin	1.0	links two sequences of a duplex to a hairpin loop
eval_pool	1.0	evaluates the hybridization specificity of a sequence pool
gc	1.0	calculates GC content
nb_unique	1.03	searches for non-unique subsequences
rand_seqs	1.01	generates random sequences
reverse	1.0	puts out reverse sequences
seq_dist	1.01	calculates distance between sequence pairs
seq_dist2	1.01	as seq_dist, but for all possible pairs
thermo	1.01	calculates thermodynamic properties such as melting temperature or free energy
		other tools
corr_coeff	1.0	calculates linear correlation coefficients
rank_corr	1.0	calculates Spearman rank correlation coefficients

1.3 Files installed with CANADA

bin/
- align.exe
- clean_out.exe
- complement.exe
- corr_coeff.exe
- dsc.exe
- dsg.exe
- duplex2hairpin.exe
- eval_pool.exe
- gc.exe
- nb_unique.exe
- rand_seqs.exe
- rank_corr.exe
- reverse.exe
- seq_dist.exe
- seq_dist2.exe
- thermo.exe

doc/

```
    Manual.pdf
    dsc_changelog.txt
    dsg_changelog.txt
samples/
    DeLaNa/
        all_features.dln
        dsc_4WJ.dln
        dsc_DX.dln
        dsc_RNG.dln
        dsg_big_pool.dln
        dsg_small_pool.dln
    config/
        dsc_config.cfg
    seq_pools/
        example_seqs.txt
        example_seqs_w_IDs.txt
    numbers/
        corr_example.txt
        corr_example_data_only.txt
README.txt
```

1.4 Installation

It is rather easy to install CANADA on your computer: Simply unpack the zip file in any directory that is convenient for you. In order to use the tools independend on what directory you are currently working in, you might want to add the path to CANADA's bin directory to your system's path variable. Just click on the Start button, open the Control Panel (in Settings), open the System dialog, choose the Advanced tab, click on the Environment Variables button, select PATH, click on the Edit button, add a semicolon and the full path of the bin directory, click OK, click OK again. Already opened command-line windows may not recognize this change (check with the path command), but freshly opened command shells should know the new path.

1.5 Acknowledgements

I'd like to thank Prof. Wolfgang Banzhaf for giving me the opportunity to do research on DNA Computing and nanotechnology and to develop this software. Funnily enough, during this research Prof. Banzhaf moved to Canada, shortly after I decided to call the software CANADA.

Big thanks to Hilmar Rauhe for introducing me to DNA Computing. Some of the stuff in CANADA is influenced by his ideas.

I gratefully acknowledge the support of Prof. Christof M. Niemeyer, including giving me the opportunity to examine the quality of my tools' output *in vitro*, and showing me (a computer scientist) a complete new point-of-view (that of a chemist) on messing around with DNA.

Finally, thanks to my colleagues in the computer science and the chemistry department of the

Universität Dortmund for nice collaboration, helpful discussions, and a great atmosphere.

1.6 Bugreports, Questions, etc.

If you discover a bug, have a question, want to suggest additional features, or have any praise or complaints, please send an email to udo.feldkamp@uni-dortmund.de.

Chapter 2

Sequence Design Tools

Since the main reason for the development of CANADA was DNA sequence design, the tools described in this chapter are the 'core tools'. For the time being, CANADA comprises two design tools, dsg, the DNA sequence generator, and dsc, the DNA sequence compiler. The first tool generates a pool of 'good' sequences (see below for details on what 'good' means), whereas the second tool also pays respect to concatenations of sequences and their influence on 'goodness'.

In all applications – DNA computing, DNA-based nanotechnology, microarray probe design – one major need is specific hybridization, i.e. the DNA molecules only hybridize with their intended partners, and not with any other molecule. Thus, this is also the main property enforced by the design tools. In particular, the tools generate n_b-unique sequence pools. This means that for any subsequence of length n_b that a sequence (or, in the case of the sequence compiler, also a concatenation of sequences) contains, the following holds:

- This subsequence does not occur anywhere else in the sequence pool.
- The Watson-Crick-complement of the subsequence does not occur anywhere in the pool.
- The subsequence is not self-complementary.

For more details on n_b-uniqueness and how the tools enforce this property please refer to [Feldkamp et al., 2003].

Other chemical or physical properties of the DNA molecules can also be restricted or predetermined by the user, like melting temperature, GC-ratio, free energy of hybridization, or fixed or forbidden subsequences.

Both tools take input files describing the DNA sequences and their properties. A description language for nucleic acids (DeLaNA) was developed for this task.

2.1 DeLaNA — The Description Language for Nucleic Acids

DeLaNA stands for Description Language for Nucleic Acids. It serves as a standardized format for input as well as output files read or written by CANADA's design tools. Nucleic acid sequences and sequence pools are described as objects by listing their properties. In input files, specification of an object's properties sets constraints and defines parameters for the design

tools, in output files, the actual physical and chemical properties of the generated sequences are listed.

A typical object definition reads like `OBJECTTYPE objectname { listofproperties }`, where `OBJECTTYPE` specifies the type of object defined, `objectname` is an identifier for this object instance, and `listofproperties` is a list of entries of the form `property = value;` specifying the object's properties. For example, the DeLaNA code block

```
SEQUENCE oligo1 {
  length = 20;
  GC_ratio = 0.5; }
```

describes a 20mer sequence called `oligo1` with a GC-ratio of 50 %.

There are currently four different object types: `SEQUENCE`, `SEQUENCETYPE`, `POOL`, and `DESIGNTOOL`. Their properties are described in the following sections.

The DeLaNA parser is case insensitive[1] concerning all keywords for objects and properties. For example,

```
SEQUENCE oligo1 {
  length = 20; }
```

is the same as

```
sequence oligo1 {
  LENGTH = 20; }
```

or

```
SeQuEnCe oligo1 {
  lEnGtH = 20; }
```

This case insensitivity does *not* include object identifiers. For example,

```
SEQUENCE my_oligo {length = 10;}
SEQUENCE My_Oligo {length = 10;}
SEQUENCE MY_OLIGO {length = 10;}
```

defines three different `SEQUENCE` objects.

DeLaNA was developed as an input (and output) language for a broad range of nucleic acid sequence design tools, a lot of which are not yet developed. Thus, not every design tool uses every feature DeLaNA offers. In the following description of DeLaNA, all its elements will be explained, while in the sections about the different design tools all DeLaNA features ignored by the respective tool will be listed.

Some example files are presented in appendix B.1.

[1] In this manual and the example files, I will choose the case such that DeLaNA is easy to read, at least in my eyes.

2.1.1 Comments

Comments are text blocks that are ignored by the DeLaNA parser. Their purpose is to add information to the molecule description in order to make it easier to read and comprehend, or to remember the reason for certain specifications. Comment blocks begin with `/*` and end with `*/`. If you want to insert short comments, `//` causes the parser to ignore everything from these symbols up to the end of the line. The DeLaNA code block

```
/* ==================================
Now follows the definition of my oligo!
================================== */
SEQUENCE my_oligo { // well chosen identifier
length = 20; // 10 is too short, 30 too long
/*
very important comment
*/
Tm = [40;50]; // good temperature
}
/*  This was the
definition of
my oligo! */
```

is read by the parser as

```
SEQUENCE my_oligo {
length = 20;
Tm = [40;50];
}
```

2.1.2 Data types

DeLaNA uses several data types for the specification of sequences' properties. These are integer and real numbers, ranges of both number types, Booleans, sequence masks, and identifiers.

Integer: Simply an integer number, positive or negative.

Real number: Real numbers can be written in the usual fixed point format (e.g. 1.234) or in scientific format (e.g. $1e-5$).

Range: A range of integers or reals is written in square brackets, and the bounds are seperated by a semicolon. For example, the description

```
SEQUENCE my_oligo {
Tm = [40;50]; }
```

limits the melting temperature to $40-50$ °C. Wherever a range is to be specified, the DeLaNA parser also accepts a single value. For example,

```
SEQUENCE my_oligo {
Tm = 45; }
```

is read by the parser as

```
SEQUENCE my_oligo {
Tm = [45;45]; }
```

and limits the melting temperature to exactly 45 °C.

Boolean: A property of this simple type can take the values 'true' and 'false', and usually serves to switch on or off some yes-no-switches of the design tool. For example, the code block

```
DESIGNTOOL {
ID_in_table = true; }
```

tells the design tools to add sequence identifiers to the output tables. Concerning the 'true' and 'false' keywords, the DeLaNA parser is case-insensitive, i.e. they may be written `true`, `True`, `TRUE`, or `TrUe`.

Sequence mask: A sequence mask is a string of characters delimited by quotation marks, and composed of base encoding characters, including those for degenerate bases as recommended by the IUPAC [Cornish-Bowden, 1985] (see appendix A.1). For example, the code block

```
SEQUENCE my_oligo {
seq_mask = "ACGTGGA"; }
```

defines the sequence object `my_oligo` to be exaclty the oligomer `ACGTGGA`. The code block

```
SEQUENCE my_oligo {
seq_mask = "WWWWWWW"; }
```

limits `my_oligo` to be composed of `A` and `T` only. The characters' case does not matter, e.g. `RRR` specifies the same sequence as `rrr` or `RrR`.

Identifier: This is also a string, but without any delimiting characters. An identifier starts with a letter, and continues with any number of letters, numbers, or underscores. For example, legal identifiers would be `my_oligo`, `seq1`, `seq_1`, `x`, or `HELLO_____WORLD`. Identifiers are used as names for sequence or pool objects, or for user defined sequence types (see below). None of the keywords for objects or properties may be used as identifiers. For example,

```
SEQUENCE pool {
length = 12; }
```

would be illegal since `pool` is the keyword for the POOL object type. The DeLaNA parser is case *sensitive* with respect to identifiers, so `seq1` and `SEQ1` are different identifiers.

2.1.3 The SEQUENCE object type

This object type describes nucleic acid sequences. Objects of this type are specified by a code block starting with the `SEQUENCE` keyword, followed by a list of identifiers, and finally and optionally the list of property specifications enclosed by curly brackets. The list of identifiers can be, in the most simple case, a single identifier:

```
SEQUENCE x { length = 10; }
```

defines a single sequence object `x`. A list can also be composed of several comma-seperated identifiers:

```
SEQUENCE x, y, z { length = 10; }
```

defines three sequence objects, all of which are 10 bases long. In order to define larger numbers of sequences, a row of identifiers with a common prefix and an individual suffix running through a given range of numbers can be specified using square brackets:

```
SEQUENCE x[3] { length = 10; }
```

defines three sequence objects with the identifiers `x1`, `x2` and `x3`. The range does not have to start with 1:

```
SEQUENCE x[3;5] { length = 10; }
```

defines the sequence objects `x3`, `x4` and `x5`. All identifiers have the same length, suffixes contain leading zeros if necessary:

```
SEQUENCE x[8;11] { length = 10; }
```

defines the sequence objects `x08`, `x09`, `x10`, and `x11`.

The property specification part is optional:

```
SEQUENCE my_oligo;
```

defines a sequence object named `my_oligo` whose properties are all set to their respective default value.

Another possibility to define a sequence object is to specify it as an instance of a SEQUENCE-TYPE object (see the next section for details).

The SEQUENCE object type has the following properties:

Property keyword	**Data type**	**Default**	**Short description**
NA_type	{DNA, RNA, PNA}	DNA	nucleic acid type
length	integer range	[0;0]	sequence length in bases
GC_ratio	real range	[0.0;1.0]	ratio of G and C in the sequence
Tm	real range	[0.0;100.0]	melting temperature
DG or DeltaG	real range	[-1e10;1e10]	Gibbs free energy
seq_mask	string	""	base sequence or mask
forbidden	special (see below)	""	list of forbidden subsequences

The following paragraphs give a more detailled description of these properties.

NA_type: This property specifies the backbone molecules. DeLaNA knows DNA, RNA, and PNA, some of the sequence handling tools (see section 3.1) can deal with DNA and RNA, but the design tools only generate DNA sequences (at least for the time being). Thus, the default value is DNA.

length: The number of nucleotides the sequence is composed of. In input files, it possibly makes sense to restrict the length to a range, giving an appropriate design tool some liberty. Since the default value is set to 0, this property should be specified in an input file.

GC_ratio: The number of guanines and cytosines in the sequence, divided by sequence length. This can be used as a (very) rough estimation of duplex stability, since molecules with a higher GC-ratio tend to form more stable duplexes with their complements. The default range is $[0.0; 1.0]$, which allows anything from 'no G or C at all' to 'only G and C'.

Tm: The melting temperature of a duplex composed of the sequence and its Watson-Crick-complement, in ℃. T_m can be calculated with different methods and parameters, which can be specified within a POOL object (see section 2.1.5 below). Surprisingly, the default range of T_m, $[0.0; 100.0]$, which actually reads like 'any possible temperature in aqueous solution', or 'no restriction', can be too restrictive under circumstances. A bad combination of nucleic acid sequence, T_m calculation method, and parameters, may result in an estimated temperature T_m with $T_m < 0$ ℃ or $T_m > 100$ ℃. If your design tool does not detect and handle such senseless results, it may not find sequences matching the default T_m range.

DG or DeltaG: Both keywords identify the same property: The Gibbs free energy difference ΔG of the hybridization reaction between the sequence and its Watson-Crick-complement, in kcal/mol·K. For most of the available parameter sets (see section 3.1.12), this is ΔG_{37}, which means that the reaction takes place at 37 ℃, for some parameter sets it is ΔG_{25}. Duplexes are considered stable if the duplex state is energetically more favourable than the single stranded state, i.e. if $\Delta G < 0$. The default range [-1e10;1e10] means 'somewhere between very low and very high'.

seq_mask: The base sequence of the nucleic acid molecule represented by this object, delimited by quotation marks. If the sequence mask contains degenerate bases (see appendix A.1), it can be used to restrict the choice of bases for particular positions. For example, the sequence mask `"NNNYYY"` tells the design tools that the second half of the sequence must contain only pyrimidine bases (indicated by the `Y`s). The bases can be written with both upper and lower case letters. Specifying the sequence mask automatically sets the length property to the mask's length.

forbidden: The value of this property is a list of sequences that must not occur as subsequences within the sequence. Each forbidden subsequence in the list is identified by

- a sequence mask (see the description of the seq_mask property),

- an identifier of a SEQUENCE object,
- such an identifier `Z` followed by a range of integers `[x,y]`, specifying the subsequence of `Z` running from the (`x`+1)th to the (`y`+1)th base (computer scientists like to start counting with 0).
- the keyword `complement` (or `Complement`) followed by one of the three foregoing sequence specifications enclosed in round brackets, specifying the Watson-Crick-complement of the sequence in brackets (e.g. `complement("AACTGG")` specifies the sequence `"CCAGTT"`).

This property also offers an additional kind of specification. If the user wants to extend the list of forbidden sequences specified e.g. in the definition of a prototypical SEQUENCETYPE object (see next section), this can be done by using `+=` instead of `=`. For example, the line `forbidden += "AACTGG";` adds `"AACTGG"` to the list.

2.1.4 The SEQUENCETYPE object type

The SEQUENCETYPE object definition does not describe a concrete sequence object, but a prototype for sequences. It can be used when some property settings are the same for a lot of sequences, so that you can specify this shared setting only once, instead of repeating the same line for each sequence object. The user can then create instances (i.e. proper sequence objects) of a defined type by using the prototype's identifier instead of the SEQUENCE keyword. The prototype definition must precede the instantiation, i.e. when you write `my_prototype my_oligo {...` there must already have been a definition beginning with `SEQUENCETYPE my_prototype {...` before.

The syntax of the definition of SEQUENCETYPE objects is the same as for SEQUENCE objects, including all the property specifications; only the object type keywords are different. Thus, I won't repeat the whole description of properties etc. Please refer to the preceding section for that stuff.

Here's an example:

```
SEQUENCETYPE tenmer {
length = 10; }

tenmer x;

tenmer y {
GC_ratio = 0.5; }
```

This code block defines two sequences: a simple 10-mer `x`, and a 10-mer `y` with additionally restricted GC-ratio. An identical definition without SEQUENCETYPE would be

```
SEQUENCE x {
length = 10; }

SEQUENCE y {
length = 10;
GC_ration = 0.5; }
```

The user can also overwrite property specifications to define sequences that slightly differ from the prototype:

```
SEQUENCETYPE balanced_tenmer {
length = 10;
GC_ratio = 0.5; }

balanced_tenmer z {
GC_ratio = [0.0;1.0];
Tm = [30;35]; }
```

Here, the definition of `z` replaces the GC-ratio restriction of the prototype with a melting temperature restriction. It is the same as

```
SEQUENCE z {
length = 10;
GC_ratio = [0.0;1.0];
Tm = [30;35]; }
```

or, since the GC-ratio specification uses the default values for this property,

```
SEQUENCE z {
length = 10;
Tm = [30;35]; }
```

Of course, using SEQUENCETYPE makes more sense when you have to define more sequences with identical or similar properties than shown in these micro-examples.

2.1.5 The POOL object type

This object type defines a sequence pool, i.e. a set of sequences and their common properties, like restrictions concerning sequence similarity or details on the T_m calculation method. Pool objects are specified by a code block starting with the `POOL` keyword, followed by an identifier, and finally the list of property specification enclosed by curly brackets.

The POOL object type has the following properties:

Property keyword	**Data type**	**Default**	**Short description**
sequences	identifier list	""	sequences in this pool
n_uniqueness or n_b	integer range	[1;1]	length n_b of unique subsequences
Hamming	integer range	[0;0]	Hamming distance
H_distance	integer range	[0;0]	H-distance
homology	real range	[0.0;1.0]	Homology
sample_conc	real	0.0	molar DNA strand concentration
Na_conc	real	0.0	molar Na^+ ion concentration
Mg_conc	real	0.0	molar Mg^{2+} ion concentration

formamide_conc	real	0.0	volume percent formamide concentration
Tm_method	method ID (see below)	NNUnified	T_m calculation method
salt_method	method ID (see below)	Unified	Salt correction method for T_m calculation
violation_tolerance	integer	0	number of positions with allowed n_b-uniqueness violation
forbidden	special (see below)	""	forbidden subsequences
no_shorties	boolean	false	regard sequences shorter than n_b
no_GGG	boolean	false	no more than two consecutive G's allowed
no_AUG	boolean	false	start codon AUG not allowed
no_GUG	boolean	false	start codon GUG not allowed
no_UUG	boolean	false	start codon UUG not allowed
no_fraying	boolean	false	no terminal A or T allowed
base_strand_GC	real range	[0.0;1.0]	GC-ratio of base strands

The following paragraphs give a more detailled description of these properties.

sequences: The value of this property is a comma-seperated list of sequence identifiers, specifying which previously defined SEQUENCE objects, including instances of prototypes defined with SEQUENCETYPE, are collected in the pool.

n_uniqueness or n_b: A sequence pool is said to be n_b-unique, when each subsequence of length n_b that occurs in the pool, occurs only once, and its Watson-Crick complement does not occur at all [Feldkamp et al., 2003]. As a consequence, self-complementary subsequences of length n_b may not occur. This property specifies the (minimum) unique subsequence length n_b. The smaller this value is chosen, the more specific will be hybridization of sequences of this pool with their complement. Unfortunately, smaller n_b also means fewer and shorter sequences. The default value of 1 means that each base occurs only once, thus it is strongly recommended to set this property in an input file.

Hamming: Specifying this property restricts the pairwise Hamming-distance between any two sequences in the pool. Since the default value of 0 does not make much sense, this property should be specified in input files for tools that use this distance measure. See the description of the seq_dist tool in section 3.1.10 for details on Hamming-distance.

H_distance: Specifying this property restricts the pairwise H-distance [Garzon et al., 1997] between any two sequences in the pool. Since the default value of 0 does not make much sense, this property should be specified in input files for tools that use this distance measure. See the description of the seq_dist tool in section 3.1.10 for details on H-distance.

homology: Specifying this property restricts the pairwise homology between any two sequences in the pool. A value of 1 stands for identical sequences, while sequences with a

homology of 0 have no base in common. Thus, the default value of [0.0;1.0] means no restrictions for this similarity measure. See the description of the seq_dist tool in section 3.1.10 for details on homology.

sample_conc: This property is used for melting temperature calculation. It specifies the molar concentration of nucleic acid strands in solution. See the description of the thermo tool in section 3.1.12 for details on the influence of sample concentration on T_m.

Na_conc: This property is used for melting temperature calculation. It specifies the molar concentration of Na^+ ions in solution. See the description of the thermo tool in section 3.1.12 for details on the influence of salt ion concentration on T_m.

Mg_conc: This property is used for melting temperature calculation. It specifies the molar concentration of Mg^{2+} ions in solution. See the description of the thermo tool in section 3.1.12 for details on the influence of salt ion concentration on T_m.

formamide_conc: This property is used for melting temperature calculation. It specifies the volume percent concentration of formamide in solution. See the description of the thermo tool in section 3.1.12 for details on the influence of formamide on T_m.

Tm_method: This property is used for melting temperature calculation. It specifies the method and parameter set used for calculating melting temperature T_m and Gibbs free energy ΔG of the hybridization of a sequence with its Watson-Crick-complement. Valid method identifiers are:

`Wallace`	The Wallace method [Suggs et al., 1981].
`PercentGC`	The percent GC method [Wetmur, 1997].
`NNBreslauer`	The nearest-neighbor method using the parameter set from Breslauer et al. [Breslauer et al., 1986].
`NNSugimoto`	The nearest-neighbor method using the parameter set from Sugimoto et al. [Sugimoto et al., 1996].
`NNSantaLucia`	The nearest-neighbor method using the parameter set of SantaLucia et al. [SantaLucia et al., 1996].
`NNUnified`	The nearest-neighbor method using the unified parameter set of the SantaLucia group [SantaLucia, 1998].
`NNTanaka`	The nearest-neighbor method using the parameter set of Tanaka et al. [Tanaka et al., 2004].

Please note that with `Wallace` and `PercentGC` only T_m can be calculated, but not ΔG. For more details on methods and parameter sets see the description of the thermo tool in section 3.1.12. All nearest-neighbor methods can also be identified without the `NN` at the beginning, e.g. `Breslauer` is also a valid identifier and means the same as `NNBreslauer`.

salt_method: This property is used for melting temperature calculation. It specifies the method used for adapting the calculated T_m to different salt concentrations. Valid method identifiers are:

`Wetmur`	The Wetmur method [Wetmur, 1997].
`SantaLucia`	The SantaLucia method [SantaLucia et al., 1996].
`CantorSchimmel`	The Cantor/Schimmel method [Cantor and Schimmel, 1980].
`Unified`	The method from the unified parameter set [SantaLucia, 1998].

For more details on these methods see the description of the thermo tool in section 3.1.12.

violation_tolerance: In some cases, e.g. when a sequence is concatenated to more than four different adjacent sequences, n_b-uniqueness must be violated in order to find sequences at all [Feldkamp, 2000]. This property controls the amount of violation the user wants to tolerate, measured in the number of base positions 'left' and 'right' from the concatenation site at which subsequences of length n_b may occur more than once. Such multiple occurrences are limited to the same position within the concatenation.

For example, if there are 10-mer sequences $x = x_1x_2 \dots x_{10}$, $y = y_1y_2 \dots y_{10}$, and $z = z_1z_2 \dots z_{10}$, which shall be concatenated to $x_1 \dots x_7x_8x_9x_{10}y_1y_2y_3y_4 \dots y_{10}$ and $x_1 \dots x_7x_8x_9x_{10}z_1z_2z_3z_4 \dots z_{10}$, and let $n_b = 4$ and $violation\_tolerance = 1$. Then the 4-mer $x_7x_8x_9x_{10}$ still has to be unique, as do all other 4-mer subsequences that lie completely within one of the sequences. But the 4-mer $x_8x_9x_{10}y_1$ may be identical to $x_8x_9x_{10}z_1$. On the other hand, $x_8x_9x_{10}y_1$ still has to be different from all other 4-mers at other positions. Since uniqueness-violation is only allowed to be tolerated at one position from the concatenation site, $x_9x_{10}y_1y_2$ again has to be unique. The number of tolerant positions is counted in both directions, i.e. $x_{10}y_1y_2y_3$ would be allowed to occur more than once if another sequence was concatenated at y's 5'-end.

forbidden: This property specifies sequences that must not occur as subsequences in any sequence of the pool, nor in any specified concatenation of sequences. The syntax is identical to that of the forbidden property of SEQUENCE objects (see above), only without the `+=` notation.

no_shorties: Normally, tools using n_b-uniqueness take predefined nucleic acid sequences, dissect them into n_b-tuples, and mark these as used, so that the design algorithm does not use them again. If such a predefined sequence is shorter than n_b, it is ignored, since it cannot be dissected into n_b-tuples, and multiple occurance of a sequence shorter than n_b does not violate n_b-uniqueness. If the user wants to be stricter, he can activate this flag (i.e. set this property to true), whereby all n_b-tuples containing the short predefined sequence are also marked as used. The same effect can be achieved by listing the short sequence under the forbidden property; the no_shorties property is in DeLaNA for compatibility with old programs only.

no_GGG: If this flag is set to true, no occurrence of three or more consecutive guanine bases is allowed in any sequence and in any specified concatenation of sequences. The same effect can be achieved by adding `"GGG"` to the list of forbidden sequences; this property is in DeLaNA for compatibility with old programs only.

no_AUG: If this flag is set to true, no occurrence of the start codon AUG is allowed in any sequence and in any specified concatenation of sequences. For DNA sequences, ATG is forbidden instead. The same effect can be achieved by adding `"AUG"` (or `"ATG"`) to the list of forbidden sequences; this property is in DeLaNA for compatibility with old programs only.

no_GUG: If this flag is set to true, no occurrence of the start codon GUG is allowed in any sequence and in any specified concatenation of sequences. For DNA sequences, GTG is forbidden instead. The same effect can be achieved by adding `"GUG"` (or `"GTG"`) to the list of forbidden sequences; this property is in DeLaNA for compatibility with old programs only.

no_UUG: If this flag is set to true, no occurrence of the start codon UUG is allowed in any sequence and in any specified concatenation of sequences. For DNA sequences, TTG is forbidden instead. The same effect can be achieved by adding `"UUG"` (or `"TTG"`) to the list of forbidden sequences; this property is in DeLaNA for compatibility with old programs only.

no_fraying: If this flag is set to true, the terminal bases are restricted to be G or C, since G-C base pairs are more stable than A-T base pairs, and thus fraying (or breathing) of duplexes can be decreased.

base_strand_GC: If the user not only wants to restrict the GC-ratio of the whole sequence, but also likes to have a homogeneous distribution of G and C over the sequence, the GC-ratio of n_b-tuples used by the design tool to construct n_b-unique sequences can be restricted by specifying this property. The default range is $[0.0; 1.0]$, which allows anything from 'no G or C at all' to 'only G and C'.

2.1.6 The DESIGNTOOL object type

This object collects some design tool specific parameters. There should be no more than one object of this type specified in an input file; a second one would overwrite the settings of the first one. Specification starts with the `DESIGNTOOL` keyword, followed by a list of parameter specifications enclosed in curly brackets.

Please not that from version 3.04 on, dsc can also read tool specific parameters from a configuration file (where such stuff actually belongs), which offers more parameters and, if provided, overrides the settings in the DeLaNA file.

The DESIGNTOOL object type contains the following parameters:

Property keyword	Data type	Default	Short description
random_seed	integer	0	starting point for the pseudo random number generator
ID_in_table	boolean	false	add sequence identifiers to output table
analyze_uniqueness	boolean	false	analyze n_b-uniqueness after sequence generation

random_seed: This parameter is the starting value for the pseudo random number generator used in dsg and dsc (ran4 from [Press et al., 1992]). Setting it to 0 (or omitting its specification in the input file) causes the tools to take the system time (number of seconds elapsed since midnight, January 1, 1970) for random seed. Therefore, two runs of dsg or dsc within the same second and with random seed 0 will result in the same sequences.

ID_in_table: The tools dsg and dsc both produce (among other output files) plain text files containing all generated nucleic acid sequences, one sequence per line (see sections 2.2 and 2.3 below). If this flag is set to true, an additional column in front of the sequences contains the identifiers of the according SEQUENCE objects. Both columns are tabulator-seperated.

analyze_uniqueness: Since dsc may tolerate n_b-uniqueness violations if the POOL property violation_tolerance is set to a value > 0, it might be interesting to check the amount of such violations after a run. If this flag is set to true, an additional output file is written, containing all n_b-tuples that occur more than once or together with their complement. The format of this output file is similar to that produced by the analysis tool nb_unique (see sections 2.3 and 3.1.7).

2.1.7 The CONCAT statement

This is no object type, but a simple statement informing dsc that two sequences are intended to be concatenated (e.g. during self-assembly), and that the tool must also regard n_b-tuples overlapping both sequences [Feldkamp, 2000, Feldkamp et al., 2003]. This statement begins with the keyword `CONCAT` followed by identifiers of the sequence objects. For example,

```
CONCAT oligo1, oligo2;
```

specifies the concatenation of the sequences previously defined in the SEQUENCE objects `oligo1` and `oligo2`. More precisely, the 3'-end of `oligo1` will be linked with the 5'-end of `oligo2`. Not all sequences have to appear on the same strand of a duplex or of a more complex structure. Thus, the user can use the `complement` keyword to handle such cases:

```
CONCAT oligo1, complement(oligo2);
```

desribes the concatentation of `oligo1` and the Watson-Crick-complement of `oligo2`. Of course, the keyword `complement` can also be used together with the first sequence object identifier, or with both identifiers.

One weakness of this kind of concatenation description is its limitation to two sequence objects. For example, if there are sequence objects a, b, c, d, and e, and in the *in vitro* application, concatenations a-b-c and d-b-e will occur. Then this would be described by the following DeLaNA block:

```
CONCAT a, b;
CONCAT b, c;
CONCAT d, b;
CONCAT b, e;
```

The tool reading this input has no means to know that there will be no concatenation a-b-e or d-b-c. Thus, it might regard too many possible concatenations of three or more sequences and be more restrictive than necessary for the intended application. It is planned to offer a more precise concatenation description method in future versions of DeLaNA and CANADA.

2.1.8 The 3WJ and 4WJ statements

These two statements are similar to `CONCAT`. They inform design tools like dsc that sequences are joined to a junction, so that base symmetry can be avoided around branching points in order to prevent branch migration. These statements begin with the keyword `3WJ` or `4WJ` followed by the identifiers of the sequences forming the junction's arms. For example,

```
3WJ a, b, c;
```

states that the three sequences `a`, `b` and `c` form a three-way junction. Please note that they join with their 5'-ends at the branching point. If a sequence is supposed to lie with its 3'-end at the branching point, use the `complement` keyword, like in

```
4WJ a, complement(b), c, d;
```

Since the strands forming the function are each part of two different arms, concatenations are automatically included in these statements. For example, when reading `3WJ a, b, c;` the parser pretends to have also read `CONCAT complement(a), b;`, `CONCAT complement(b),c;` and `CONCAT complement(c), a;`, so you don't have to add these three statements in the DeLaNA file.

2.2 dsg — The DNA Sequence Generator (version 2.01)

This tool generates pools of n_b-unique sequences. A sequence pool is said to be n_b-unique, when each subsequence of length n_b that occurs in the pool occurs only once, and its Watson-Crick complement does not occur at all [Feldkamp et al., 2003]. As a consequence, self-complementary subsequences of length n_b may not occur.

The user can add a lot of other restrictions on the sequences' physical or chemical properties, like melting temperature, Gibbs free energy, or GC-ratio. It is also possible to restrict the choice of bases at certain positions or fix complete subsequences, or to forbid the use of user-specified subsequences. See the description of the DeLaNA input language for an enumeration of all sequence properties that can be specified by the user. DeLaNA features not used by dsg are listed further below.

dsg maps the search for a set of n_b-unique sequences on the search for a set of vertex-disjoint paths through a graph [Feldkamp et al., 2003]. This search is, to some extend, random-driven, i.e. different seeds for the random number generator lead to different sequences. Furthermore, different random seeds may determine whether dsg is successful in finding proper sequences at all. So, if dsg should fail to generate a sequence pool for your input file, e.g. because of property specifications that are too restrictive (see below), try a few more runs with different random seeds. If you set the random seed to 0 in your input file, and you make sure that each run is started at least one second later than the previous run, dsg automatically chooses different random seeds for each run.

Since the major requirement for the molecules is hybridization specificity, a correct setting of n_b is of essential importance. The smaller n_b is chosen, the less similar are the sequences, and the lower is the danger of undesired hybridizations. But smaller n_b also means a smaller number of 'building blocks' available for sequence construction, and maybe even not enough to generate sequences of the desired length. Thus, dsg might fail to generate the desired sequence pool.

See [Feldkamp et al., 2003] for details on the relationship between n_b, number of sequences, and their length.

Adding further requirements, e.g. restriction of melting temperature or limited choice of bases, certainly decrease the chances for dsg to find a sequence set satisfying all these requirements. If dsg is not successful for a certain combination of restriction, try to generate the desired number of sequences without any further resctrictions, so that you can see whether your choice of n_b is already too strict. Furthermore, if dsg is successful without the additional restrictions, e.g. concerning melting temperature, you can have a look at the output file and see what are typical melting temperatures for DNA sequences with the given length, calculation method, and parameters. Maybe the temperature range you chose in your input file was completely unrealistic?

Please note that dsg is frozen in version 2.01 and is no longer maintained or further developed. If you want to generate a pool of sequences without concatenations, you can use dsc with the `-g c` argument. You will get a higher yield/success probabilty, and because of some code optimization, dsc with `-g c` is no longer slower than dsg (which was the case in former versions because dsc has some more conditions to check, flags to set, and variables to manage).

Usage

`dsg [-p <number>] [-t] [-e <error_file>] [-{h|?}] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-p	Protocol level (default 1)	integer $\in \{0, 1, 2, 3, 4\}$
-t	Time measuring mode	
-e	Error file	file name

Arguments explained

-h and -?: dsg writes a short description of what it does and an overview of its arguments to standard output.

-p: This argument must be followed by an integer which must lie in $\{0, 1, 2, 3, 4\}$. This value is the protocol level, which determines how much information is written to standard output during a run. Valid protocol levels and their meaning are:

0 Silent mode. No output is written to standard output.

1 Normal mode (default). Messages on success or failure of the generation process are printed as well as start and end time.

2 Chatty mode. In addition to the information printed with protocol level 1, the chosen random seed and the sequence mask that is currently processed are reported.

3 Debug level 1. Information concerning pre-/post-conditions of functions etc. are printed.

4 Debug level 2. A lot of debug information is printed, including a complete dump of the data structures containing the objects specified in the DeLaNA input file, and information

on which n_b-tuples are examined in which step of the generation algorithm, and whether they can be added to the currently constructed sequence.

Normally, users should only use protocol levels between 0 and 2. Level 4 might be helpful if one wants to know exactly where/why dsg fails to generate sequences. This argument is ignored in time measuring mode (-t switch).

-t: This switch activates the time measuring mode. This means that all output is suppressed, except one line containing a flag (0 or 1) indicating the program's success (i.e. whether it managed to generate all sequences), the number of sequences generated, the number of steps through the base strand graph needed, and the time needed, measured in milliseconds. These values are tab-delimited. The measured time does not include reading the input file or writing output files. Since all other output is suppressed, any protocol level set with the -p argument is ignored.

-e: This argument must be followed by a valid file name. If this switch is set all error messages are redirected from the console into this file.

Input

dsg takes a DeLaNA file as input, which describes the sequences to be generated and restrictions concerning their properties. Not all features of DeLaNA are used by dsg. The differences are as follows.

In SEQUENCE and SEQUENCETYPE objects:

NA_type	So far, only DNA sequences are generated.
length	dsg only generates sequences with an exactly specified length. If a range of lengths is specified in the DeLaNA file dsg chooses the upper bound as sequence length.
Tm	If the estimated melting temperature of a sequence lies below 0 °C (above 100 °C) dsg sets this result to 0 °C (100 °C). Thus, a range of [0;100] in the input file really means 'any temperature'.
seq_mask	Any subsequence of the sequence mask of length n_b that does not contain degenerate base symbols is marked as forbidden, as is the Watson-Crick-complement of such a subsequence, so that dsg does not use them in another sequence. If a sequence listed under the forbidden property of the sequence pool object occurs as a subsequence in a predefined sequence mask dsg ignores this. If the sequence mask is completely unambiguous (i.e. it does not contain any degenerate base symbols), dsg does not check this sequence for violation of constraints on T_m, ΔG, GC-ratio, or homology.
forbidden	This property is ignored, only the POOL object's forbidden property is read.

In POOL objects:

POOL	For dsg, all sequences within a DeLaNA file are in one pool. Thus, only one POOL object has to be defined. If the input file contains more than one POOL objects only the properties of the first one will be used by dsg.

sequences	Since all sequences in the input file belong to one pool it is not necessary to specify which sequence belongs to which pool. Thus, dsg ignores this property.
n_uniqueness	dsg only uses an exactly specified length of unique subsequences n_b. If a range of lengths is specified in the DeLaNA file dsg chooses the upper bound as n_b. Furthermore, the choice is limited to $n_b \in \{1, \ldots, 15\}$.
Hamming	Since dsg does not use Hamming distance this property is ignored.
H_distance	Since dsg does not use H-distance this property is ignored.
homology	dsg only uses maximum homology. If a range of lengths is specified in the DeLaNA file dsg ignores the lower bound.
violation_tolerance	Since dsg ignores concatenations it also ignores this property.

In DESIGNTOOL objects:

analyze_uniqueness	Since dsg does not allow uniqueness violations, an uniqueness analysis of the output pool is usually not necessary. If a user likes to do such an analysis the tool nb_unique can be used.

In CONCAT, 3WJ and 4WJ statements:
Since dsg does not regard concatenations of sequences or structural motifs, it ignores CONCAT, 3WJ and 4WJ statements.

See appendix B.1 for example input files.

Output

Besides the messages printed to standard output described above under the protocol level argument, dsg produces three output files, a DeLaNA file, a pool file, and a file containing only the sequences. All three files are produced only if the search for all sequences was successful.

The DeLaNA output file has the same name as the input file, but with an additional *_out* attached, and the extension is always *.dln*. For example, if the input file was named *ten_20mers.dln* then the output file is called *ten_20mers_out.dln*. This file contains the same objects as the input file, but the sequence masks of the SEQUENCE objects no longer contain degenerate bases. Instead, they contain the DNA sequences found by dsg. The sequence object properties Tm, DG, and GC_ratio are set to the values calculated for the according sequence. If the random_seed of the DESIGNTOOL object was set to 0 in the input file, it here shows the random seed actually chosen by dsg.

The pool file is called *dsg.pool* and contains the generated sequences in the pool file format for dsg version 1.01a (the old one with the GUI).

The third file is called *sequences.txt* and contains only the base sequences, one sequence per line. If the ID_in_table flag of the DESIGNTOOL object is set to true an additional column with the SEQUENCE object identifiers is printed in front of the sequences. Both columns are tabulator seperated.

2.3 dsc — The DNA Sequence Compiler (version 3.09)

This tool is not limited to generating a pool of unrelated sequences. It also pays respect to the concatenation of sequences, i.e. it also considers n_b-tuples overlapping two (or maybe more, if sequences are shorter than n_b) concatenated sequences, when enforcing n_b-uniqueness (see section 2.2 for details on n_b-uniqueness). It also allows for a controlled violation of n_b-uniqueness if this should be necessary for a successful sequence generation (see section 2.1.5 and [Feldkamp, 2000]).

See the description of dsg above for some tips on restrictions and sequence generation. Please refer to [Feldkamp, 2000, Feldkamp et al., 2003] for more details on how dsc works.

Usage

`dsc [-p <number>] [-t] [-g {1|n|b|c}] [-c <config_file>] [-e <error_file>] [-{h|?}] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-p	Protocol level (default 1)	integer $\in \{0, 1, 2, 3, 4\}$
-t	Time measuring mode	
-g	Group size (default c)	character $\in$ {1,n,b,c}
-c	Configuration file	file name
-e	Error file	file name

Arguments explained

-h and -?: dsc writes a short description of what it does and an overview of its arguments to standard output.

-p: This argument must be followed by an integer which must lie in $\{0, 1, 2, 3, 4\}$. This value is the protocol level, which determines how much information is written to standard output during a run. Valid protocol levels and their meaning are:

0 Silent mode. No output is written to standard output.

1 Normal mode (default). Messages on success or failure of the generation process are printed as well as start and end time.

2 Chatty mode. In addition to the information printed with protocol level 1, the chosen random seed and the sequences that are currently processed are reported.

3 Debug level 1. Information concerning pre-/post-conditions of functions etc. are printed.

4 Debug level 2. A lot of debug information is printed, including a complete dump of the data structures containing the objects specified in the DeLaNA input file, and information on which n_b-tuples are examined in which step of the generation algorithm, and whether they can be added to the currently constructed sequence.

Normally, users should only use protocol levels between 0 and 2. Level 4 might be helpful if one wants to know exactly where/why dsc fails to generate sequences. This argument is ignored in time measuring mode (-t switch).

-t: This switch activates the time measuring mode. This means that all output is suppressed, except one line containing a flag (0 or 1) indicating the program's success (i.e. whether it managed to generate all sequences), the number of sequences generated, the number of steps through the base strand graph needed, the time needed, measured in milliseconds, and the ratio of actually used to theoretically needed base strands. These values are tab-delimited. The measured time does not include reading the input file or writing output files. Since all other output is suppressed, any protocol level set with the -p argument is ignored. If the argument -g is set to c, the number of colors used is also printed as last-but-second value (see below).

-g: This argument was actually only implemented for some experiments concerning design strategies. Probably, only two settings are really interesting. Using this argument overrides settings of the group_size parameter in a configuration file (see description of -c below). The argument must be followed by one of the characters 1, n, b, or c. It defines the sequence group size. The sequences to be generated are divided into groups, and the sequences of one group are generated in parallel (see [Feldkamp, 2000, Feldkamp et al., 2003] for details on what that means). If this argument is set to 1, each sequence it generated seperately. If it is set to n, all sequences are generated in parallel.[2] If this switch is set to b, dsc assumes that the sequences can be arranged in a bipartite graph with sequences as vertices and concatenations as edges. The sequence identifiers in the input DeLaNA file must begin with X or Y, and CONCAT statements must contain exactly one X-sequence with one Y-sequence. All X-sequences are then generated in parallel, as well as all Y-sequences.[3] If this argument is set to c or is omitted, all sequences are arranged in a graph as described above, and a minimal coloring is approximated with the Welsh-Powell-algorithm [Welsh and Powell, 1967]. All sequences of the same color are then generated in parallel.

Generating sequences in a parallel fashion is supposed to grant the algorithm more flexibility in correcting paths that run into a dead end, while in sequential generation, all sequences found before the one currently under construction are fixed and can no longer be corrected. Thus, the probability for successfully finding sequences is supposed to be higher the more sequences are generated in parallel. Experiments have shown that this is in fact the case for several scenarios that each focus on a single structural aspect (no concatenation at all, concatenation with different partner sequences, concatenation with more than four partner sequences etc.). But they also showed that setting the -g switch to 1 can be more successful when designing real-world structural motifs like double-crossover tiles [Fu and Seeman, 1993]. The reasons behind this are not yet understood. It is recommended to try both settings, -g 1 and -g c, in order to maximize chances for successful sequence generation.

-c: This argument must be followed by a valid file name. If this switch is set, tool specific parameters are read from this file, and all settings within a DESIGNTOOL object in a DeLaNA input file are ignored. See the description of the configuration file format below. Using the -g argument overrides the setting of the group_size parameter in the configuration file.

-e: This argument must be followed by a valid file name. If this switch is set all error messages

[2] *Caveat emptor:* This does not strictly enforce n_b-uniqueness, so that there may be $n_b$mers overlapping two concatenated sequences that occur more than once.

[3] This is not very realistic and thus only interesting for my experiments.

are redirected from the console into this file.

Input

dsc takes a DeLaNA file as input, which describes the sequences to be generated and restrictions concerning their properties. Not all features of DeLaNA are used by dsc. The differences are as follows.

In SEQUENCE and SEQUENCETYPE objects:

NA_type	So far, only DNA sequences are generated.
length	dsc only generates sequences with an exactly specified length. If a range of lengths is specified in the DeLaNA file dsc chooses the upper bound as sequence length.
Tm	If the estimated melting temperature of a sequence lies below 0 °C (above 100 °C) dsc sets this result to 0 °C (100 °C). Thus, a range of [0;100] in the input file really means 'any temperature'.
seq_mask	Any subsequence of length n_b of the sequence mask that does not contain degenerate base symbols is marked as forbidden, as is the Watson-Crick-complement of such a subsequence, so that dsc does not use them in another sequence. If a sequence listed under the forbidden property of the sequence pool object occurs as a subsequence in a predefined sequence mask dsc ignores this. If the sequence mask is completely unambiguous (i.e. it does not contain any degenerate base symbols), dsc does not check this sequence for violation of constraints on T_m, ΔG, GC-ratio, or homology.
forbidden	This property is ignored, only the POOL object's forbidden property is read.

In POOL objects:

POOL	For dsc, all sequences within a DeLaNA file are in one pool. Thus, only one POOL object has to be defined. If the input file contains more than one POOL objects only the properties of the first one will be used by dsc.
sequences	Since all sequences in the input file belong to one pool it is not necessary to specify which sequence belongs to which pool. Thus, dsc ignores this property.
n_uniqueness	dsc only uses an exactly specified length of unique subsequences n_b. If a range of lengths is specified in the DeLaNA file dsc chooses the upper bound as n_b. Furthermore, the choice is limited to $n_b \in \{1, \ldots, 15\}$.
Hamming	Since dsc does not use Hamming distance this property is ignored.
H_distance	Since dsc does not use H-distance this property is ignored.
homology	dsc only uses maximum homology. If a range of lengths is specified in the DeLaNA file dsc ignores the lower bound.

See appendix B.1 for example input files.

Configuration

Each line in a configuration file has the format `parameter = value;` setting the according tool

parameter. Comment and data type specifications are the same as for DeLaNA. The following parameter can be configured:

Parameter name	Data type	Default	Short description
random_seed	integer	0	starting point for the pseudo random number generator
ID_in_table	boolean	false	add sequence identifiers to output table
analyze_uniqueness	boolean	false	analyze n_b-uniqueness after sequence generation
RTF	boolean	false	output of uniqueness analysis in rich text format
group_size	character $\in$ {1, c, n, b}	c	how many sequences are generated in parallel?
group_shuffling	boolean	false	random order for sequence groups?
successor_choice	character $\in$ {s, e, f}	s	how is n_b-uniqueness enforced when choosing successors?
SC_parameter	real	0.0	parameter for the selected successor choice method

random_seed: This parameter is the starting value for the pseudo random number generator (ran4 from [Press et al., 1992]). Setting it to 0 (or omitting its specification) causes the tool to take the system time (number of seconds elapsed since midnight, January 1, 1970) for random seed. Therefore, two runs of dsc within the same second and with random seed 0 will result in the same sequences.

ID_in_table: dsc produces (among other output files) plain text files containing all generated nucleic acid sequences, one sequence per line. If this flag is set to true, an additional column in front of the sequences contains the identifiers of the according SEQUENCE objects. Both columns are tabulator-seperated.

analyze_uniqueness: Since dsc may tolerate n_b-uniqueness violations if the POOL property violation_tolerance is set to a value > 0, it might be interesting to check the amount of such violations after a run. If this flag is set to true, an additional output file is written, containing all n_b-tuples that occur more than once or together with their complement. The format of this output file is similar to that produced by the analysis tool nb_unique (see section 3.1.7). If the RTF flag is set, this file is in rich text format, otherwise it is plain ASCII.

RTF: If this flag is set, the output file of the n_b-uniqueness analysis is in rich text format, with the uniqueness-violating base strands written in red within the complete sequences. Otherwise, the file is plain ASCII. If the analyze_uniqueness flag is not set, the RTF flag has no effect.

Seperate_Entries: In the output file of the n_b-uniqueness analysis, each base strand has as many lines as occurences in the sequences. If this flag is set, groups of lines for different base strands are seperated by an empty line in the output file.

group_size: This parameter can be set to any of the characters `1`, `c`, `n`, or `b`. They encode the same group sizes that can be selected by the `-g` argument. See the description of this argument above for details. Using the `-g` argument in the command line overrides the setting in the configuration file. In order to make your configuration file more readable, you can write `coloring` for `c` and `bipartite` for `b`.

group_shuffling: If this flag is set to true and group_size is set to `1` or `c`, the order in which the sequences or sequence groups are generated is chosen randomly. If this flag is not set (which is the default) the order is determined by BFS for group_size `1` and vertex degree for `c`. If group_size is set to `n` or `b` this flag has no effect.

successor_choice: This parameter can be set to any of the characters `s`, `e`, or `f`. You can also use the complete choice method keywords `strict`, `error_probability` and `fixed_bias`. They describe different ways (and levels of strictness) to enforce n_b-uniqueness when choosing a successor node for a path representing a sequence. Some methods need a numerical parameter which must be provided as SC_parameter in the configuration file.

- **strict:** n_b-uniqueness is strictly enforced, i.e. each subsequence of length n_b may appear only once.
- **error_probability:** When a potential successor node is already used, it is nevertheless taken into account as a successor candidate with a certain probability (SC_parameter). Please note that usually the actual probability of incorrect multiple use of the node is smaller than the given parameter, since there are also other (unused) successor candidates.
- **fixed_bias:** Used and unused nodes are both taken into account as successor candidates. The probability for any used node to be chosen equals SC_parameter times the probability for an unused node.

SC_parameter: This parameter is used by the probabilistic successor choice methods `error_probability` and `fixed_bias`, and is ignored in the `strict` case. Setting this parameter to 0.0 make both methods strict again, since mistakes are then made with probability zero. For `error_probability`, values should be between 0 and 1. A parameter of 1 means that used nodes are always treated as if they were unused. For `fixed_bias`, values can also be greater than 1, but that does not make much sense, since a parameter of 1 here already means that used and unused nodes are chosen with equal probability. A higher value would cause dsc to prefer already used nodes over unused ones.

See appendix B.1 for an example configuration file.

Output

Besides the messages printed to standard output described above under the protocol level argument, dsc produces two output files, a DeLaNA file and a file containing only the sequences. Both files are produced only if the search for all sequences was successful.

The DeLaNA output file has the same name as the input file, but with an additional *_out* attached, and the extension is always *.dln*. For example, if the input file was named *DX_tiles.dln* then the output file is called *DX_tiles_out.dln*. This file contains the same objects as the

input file, but the sequence masks of the SEQUENCE objects no longer contain degenerate bases. Instead, they contain the DNA sequences found by dsc. The sequence object properties Tm, DG, and GC_ratio are set to the values calculated for the according sequence. If the random_seed of the DESIGNTOOL object was set to 0 in the input file, it here shows the random seed actually taken by dsc.

The second file is called *sequences.txt* and contains only the base sequences, one sequence per line. If the ID_in_table flag of the DESIGNTOOL object is set to true an additional column with the SEQUENCE object identifiers is printed in front of the sequences. Both columns are tabulator seperated.

If the Analyze_Uniqueness property of the DESIGNTOOL object in the input file or the according parameter in the configuration file is set to true, another file named *ReappearingBaseStrands.txt* (or *ReappearingBaseStrands.rtf*, if the RTF flag is set) is written. This contains all base strands of the length specified in the input file that violate n_b-uniqueness by occuring more than once, occuring together with their complement, or being self-complementary. The output format is similar to that of the nb_unique tool (see section 3.1.7).

Chapter 3

Other Tools

3.1 Tools for Handling Nucleic Acid Sequences

3.1.1 align (version 1.0)

This program has two related functions: It can calculate pairwise global alignments of a pool of sequences, or search for the most stable duplex for each pair and calculate the Gibbs free energy.

In global alignment mode, each possible pair of sequences taken from the input pool is formed, and also each possible pair of an input sequence and the complement of another or the same sequence. For each pair global alignment is calculated using the Needleman-Wunsch algorithm [Needleman and Wunsch, 1970]. The scores for base matches, mismatches, and opening or extending gaps can be set with -s, -q, -o, and -e command line arguments (see below). The optimal alignment for each pair is written to the file *alignment.txt*, a table of the according scores is written to *Scores.txt* (see paragraph on output below).

In duplex mode, the same sequence pairs as described above are formed, and for each such pair the hybridization conformation with the lowest Gibbs free energy (and thus the highest stability) is calculated. To this end, a dynamic programming algorithm is used, similar to the Needleman-Wunsch algorithm for global alignments, but with thermodynamic scoring for base pairing and base stacking. This scoring uses the nearest-neighbor model with parameters taken from the unified parameter set of the SantaLucia group [SantaLucia, 1998], single-base-mismatch parameters from [Peyret et al., 1999, Allawi et al., 1997, Allawi and SantaLucia, 1998b, Allawi and SantaLucia, 1998c, Allawi and SantaLucia, 1998a], and single-base-bulge parameters from [Tanaka et al., 2004]. The base pairing with minimal free energy for each pair is written to the file *alignment.txt*, a table of the free energies is written to *DeltaG.txt* (see paragraph on output below). In duplex mode the -rna switch is ignored, since align does not (yet) contain thermodynamic parameters for RNA.

Usage

`align [-{h|?}] [-i] [-{G|D}] [-{dna|rna}] [-s x] [-q x] [-o x] [-e x] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-G	Global alignment mode (default mode)	
-D	Duplex mode	
-dna	Sequences are DNA (default)	
-rna	Sequences are RNA (ignored in duplex mode)	
-s	Score for matching bases (default 1.0)	real number
-q	Score for mismatches (default -3.0)	real number
-o	Score for gap opening (default -11.0)	real number
-e	Score for gap extension (default -1.0)	real number

Arguments explained

-h and -?: align writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells align that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below.

-G and -D: Only one of these two switches should be used since they state whether align works in global alignment mode (-G) or in duplex mode (-D). If both switches are used, the second one in the command line overrules the first one. If none of these switches is used, align runs in global alignment mode.

-dna and -rna: Only one of these two switches should be used since they state whether the sequences in the input file are DNA or RNA sequences. This is only relevant for cleaning up the sequences, i.e. removing everything that is not a proper base. See the input paragraph for more details. If both switches are used, the second one in the command line overrules the first one. If none of these switches is used, the input sequences are supposed to be DNA. Any of these switches are ignored in duplex mode (-D), in this mode the input sequences are treated as DNA.

-s, -q, -o, -e: Each of these arguments must be followed by a real number which defines the according score for the global alignment. For a more detailed description of these scores, please refer to [Gibas and Jambeck, 2001] or any other good book on bioinformatics. If some or all of these switches are not used, they are set to 1.0 (-s), -3.0 (-q), -11.0 (-o), and -1.0 (-e), respectively. These default values are taken from blastn [Gibas and Jambeck, 2001]. In duplex mode (-D) these four switches are ignored.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. If the -dna switch is used (or neither -dna nor -rna) this includes `u` and `U`; if the -rna switch is used `t` and `T` are ignored. The case of the characters does not matter.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic

acid sequence. The identifiers read in are also used in the output files (see below).

See appendix B.3 for examples of input files.

Output

In both modes align writes two output files. In global alignment mode these are *alignment.txt* and *Scores.txt*, in duplex mode these are *alignment.txt* and *DeltaG.txt*.

In global alignment mode, *alignment.txt* contains the optimal alignment for each sequence pair. Each alignment starts with a header identifying the two sequences of the pair. If -i is used, the according identifiers read from the input file are used here, otherwise align gives the sequences numbers starting with 1 for the first sequence. If the second sequence of the pair is the complement of an input sequence, this is also indicated. The next line contains the first base sequence, delimited by `5'-` and `-3'` markers. Internal gaps are marked by `-`, terminal gaps by spaces. The third line indicates base matches with `|` and mismatches with spaces. The last line shows the second sequence of the pair, also with delimiters. Two alignments are seperated by an empty line. Two example alignments are shown here:

```
3 vs. 2:
5'-ggatc---gcaggatctcagtca-3'
    |||    ||||||| ||
5'-agataacagcaggatttctt   -3'

8 vs. complement of 8:
5'-gtccttcccgcggtttctac-3'
   ||     ||||||     ||
5'-gtagaaaccgcgggaaggac-3'
```

Scores.txt contains two tables, one for the pairs consisting of two input sequences, and one for the pairs consisting of one input sequence and the complement of an input sequence. Each table is preceeded by an according header followed by an empty line. The tables are seperated by two empty lines.

Each table has a header line containing the sequence identifiers of the pairs' second sequences. Each following line of the table starts with the according sequence identifier of the pairs' first sequence. Then follow the scores. Only half of the table is filled (lower triangle), since the alignment of sequence X versus sequence Y has the same score as the alignment of Y versus X. Columns are seperated by tabulators. Such a file could look like this:

```
Global alignment scores:

        1       2       3       4       5       6       7       8
1       20
2       -48     20
3       -32     -29     20
4       -16     -44     -36     20
5       -40     -32     -32     -32     20
6       -43     -40     -40     -43     -44     20
7       -36     -32     -51     -44     -48     -52     20
8       -40     -40     -40     -44     -32     -36     -40     20
```

```
Global alignment scores vs. complement:

        1       2       3       4       5       6       7       8
1       -39
2       -42     -36
3       -46     -43     -36
4       -50     -42     -47     -36
5       -32     -42     -48     -48     -46
6       -44     -36     -24     -44     -47     -36
7       -44     -44     -40     -44     -36     -56     -36
8       -44     -40     -40     -43     -48     -40     -44     -20
```

The alignment of sequence 3 vs. sequence 2 shown above has a score of -29, the alignment of sequence 8 vs. its own complement has a score of -20. The tables usually do not look nice in a text editor, but can be easily imported into MS-Excel or another spreadsheet program.

In duplex mode, *alignment.txt* looks like described above for the alignment mode, but the '|' in the third line of an alignment now indicates a Watson-Crick base pair. Here are some examples:

```
6 vs. 3:
5'-agaccgg-gctcc-gcacctgt-3'
    |||  | | ||| ||
3'-actga-ctctaggacgctagg -5'

7 vs. complement of 7:
5'-cttcacatacaaa-attaatc-3'
   |||||||||||| | |||||||
3'-gaagtgtatgt-tttaattag-5'
```

If the gaps in the second alignment irritate you: some single base bulges are (at least in theory) more stable than a perfect helix [Tanaka et al., 2004].

DeltaG.txt looks like *Scores.txt*, for example:

```
DeltaG:

        1       2       3       4       5       6       7       8
1       -2.17
2       -2.05   -3.49
3       -4.25   -5.56   -5.45
4       -2.89   -1.45   -5.24   -4.77
5       -2.3    -11.13  -6.35   -2.3    -5.48
6       -5.2    -2.58   -6.28   -8.46   -1.45   -11.11
7       -1      -2      -1.28   -1.28   -2.3    -3      -4.34
8       -2.17   -4.42   -4.42   -4.01   -1.3    -3.86   -1.44   -1.44


DeltaG vs. complement:
```

```
        1       2       3       4       5       6       7       8
1       -28.02
2       -2.3    -26.29
3       -5.71   -4.02   -27.28
4       -3.31   -1.72   -2.14   -30.1
5       -2.3    -3.33   -5.44   -5.23   -23.96
6       -8.32   -2.58   -9.13   -4.95   -1.3    -32.56
7       -2.3    -2.74   -1.28   -6.75   -6.75   -1.28   -23.44
8       -3.63   -6.33   -7.67   -8.56   -8.17   -5.47   -3.58   -28.3
```

Examples

`align example_seqs.txt`
`align -G example_seqs.txt`
both calculate global alignments for the sequences in *example_seqs.txt*.

`align -i example_seqs_w_IDs.txt`
calculates global alignments for the sequences in *example_seqs_w_IDs.txt* which contains sequence identifiers in the first column.

`align -q -1.0 -o -5.0 example_seqs.txt`
calculates global alignments with less restricting penalties for mismatches and gap openings.

`align -i -rna -G -s 1.5 -q -1.5 -o -3.1415 -e -0.5 example_RNA_seqs.txt`
reads RNA sequences from a file with sequence identifiers and calculates global alignments using these interesting scores.

`align -D example_seqs.txt`
calculates the most stable base pairing for each duplex.

3.1.2 clean_out (version 1.0)

This program reads nucleic acid sequences, removes all whitespaces and other characters not encoding bases, and writes the clean sequences to standard output. The sequences are usually read from an input file. If no filename is given, `clean_out` works in interactive mode. In this mode the user is prompted to type in a sequence, and the clean sequence is put out immediately. You can leave the interactive mode (and exit the program) by entering `@`.

Usage

`clean_out [-{h|?}] [-i] [-{dna|rna|iupac}] [<input_file>]`

Arguments overview

Argument	Description
-h	Print help text
-?	Print help text
-i	Input file contains sequence IDs
-dna	Sequences are DNA (default)
-rna	Sequences are RNA
-iupac	Sequences contain degenerate bases

Arguments explained

-h and -?: clean_out writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells clean_out that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below. In interactive mode, this switch is ignored.

-dna, -rna, and -iupac: Only one of these three switches should be used since they state whether the sequences in the input file are DNA or RNA sequences, or whether they contain degenerate bases. If the -dna switch is used (or none of the three switches) `u` and `U` are treated like non-base characters and thus are removed; if the -rna switch is used `t` and `T` are removed. The case of the characters does not matter. If -iupac is used, the input sequences may contain degenerate bases, e.g. `S` for the strong binding bases `C` and `G`. Degenerate bases are encoded according to the IUPAC-IUB recommendations [Cornish-Bowden, 1985] (see appendix A.1). If two or all switches are used, the last one in the command line overrules the preceding ones. If none of these switches is used, the input sequences are supposed to be DNA.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the clean sequences are written to standard output, one per line.

Examples

`clean_out example_seqs.txt`
removes all character except `a`, `A`, c, `C`, `g`, `G`, `t`, and `T` from the sequences in the file *example_seqs.txt* and writes the clean sequences to standard output.

`clean_out -i -rna example_RNA_seqs_w_IDs.txt`
removes all character except `a`, `A`, c, `C`, `g`, `G`, `u`, and `U` from the sequences in the file *example_RNA_seqs_w_IDs.txt* which contains sequence identifiers and writes the clean sequences to standard output.

An interactive sequence cleaning tolerating degenerate bases could look like this:

```
>clean_out -iupac
Enter sequence (@ to exit):
acgatcgac
acgatcgac

Enter sequence (@ to exit):
abcdefghijklmnopqrstuvwxyz
abcdghkmrstuvwy
```

```
Enter sequence (@ to exit):
@

>
```

3.1.3 complement (version 1.0)

This program reads nucleic acid sequences and writes their Watson-Crick complements to standard output. The sequences are usually read from an input file. If no filename is given, complement works in interactive mode. In this mode the user is prompted to type in a sequence, and the complement is put out immediately. You can leave the interactive mode (and exit the program) by entering `@`.

Usage

`complement [-{h|?}] [-i] [-{dna|rna|iupac}] [<input_file>]`

Arguments overview

Argument	Description
-h	Print help text
-?	Print help text
-i	Input file contains sequence IDs
-dna	Sequences are DNA (default)
-rna	Sequences are RNA
-iupac	Sequences contain degenerate bases

Arguments explained

-h and -?: complement writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells complement that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below. In interactive mode, this switch is ignored.

-dna, -rna, and -iupac: Only one of these three switches should be used since they state whether the sequences in the input file are DNA or RNA sequences, or whether they contain degenerate bases. These switches determine which characters in the input sequences are treated as non-base encoding characters and are removed in a preprocessing step. They also determine whether the complementary base of adenine is thymine or uracil. If the -dna switch is used (or none of the three switches) `u` and `U` are treated like non-base characters and thus are removed; if the -rna switch is used `t` and `T` are removed. The case of the characters does not matter. If -iupac is used, the input sequences may contain degenerate bases, e.g. `S` for the strong binding bases `C` and `G`. Degenerate bases are encoded according to the IUPAC-IUP recommendations [Cornish-Bowden, 1985] (see appendix A.1). The complementary base in the output is also encoded in this way, e.g. the complement of `r` (purine) is `y` (pyrimidine). If two or all switches are used, the last one in the command line overrules the preceding ones. If none of these switches is used, the input sequences are supposed to be DNA.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. This ignorance is directed by the use of -dna, -rna, and -iupac, as described above.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the complementary sequences are written to standard output.

Examples

`complement example_seqs.txt`
reads sequences from the file *example_seqs.txt* and writes their Watson-Crick complements to standard output.

`complement -i -rna example_RNA_seqs_w_IDs.txt`
reads RNA sequences from the file *example_RNA_seqs_w_IDs.txt* which contains sequence identifiers and writes the complementary sequences to standard output.

An interactive session tolerating degenerate bases can look like this:

```
>complement -iupac
Enter sequence (@ to exit):
aaggtccc
gggacctt

Enter sequence (@ to exit):
acctttyyrsswww
wwwssyrraaaggt

Enter sequence (@ to exit):
@

>
```

3.1.4 duplex2hairpin (version 1.0)

This program reads pairs of nucleic acid sequences from an input file, connects them *via* a linker sequence and writes the resulting sequences to standard output. This is useful when you want to use a single stranded secondary structure prediction program like RNAfold [Hofacker et al., 1994] for predicting duplex formation of a pair of sequences. To this end, one connects the two sequences of interest with a linker sequence of non-pairing bases which will form a hairpin loop with the two sequences forming the stem [Ackermann and Gast, 2003]. You can also add constraints for RNAfold to the output, forcing that program to allow only base pairs between the two sequences, but not between bases of one and the same sequence.

Usage

`duplex2hairpin [-{h|?}] [-n <number>] [-i] [-c] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-n	Linker sequence length (default 16)	integer > 0
-c	Add constraint lines	

Arguments explained

-h and -?: duplex2hairpin writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells duplex2hairpin that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. See also the paragraphs on input below.

-n: This argument must be followed by a positive integer number. This is the number of non-hybridizing bases (here: `n`) that will link the two sequences of each pair. If this argument is omitted, the linker is 16 characters long. This is supposed to be long enough to suppress the effects of loop length changes due to changing numbers of unhybridized (proper) bases adjacent to the linker [Ackermann and Gast, 2003].

-c: If this switch is used, each connected sequence in the output is followed by a constrain line, telling RNAfold to allow only intermolecular base pairs, but no intramolecular ones. These constraint lines begin with a sequence of `<` as long as the first DNA sequence, continue with a series of `x` as long as the linker, and end with a sequence of `>` as long as the second DNA sequence. See the man page of RNAfold (available on the WWW) for more details on constraints.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. duplex2hairpin regards two consecutively read sequences as one pair that shall be connected.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the connected sequences are written to standard output. Each line consists of the first sequence of the connected pair, a number of linker bases (`n`) determined by the -n argument, and the second sequence. If the -c switch is used, each connected sequence is followed by a constraint line. See the description of this switch for details. See below for some output examples.

Examples

`duplex2hairpin example_seqs.txt`
reads the eight sequences from *example_seqs.txt*, builds four pairs, and writes these pairs connected through a linker sequence to standard output. The output looks like this (see appendix B.3 for input sequences):

```
ttcgccctgctactaacacgnnnnnnnnnnnnnnnnnagataacagcaggatttctt
ggatcgcaggatctcagtcannnnnnnnnnnnnnnnngtgaccctccttccagtccg
gaattccatatcccttccaannnnnnnnnnnnnnnnnnagaccgggctccgcacctgt
cttcacatacaaaattaatcnnnnnnnnnnnnnnnnngtccttcccgcggtttctac
```

`duplex2hairpin -i -n 5 example_seqs_w_IDs.txt`
reads the eight sequences from the file *example_seqs_w_IDs.txt* which contains sequence identifiers, build four pairs, and write these pairs connected through a linker sequence of length 5 to standard output. The output looks like this:

```
ttcgccctgctactaacacgnnnnnagataacagcaggatttctt
ggatcgcaggatctcagtcannnnngtgaccctccttccagtccg
gaattccatatcccttccaannnnnagaccgggctccgcacctgt
cttcacatacaaaattaatcnnnnngtccttcccgcggtttctac
```

`duplex2hairpin -c example_seqs.txt`
does the same as the first example but adds a constraint line for RNAfold after each connected sequence, resulting in the following output:

```
ttcgccctgctactaacacgnnnnnnnnnnnnnnnnnagataacagcaggatttctt
<<<<<<<<<<<<<<<<<<<<xxxxxxxxxxxxxxxx>>>>>>>>>>>>>>>>>>>>
ggatcgcaggatctcagtcannnnnnnnnnnnnnnnngtgaccctccttccagtccg
<<<<<<<<<<<<<<<<<<<<xxxxxxxxxxxxxxxx>>>>>>>>>>>>>>>>>>>>
gaattccatatcccttccaannnnnnnnnnnnnnnnnnagaccgggctccgcacctgt
<<<<<<<<<<<<<<<<<<<<xxxxxxxxxxxxxxxx>>>>>>>>>>>>>>>>>>>>
cttcacatacaaaattaatcnnnnnnnnnnnnnnnnngtccttcccgcggtttctac
<<<<<<<<<<<<<<<<<<<<xxxxxxxxxxxxxxxx>>>>>>>>>>>>>>>>>>>>
```

3.1.5 eval_pool (version 1.0)

This program reads DNA sequences from an input file (which contains one sequence in each line) and evaluates the hybridization specificity of the pool, using various different measures. Some are pairwise distance measure; the measured distances for all pairs are then aggregated using one of several differnet functions. Other measures are pool properties that simply deliver one number for the whole pool. The results of the evaluations are written to standard output in one tabulator-seperated line, with a column for each measure.

Usage

`seq_dist [-{h|?}] [-i] [-t] [-m <string> [-a <string>]] [-p <string>] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-t	Write headline for table	
-m	Distance measure selection (default Ham;Hme;Hdi;Hom;Edi)	string
-a	Aggregation function selection (default KD)	string
-p	Pool property selection (default empty)	string

Arguments explained

-h and -?: eval_pool writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells eval_pool that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. See also the paragraph on input below.

-t: This switch tells eval_pool to print a header line identifying the distance measures, aggregation functions and pool properties. See the paragraph on output below for details.

-m: This argument must be followed by a string that determines which distance measures are calculated, and with which parameters. Each measure is identified by a three-letter abbreviation, different measures are seperated by `;`, numerical parameters are seperated by `,`. The measures and their abbreviations are the same as for seq_dist, please see section 3.1.10 for details. Differing from seq_dist, here all measures except Hme and Hdi are measured between the first sequence and the complement of the second one, because the distances model the probability of hybridization. Adding a `c` to the abbriviation or omitting the `c` (like for the seq_dist tool) has no effect here. Abbriviations can be used more than once so that the same distance measure can be computed with different parameters. See the Examples paragraph below for some example strings. The default string is `Ham;Hme;Hdi;Hom;Edi`.

-a: This argument must be followed by a string that determines which function is used to aggregate all pairwise distances to one numeric value. Each function is identified by a two-letter abbreviation, different functions are seperated by `;`. Normally, the distances are aggregated by comparing the intended pairings (i.e. each sequence vs. its complement) to undesired pairings (usually each sequence vs. each other sequence). An additional `c` behind an abbreviation indicates that undesired hybridizations means those between any sequence of the pool and the complement of any other sequence. An additional `b` behind an abbreviation indicates that undesired hybridizations means both types of pairings. See the Examples paragraph below for some example strings. The default string is `KD`.

The aggregation functions and their abbreviations are:

KD tells eval_pool to aggregate the distances with the function K_D, which delivers the difference between the distance measurement for the most probable undesired pairing and the distance measurement for the least probable desired pairing.

KA tells eval_pool to aggregate the distances with the function K_A, which delivers the difference between the mean of all measurements for undesired pairings and the mean of all measurements for desired pairings.

-p: This argument must be followed by a string that determines which pool properties are calculated, and with which parameters. Each property is identified by a three-letter abbreviation, different measures are seperated by `;`, numerical parameters are seperated by `,`. Adding a `c` or `b` to an abbreviation has the same effect as described for **-a** above. See the Examples paragraph below for some example strings. The default string is the empty string, stating that no pool property shall be calculated.

The pool properties, their abbreviations and the parameters are:

Unq measures the uniqueness of tuples of a fixed length L. The number of unique L-tuples appearing in the pool is counted and compared to the maximum number that can appear. The result is $1 - unique/maximum$. Normally, a L-tuple is unique when it only appears once in the whole pool. If it appears twice (or more times) in the pool only one appearance is counted. When using `Unqc`, a L-tuple is unique when its complement does not appear in the pool. Otherwise, the tuple and its complement are counted only as one appearance. When using `Unqb`, a L-tuple is unique if neither itself nor its complement appear anywhere else in the pool.

LZ5 measures the Lempel-Ziv-complexity of the pool. This is similar to the LZ-complexity of a sequence [Lempel and Ziv, 1976], expanded to a set of sequences. It is actually the sum of the LZ-complexities of the sequences in the set, but with some minor changes. If the algorithm reaches the end of one sequence and starts calculating the complexity of the next sequence in the set, it keeps the vocabulary of subwords found so far. Furthermore, since the exact value of the LZ-complexity depends on the order in which the sequences are processed, they are sorted lexicographically beforehand in order to get only one value for a sequence set, independent of the sequence order in the input file. And finally, the LZ-algorithm always counts the last subword of a sequence, even if it is already in the vocabulary (such a subword is called a non-exhaustive component in [Lempel and Ziv, 1976]). While this is okay for a single sequence, it can lead to counting more non-exhaustive than exhaustive components when calculating the complexity of a pool. Therefore, non-exhaustive components are never counted here.

LZ6 measures the Lempel-Ziv-complexity of the pool as described above for LZ5, but only subwords with a minimum length (which is given as a numerical parameter) are added to the vocabulary. This may model hybridization better than simple LZ-complexity, since in duplex nucleation, three or more consecutive base pairs are needed to form a stable duplex [Cantor and Schimmel, 1980].

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the results are written to standard output. If the `-t` switch is used, the first line of the table contains the column headers, indicating the distance measures calculated and aggregation function, or the pool property. The second line (or the only one if `-t` is not used) contains the aggregated measures and pool properties. The columns are seperated by tabulators. If one evaluates several pools, one can use `-t` to generate a table header as the first line of an output file (redirecting the output from standard output into that file) when calling eval_pool for the first pool, and then append the results for the following pools omitting `-t`. See below for some example outputs.

Examples

`eval_pool example_seqs.txt`
reads the eight sequences contained in the file *example_seqs.txt*, and calculates for each pair (X, Y) the Hamming distance between X and $\overline{Y}$, H-measure between X and Y, H-distance between the poligos containing X and Y, Homology between X and $\overline{Y}$, and Edit-distance between X and $\overline{Y}$. Then, the measures are aggregated using K_D. The output looks like this:

```
10      10      0       0.5     10
```

`eval_pool -i -t example_seqs_w_IDs.txt`
does the same, but reads also sequence identifiers from the input file, and adds a header line to the output:

```
K_D^n(Ham_c)    K_D^n(Hme)      K_D^n(Hdi)      K_D^n(Hom_c)    K_D^n(Edi_c)
10      10      0       0.5     10
```

`eval_pool -t -m Ham -a KD;KA example_seqs.txt`
calculates the Hamming distances between X and $\overline{Y}$ for all sequence pairs (X, Y), aggregates them with both functions K_D and K_A, and outputs both results:

```
K_D^n(Ham_c)    K_A^n(Ham_c)
10      15.9063
```

`eval_pool -t -m Ham -a KDc;KAc example_seqs.txt`
does the same, but treats hybridizations between pool sequences and the complements of other pool sequences as undesired:

```
K_D^c(Ham_c)    K_A^c(Ham_c)
9       14.8571
```

`eval_pool -p Unq;LZ6,3 example_seqs.txt`
calculates the uniqueness pool property and the LZ-complexity of the pool, adding only subwords of minimum length 3 to the vocabulary.

`eval_pool -p LZ6,3;LZ6,4 example_seqs.txt`
calculates two LZ-complexities of the pool, adding only subwords of minimum length 3 or 4 to the vocabulary, respectively.

3.1.6 gc (version 1.0)

This program reads DNA or RNA sequences and writes their length (in bases), absolute GC content, and GC ratio (i.e. GC content devided by sequence length) to standard output.

The sequences are usually read from an input file. If no filename is given, `gc` works in interactive mode. In this mode the user is prompted to type in a sequence, and the calculated data are put out immediately. You can leave the interactive mode (and exit the program) by entering `@`.

Usage

`gc [-{h|?}] [-i] [<input_file>]`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	

Arguments explained

-h and -?: `gc` writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells `gc` that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below. In interactive mode, this switch is ignored.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the calculated data (length, number of G and C bases, GC ratio) are written to standard output.

The output format is the same in interactive mode (when no input file is given). See the next paragraph for some example outputs.

Examples

`gc example_seqs.txt`
reads sequences from the file *example_seqs.txt*, calculates length, absolute GC content and GC ratio and writes the results to standard output. The output starts like this:

```
sequence length number_GC GC_ratio
gtacttccttaaacgacgcagg 22 11 0.5
ggcggtaaaagaatcttggctg 22 11 0.5
catatctcggcacacatgatgg 22 11 0.5
...
```

```
gc -i example_seqs_w_IDs.txt
```
also reads sequence identifiers from the input file:

```
ID sequence length number_GC GC_ratio
eins gtacttccttaaacgacgcagg 22 11 0.5
Karl ggcggtaaaagaatcttggctg 22 11 0.5
23 catatctcggcacacatgatgg 22 11 0.5
...
```

An interactive session can look like this:

```
>gc
Enter sequence (@ to exit):
aaaaaaaaaaa
sequence        length  number_GC       GC_ratio
aaaaaaaaaaa     11      0       0

Enter sequence (@ to exit):
gcgcgcgcgcg
sequence        length  number_GC       GC_ratio
gcgcgcgcgcg     11      11      1

Enter sequence (@ to exit):
acgugcaugaccgauca
sequence        length  number_GC       GC_ratio
acggcagaccgaca  14      9       0.642857

Enter sequence (@ to exit):
@

>
```

3.1.7 nb_unique (version 1.03)

This program reads a pool of DNA sequences from an input file, analyzes the frequency of subsequences of a given length, and reports non-unique subsequences to standard output. A sequence pool is said to be n_b-unique, when each subsequence of length n_b that occurs in the pool, occurs only once, and its Watson-Crick complement does not occur at all [Feldkamp et al., 2003]. As a consequence, self-complementary subsequences may not occur. nb_unique finds and reports each subsequence of length n_b within a given range violating this uniqueness property by occuring more than once (complement included) or being self-complementary. This uniqueness concept is also used in the sequence design tools.

Usage

`nb_unique [-{h|?}] [-l <number>] [-u <number>] [-i] [-r] [-s] [-d] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-r	Output file in rich text format	
-s	Seperate entries of different base strands	
-d	Input file is in DeLaNA format	
-l	Lower bound for subsequence length (default 4)	integer $\in \{1, \ldots, 15\}$
-u	Upper bound for subsequence length (default 10)	integer $\in \{1, \ldots, 15\}$

Arguments explained

-h and -?: nb_unique writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells nb_unique that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in output. See also the paragraphs on input below.

-r: If this flag is set, the output is in rich text format, with the uniqueness-violating subsequences written in red within the complete sequences. Therefore it makes sense to redirect the output into a RTF file. If this flag is not set (which is default) the output is plain ASCII.

-s: In the output, each subsequence has as many lines as occurences in the input sequences. If this flag is set, groups of lines for different subsequences are seperated by an empty line in the output.

-d: If this flag is set, the input file is expected to be in DeLaNA format (described in section 2.1). The arguments -i and -u are ignored. If the -l argument is provided, this value is taken as subsequence length. Otherwise, subsequence length is as stated within the DeLaNA file. This mode is particularly useful, when the user wants to analyze concatenations of sequences.

-l and -u: These arguments must be followed by positive integer numbers less than 16. These numbers are the lower and upper bound for subsequence length. For each subsequence length between these bounds (both bounds included) nb_unique runs an uniqueness analysis. If both bounds are equal exactly one analysis is done. If -l is omitted the lower bound is set to 4, if -u is omitted the upper bound is set to 10.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence. If the -d switch is used, input is expected to be in DeLaNA format, which is described in section 2.1.

See appendix B.3 for examples of input files.

Output

No files are written, the results of the analysis are written to standard output. The output is a table with seven columns. The first line contains the column headers, each following line shows information on one non-unique subsequence. The columns are tabulator-seperated. This does not always look nice in a text editor, but is useful for easy import in MS-Excel or other spreadsheet programs. The seven entries of a line contain:

- the subsequence length currently examined
- the non-unique subseqence found
- the number of times this subsequence and its Watson-Crick complement occur in the pool.
- This entry identifies the sequence in which the subsequence occurs. If the -i switch is used, this entry contains the identifier read from the input file. Otherwise, the sequences in the input file are consecutively numbered (starting with 0), and the fourth entry contains this number.
- the starting position of the subsequence in the input sequence (starting with 0)
- the base sequence of the input sequence in which the subsequence occurs.
- the word 'yes' or 'no', depending on whether the subsequence is self-complementary or not.

See below for some output examples.

If the -r switch is used, the output is in rich text format, with the uniqueness-violating base strands written in red within the complete sequences, and should be redirected into a RTF file. If the -s switch is used, lines dealing with different base strands are seperated by an empty line.

Examples

`nb_unique example_seqs.txt`
analyses the n_b-uniqueness of the eight sequences from *example_seqs.txt*, with n_b running from 4 to 10, and writes subsequences violating this uniqueness to standard output. The output starts like this (see appendix B.3 for input sequences):

```
Length Base strand Frequency Seq-No Pos Sequence self-comp
4 tcgc 2 0 1 ttcgccctgctactaacacg no
4 tcgc 2 2 3 ggatcgcaggatctcagtca no
4 gccc 2 0 3 ttcgccctgctactaacacg no
4 gggc 2 5 5 agaccgggctccgcacctgt no
...
```

This means that the 4mer subsequence `tcgc` occurs twice within the input pool, once starting at the second base of the first sequence, and once at the fourth base of the third sequence (numbering of sequence and positioning starts with 0). The subsequence `gccc` occurs only once, but its complement `gggc` is also present, thus violating 4-uniqueness. None of these subsequences is self-complementary. The output ends with

```
...
7 ccttcca 2 3 8 gtgaccctccttccagtccg no
7 ccttcca 2 4 12 gaattccatatcccttccaa no
```

which not only indicates that this subsequence occurs twice. Since the analysis continued up to $n_b = 10$ but the last output line is for $n_b = 7$, one can conclude that no subsequence of length 8 (or longer) violates n_b-uniqueness. Thus, the input pool is 8-unique.

`nb_unique -s example_seqs.txt`
produces similar output, only a little bit easier to read:

```
Length Base strand Frequency Seq-No Pos Sequence self-comp

4 tcgc 2 0 1 ttcgccctgctactaacacg no
4 tcgc 2 2 3 ggatcgcaggatctcagtca no

4 gccc 2 0 3 ttcgccctgctactaacacg no
4 gggc 2 5 5 agaccgggctccgcacctgt no

...
```

`nb_unique -i -l 6 -u 6 example_seqs_w_IDs.txt`
analyzes the input sequences for $n_b = 6$ only, reading and using also the sequence identifiers from the input file. The output looks like this:

```
Length Base strand Frequency Seq-No Pos Sequence self-comp
6 cctgct 2 Karl 5 ttcgccctgctactaacacg no
6 agcagg 2 Willy 7 agataacagcaggatttctt no
6 gcagga 2 Willy 8 agataacagcaggatttctt no
6 gcagga 2 seq03 5 ggatcgcaggatctcagtca no
6 caggat 2 Willy 9 agataacagcaggatttctt no
6 caggat 2 seq03 6 ggatcgcaggatctcagtca no
6 tccttc 2 xyz1 7 gtgaccctccttccagtccg no
6 tccttc 2 last_seq 1 gtccttcccgcggtttctac no
6 ccttcc 3 xyz1 8 gtgaccctccttccagtccg no
6 ccttcc 3 dog 12 gaattccatatcccttccaa no
6 ccttcc 3 last_seq 2 gtccttcccgcggtttctac no
6 cttcca 2 xyz1 9 gtgaccctccttccagtccg no
6 cttcca 2 dog 13 gaattccatatcccttccaa no
6 gaattc 1 dog 0 gaattccatatcccttccaa yes
6 attaat 1 Igor 13 cttcacatacaaaattaatc yes
6 ccgcgg 1 last_seq 7 gtccttcccgcggtttctac yes
```

The last three 6mers are self-complementary.

3.1.8 rand_seqs (version 1.01)

This program generates a pool of DNA sequences randomly. At each base within the sequences, the probability for each of the four bases to be chosen is 0.25. As pseudo random number generator the ran4 routine from [Press et al., 1992] is used.

Usage

`rand_seqs [-{h|?}] [-n <number>] [-l <number> [-u <number>]] [-s <number>] [-o <output_file>] [-i]`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-n	Number of sequences (default 100)	integer ≥ 0
-l	Sequence length (or lower bound, default 20)	integer ≥ 0
-u	Upper bound for sequence length	integer ≥ 0
-s	Random seed (default 0)	integer
-o	Output file name	string
-i	Add IDs to output	

Arguments explained

-h and -?: rand_seqs writes a short description of what it does and an overview of its arguments to standard output.

-n This argument must be followed by a non-negative integer number which sets the number of sequences that rand_seqs generates. If this argument is omitted, 100 sequences are generated.

-l This argument must be followed by a non-negative integer number which sets the length of the sequences that rand_seqs generates. If this argument is omitted, the generated sequences are 20 bases long. If the -u switch is also used (see below) the number given after -l defines the lower bound of sequence lengths.

-u This argument must be followed by a non-negative integer number which sets the upper bound for sequence lengths. If this switch is used rand_seqs generates sequences with random lengths that are uniformly distributed between the bounds fixed by -l and -u. If this switch is omitted, all generated sequences have exactly the length determined by -l.

-s This argument must be followed by an integer number which sets the starting point for the pseudo random number generator (the so-called random seed). Setting the random seed to 0 (which is the default value) tells rand_seqs to use the number of seconds elapsed since midnight (00:00:00), January 1, 1970, as stated by system time. CAVEAT: The random seed 0 is actually a mechanism to get a new random seed every time rand_seqs is started. Unfortunately, since a *pseudo* RNG is used, restarting rand_seqs with the same random seed produces the same DNA sequences. Thus, if you restart rand_seqs with 0 as random seed (or without the -s switch) several times within the same second (e.g. when it is called by a script or batch file), your output will always be the same. And even if rand_seqs is called once every second, the generated sequences will be quite similar (try `rand_seqs -n 1 -l 20` several times once a second to see what I mean). Therefore, if you want to generate a number of different sequence pools by calling rand_seqs from within a script, I recommend to use a row of random numbers as random seeds.

-o This argument must be followed by a string that forms a valid filename. The output (the random sequences) is then written to the file with the given name. If a file with the given name already exists its contents are overwritten. If -o is omitted output is written to standard output.

-i If this switch is used each sequence in the output is preceeded by a sequential number (as a simple sequence identifier). The number is seperated from the sequence by a tabulator.

Input

rand_seqs does not take any input besides the command line arguments.

Output

The random sequences are written to standard output, one sequence per line. If the -i switch is used, each sequence is preceeded by a number and a tabulator. If the -o switch is used, output is written to the given file.

Examples

`rand_seqs`
`rand_seqs -s 0`
both generate 100 20mers, taking system time as random seed, and write the sequences to standard output.

`rand_seqs -n 12 -l 23 -s 111 -o twelve_sequences.txt`
generates twelve 23mers, taking 111 as random seed, and writes the sequences to the file *twelve_sequences.txt*.

`rand_seqs -l 20 -u 30 -i`
generates 100 sequences with random lengths ranging between 20 and 30 (boundaries included) and writes them to standard output, preceeded by sequential numbers.

3.1.9 reverse (version 1.0)

This program reads nucleic acid sequences and writes the reverse sequences to standard output. The sequences are usually read from an input file. If no filename is given, reverse works in interactive mode. In this mode the user is prompted to type in a sequence, and the reverse sequence is put out immediately. You can leave the interactive mode (and exit the program) by entering `@`.

Usage

`reverse [-{h|?}] [-i] [-{dna|rna|iupac}] [<input_file>]`

Arguments overview

Argument	Description
-h	Print help text
-?	Print help text
-i	Input file contains sequence IDs
-dna	Sequences are DNA (default)
-rna	Sequences are RNA
-iupac	Sequences contain degenerate bases

Arguments explained

-h and -?: reverse writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells reverse that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below. In interactive mode, this switch is ignored.

-dna, -rna, and -iupac: Only one of these three switches should be used since they state whether the sequences in the input file are DNA or RNA sequences, or whether they contain degenerate bases. These switches determine which characters in the input sequences are treated as non-base encoding characters and are removed in a preprocessing step. If the -dna switch is used (or none of the three switches) `u` and `U` are treated like non-base characters and thus are removed; if the -rna switch is used `t` and `T` are removed. The case of the characters does not matter. If -iupac is used, the input sequences may contain degenerate bases, e.g. `S` for the strong binding bases `C` and `G`. Degenerate bases are encoded according to the IUPAC-IUB recommendations [Cornish-Bowden, 1985] (see appendix A.1). If two or all switches are used, the last one in the command line overrules the preceding ones. If none of these switches is used, the input sequences are supposed to be DNA.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. This ignorance is directed by the use of -dna, -rna, and -iupac, as described above.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the reverse sequences are written to standard output.

Examples

`reverse example_seqs.txt`
reads sequences from the file *example_seqs.txt* and writes the according reverse sequences to standard output.

`reverse -i -rna example_RNA_seqs_w_IDs.txt`
reads RNA sequences from the file *example_RNA_seqs_w_IDs.txt* which contains sequence identifiers and writes the reverse sequences to standard output.

An interactive session tolerating degenerate bases can look like this:

```
>reverse -iupac
Enter sequence (@ to exit):
aacgttt
tttgcaa

Enter sequence (@ to exit):
aaysswrrbby
ybbrrwssyaa

Enter sequence (@ to exit):
```

@

>

3.1.10 seq_dist (version 1.01)

This program reads DNA sequences from an input file (which contains one sequence in each line) and compares each pair of consecutive sequences with several distance measures (so that if n sequences are read $n/2$ pairs are compared). If you want to measure the distance between all possible pairs of sequences in a pool, please use seq_dist2 (see section 3.1.11). The results of the comparisons are written to standard output in one table with a line for each sequence pair and a column for each distance measure.

Usage

`seq_dist [-{h|?}] [-i] [-m <string>] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-m	Distance measure selection (default Hamc;Hme;Hmec;Hdi;Homc;Edic)	string

Arguments explained

-h and -?: seq_dist writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells seq_dist that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below.

-m: This argument must be followed by a string that determines which distance measures are calculated, and with which parameters. Each measure is identified by a three-letter abbreviation, different measures are seperated by `;`, numerical parameters are seperated by `,`. An additional `c` immediately behind the abbreviation indicates that the distance is calculated between the first sequence and the Watson-Crick-complement of the second one. This is useful e.g. when the distance is supposed to indicate the likeliness of two sequences to hybridize. The same abbreviation may occur several times in the string, e.g. when you want to measure a distance with different parameters. See the Examples paragraph below for some example strings. The default string is `Hamc;Hme;Hmec;Hdi;Homc;Edic`.

In the following description, X and Y are two DNA sequences of length n and m, respectively, $x_i, 1 < i < n$, identifies the i-th base of X, and $\overline{Y}$ is the Watson-Crick-complement of Y. The measures, their abbreviations and parameters are:

Ham Hamming distance, i.e. the number of positions i with $x_i \neq y_i$. If $n > m$ (or $n < m$), the number of unpaired bases $n - m$ (or $m - n$) is added.

Hme H-measure [Garzon et al., 1997]. This is an extension of Hamming distance, shifting the sequences and taking the minimum Hamming distance. Furthermore, X is compared with $\overline{Y}$. Let $Ham(\cdot,\cdot)$ be the Hamming distance and $\sigma^k(Y)$ be the sequence Y shifted for k positions to the right (so that negative k results in a left-shift). Then the H-measure $h(\cdot,\cdot)$ is given by $Hme(X,Y) = \min_{-n<k<n}\{|k| + Ham(X,\sigma^k(\overline{Y}))\}$.

Hdi H-distance [Garzon et al., 1997]. Since the H-measure is no metric (e.g. $h(X,X) > 0$ for all sequences that are not self-complementary), Garzon et al. also invented the H-distance, which is not measured between single sequences but between sequence classes, so-called *projective oligos*, or *poligos*. Each poligo contains two sequences that are Watson-Crick-complements of one another. Poligos containing a self-complementary sequence contain only this one sequence. The H-distance between two poligos PX and PY is given by $Hdi(PX,PY) = \min_{\substack{X\in PX,\\ Y\in PY}}\{Hme(X,Y)\}$.

Hom Homology [Feldkamp, 2000] is defined as $Hom(X,Y) = \max\{match(X,\sigma^k(Y))\}/l$, where σ is the shift function described above, *match* counts the number of identical bases $x_i = y_{i+k}$, and $l = \max\{n,m\}$ is the length of the longer sequence. The result lies between 0 and 1, with 1 indicating identical sequences and 0 indicating sequences with no bases in common. Thus, $1 - Hom(X,Y)$ would actually be more suited for a distance measure.

Edi Edit- or Levenshtein-distance (e.g. [Gusfield, 1997]), i.e. the minimum number of operations needed to turn the first sequence into the second. Legal operations are *Insert* adding exactly one base to the sequence, *Delete* removing exactly one base from the sequence, and *Replace* turning one base into another one. For example, $Edi(\texttt{aaaa},\texttt{aacca}) = 2$ because you must at least replace one `a` with a `c` and insert another `c` to get from `aaaa` to `aacca`.

Gal Global alignment using the Needleman-Wunsch algorithm [Needleman and Wunsch, 1970] (better explained e.g. in [Durbin et al., 1998]). Four numerical parameters are needed: the score for matching bases, the penalty for mismatches, the penalty for opening a gap, and the penalty for extending a gap.

LZ1 is a measure based on Lempel-Ziv-complexity [Lempel and Ziv, 1976]. Let $c(X)$ be the Lempel-Ziv-complexity of sequence X and XY the concatenation of sequences X and Y. Then $LZ1(X,Y) = \frac{max\{c(XY)-c(X),c(YX)-c(Y)\}}{max\{c(X),c(Y)\}}$. This measure is called $d^*(X,Y)$ in [Otu and Sayood, 2003].

LZ2 is another Lempel-Ziv-complexity based measure, namely d_1^{**} from [Otu and Sayood, 2003]. It is defined as $LZ2(X,Y) = \frac{c(XY)-c(X)+c(YX)-c(Y)}{\frac{1}{2}(c(XY)+c(YX))}$.

LZ3 is like LZ1, but with a minimum length for vocabulary entries. This minimum length is needed as a parameter.

LZ4 is like LZ2, but with a minimum length for vocabulary entries. This minimum length is needed as a parameter.

LE1 L-tuple based measure $d_L^E(X,Y)$ from [Vinga and Almeida, 2003]. This measure and the following ones calculate for each sequence X a vector $c_L^X = (c_{L,1}^X, ..., c_{L,K}^X)$ with L being

a fixed subsequence length, $K = 4^L$ the number of all possible subsequences of length L, and $c^X_{L,i}$ counting how often the i-th possible L-tuple can be found in sequence X. Then this distance measure is defined as the Euklidean distance between two such vectors: $LE1(X,Y) = (c^X_L - c^Y_L)^T \cdot (c^X_L - c^Y_L) = \sum_{i=1}^{K}(c^X_{L,i} - c^Y_{L,i})^2$. This measure needs the tuple length L as a parameter.

LE2 L-tuple based measure d^2 from [Vinga and Almeida, 2003]. This is a cumulated version of LE1 for several tuple lengths L: $LE2(X,Y) = \sum_{L=l}^{u}\sum_{i=1}^{K}(c^X_{L,i} - c^Y_{L,i})^2$. All weight terms ρ_i mentioned in [Vinga and Almeida, 2003] are here set to 1. This measure needs the lower and upper bounds l and u for the tuple length L as parameters.

LSE L-tuple based measure d^{SE}_L from [Wu et al., 1997] (reproduced incorrectly in [Vinga and Almeida, 2003]). This is a standardized version of the Euklidean distance. $LSE(X,Y) = \sum_{i=1}^{K}\frac{(c^X_{L,i}-c^Y_{L,i})^2}{s_{ii}}$, where $s_{ii} = l \cdot 4^{-L} - 4^{-2L}(l^2 - (l-L+1)(l-L)) + \sum_{k=1}^{L-1}(l-k)4^{-k}Q_{L-k}$, with $l = n-L+1$ and $Q_j = \begin{cases} 1 & \text{if } (z_1,\ldots,z_j) = (z_{L-j+1},\ldots,z_L), \\ 0 & \text{else} \end{cases}$ and $Z = (z_1,\ldots,z_L)$ is the i-th L-tuple. This measure needs the tuple length L as a parameter.

LS2 L-tuple based measure d^{SE*} from [Vinga and Almeida, 2003]. This is a cumulated version of LSE, defined as $LS2(X,Y) = \sum_{L=l}^{u}\sum_{i=1}^{K} LSE(X,Y)$. This measure needs the lower and upper bounds l and u for the tuple length L as parameters.

LLC L-tuple based measure d^{LCC}_L from [Vinga and Almeida, 2003]. This measure is based on the linear correlation coefficient of the frequency vectors f^X_L with $f^X_{L,i} = c^X_{L,i}/(n-L+1)$. It is defined as $LLC(X,Y) = \frac{K\sum_{i=1}^{K} f^X_{L,i}\cdot f^Y_{L,i} - \sum_{i=1}^{K} f^X_{L,i}\cdot\sum_{i=1}^{K} f^Y_{L,i}}{\left[K\sum_{i=1}^{K}(f^X_{L,i})^2 - (\sum_{i=1}^{K} f^X_{L,i})^2\right]^{1/2}\cdot\left[K\sum_{i=1}^{K}(f^Y_{L,i})^2 - (\sum_{i=1}^{K} f^Y_{L,i})^2\right]^{1/2}}$. Actually, it makes more sense to take $1-LLC(X,Y)$ as a distance measure. This measure needs the tuple length L as a parameter.

Lco L-tuple based measure d^{cos}_L from [Vinga and Almeida, 2003]. This measure uses the angle between the counting vectors: $Lco(X,Y) = \theta_{XY}$, with $\cos(\theta_{XY}) = \frac{\sum_{i=1}^{K} c^X_{L,i}\cdot c^Y_{L,i}}{\sqrt{\sum_{i=1}^{K}(c^X_{L,i})^2}\cdot\sqrt{\sum_{i=1}^{K}(c^Y_{L,i})^2}}$. This measure needs the tuple length L as a parameter.

Lev L-tuple based measure d^{Evol}_L from [Vinga and Almeida, 2003]. This measure also used the angle between the counting vectors: $Lev(X,Y) = -\ln[(1+\cos\theta_{XY})/2]$. This measure needs the tuple length L as a parameter.

Lcm L-tuple base measure that counts the L-tuples common to both sequences. This measure needs the tuple length L as a parameter.

LCR Longest common substring, i.e. the longest stretch of *consecutive* bases common to both sequences.

LCQ Longest common subsequences, i.e. the longest series of bases common to both sequences in the same order, but *not necessarily consecutive*.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored. Two consecutive lines contain the sequences of one pair, so that if n lines are read, $n/2$ pairs are compared. If the number of sequences read from the input file is odd, the last sequence is discarded.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the table is written to standard output. The first line of the table contains the column headers, indicating the distance measures calculated, and whether sequence X is compared to sequence Y or its complement $\overline{Y}$. Each following line starts with the identifiers of both sequences, seperated by `,`. If the -i switch is used, these are the identifiers read from the input file. Otherwise, each sequence gets a consecutive number. The following entries are the distances calculated with the different measures. The columns are seperated by tabulators. This does not look very nice in text editors but allows for easy import in MS-Excel or other spreadsheet programs. See below for some example outputs.

Examples

`seq_dist example_seqs.txt`
reads the eight sequences contained in the file *example_seqs.txt*, forms four sequence pairs, and calculates for each such pair (X, Y) the Hamming distance between X and $\overline{Y}$, H-measure between X and Y, H-measure between X and $\overline{Y}$, H-distance between the poligos containing X and Y, Homology between X and $\overline{Y}$, and Edit-distance between X and $\overline{Y}$. The output looks like this:

```
pair Hamming(compl.) HMeasure HMeasure(compl.) HDistance Homology(compl.) Edit(compl.)
1,2 19 11 13 11 0.45 13
3,4 18 12 12 12 0.4 10
5,6 17 12 13 12 0.4 13
7,8 16 14 14 14 0.3 14
```

`seq_dist -i example_seqs_w_IDs.txt`
does the same, but reads also sequence identifiers from the input file, which are also used in the first column of the output:

```
pair Hamming(compl.) HMeasure HMeasure(compl.) HDistance Homology(compl.) Edit(compl.)
Karl,Willy 19 11 13 11 0.45 13
seq03,xyz1 18 12 12 12 0.4 10
dog,cat 17 12 13 12 0.4 13
Igor,last_seq 16 14 14 14 0.3 14
```

`seq_dist -m Ham;Hamc example_seqs.txt`
calculates for each of the four sequence pairs (X, Y) the Hamming distance between X and Y, and the Hamming distance between X and $\overline{Y}$.

`seq_dist -m Gal,1,-1,-3,-0.5 example_seqs.txt`
calculates the global alignment scores for the four sequence pairs, with a base match score of 1, a mismatch penalty of -1, a gap opening penalty of -3, and a gap extension penalty of -0.5.

`seq_dist -m LE1,3;LE1,4;LE1,5 example_seqs.txt`
calculates for the four sequence pairs the L-tuple based distance d_L^E with tuple lengths 3, 4, and 5.

3.1.11 seq_dist2 (version 1.01)

This program reads DNA sequences from an input file (which contains one sequence in each line) and compares each possible pair of sequences with several distance measures. Otherwise, seq_dist2 is very similar to seq_dist, so I will here only describe features in which the two programs differ. For the rest, please refer to section 3.1.10. The results of the comparisons are written to standard output. You can choose whether you want all results in one big table or whether one table for each distance measure is written.

Usage

`seq_dist2 [-{h|?}] [-i] [-m <string>] [-t {1|n}] <input_file>`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	
-m	Distance measure selection (default Hamc;Hme;Hmec;Hdi;Homc;Edic)	string
-t	Number of tables (default 1)	`1` or `n`

Arguments explained

-h and -?: seq_dist2 writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells seq_dist2 that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below.

-m: The string following this argument determines which distance measures are calculated. See section 3.1.10 for details.

-t: This argument must be followed by `1` or `n`, which determines the number of output tables. A `1` means that all results are written in one big table, `n` tells seq_dist2 to write one table for each distance measure. See the output paragraph below for more details. If this argument is omitted, seq_dist2 writes all results in one table.

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the table is written to standard output. If -t 1 is used (or the -t switch is omitted), the output table looks like the one produced by seq_dist (see section 3.1.10). If -t n is used, one table is written for each distance measure used. The first line of each table contains the distance measure name and whether the complement of the second sequence is used, followed by the identifiers of the pairs' second sequences. If the -i switch is used, these are the identifiers read from the input file. Otherwise, each sequence gets a consecutive number. Each following line contains the identifier of the pairs' first sequence and the calculated distances. The columns are seperated by tabulators. This does not look very nice in text editors but allows for easy import in MS-Excel or other spreadsheet programs. Tables are seperated by empty lines. See below for some example outputs.

Examples

`seq_dist2 example_seqs.txt`
reads the eight sequences contained in the file *example_seqs.txt*, forms 64 sequence pairs, and calculates for each such pair (X, Y) the Hamming distance between X and $\overline{Y}$, H-measure between X and Y, H-measure between X and $\overline{Y}$, H-distance between the poligos containing X and Y, Homology between X and $\overline{Y}$, and Edit-distance between X and $\overline{Y}$. The output starts like this:

```
pair Hamming(compl.) HMeasure HMeasure(compl.) HDistance Homology(compl.) Edit(compl.)
1,1 18 10 0 0 0.5 13
1,2 19 11 13 11 0.45 13
1,3 18 13 13 13 0.35 12
1,4 19 13 9 9 0.35 13
1,5 13 13 13 13 0.35 12
1,6 16 14 11 11 0.3 15
1,7 16 14 12 12 0.3 13
1,8 16 13 10 10 0.35 13
2,1 19 11 13 11 0.45 13
2,2 14 10 0 0 0.5 10
...
```

`seq_dist2 -t 1 example_seqs.txt`
does the same.

`seq_dist2 -i example_seqs_w_IDs.txt`
also reads the sequence identifiers from the input file and uses them in the output, which then looks like this:

```
pair Hamming(compl.) HMeasure HMeasure(compl.) HDistance Homology(compl.) Edit(compl.)
Karl,Karl 18 10 0 0 0.5 13
Karl,Willy 19 11 13 11 0.45 13
Karl,seq03 18 13 13 13 0.35 12
Karl,xyz1 19 13 9 9 0.35 13
Karl,dog 13 13 13 13 0.35 12
Karl,cat 16 14 11 11 0.3 15
Karl,Igor 16 14 12 12 0.3 13
Karl,last_seq 16 13 10 10 0.35 13
Willy,Karl 19 11 13 11 0.45 13
Willy,Willy 14 10 0 0 0.5 10
```

...

`seq_dist2 -m Ham;Edic -t n example_seqs.txt`
calculates Hamming distance and Edit distance with complementary second sequences, and prints two tables, one for each measure. The output looks like this:

```
Hamming 1 2 3 4 5 6 7 8
1 0 17 13 9 15 17 14 15
2 17 0 13 16 13 15 13 15
3 13 13 0 14 13 15 19 15
4 9 16 14 0 13 16 16 16
5 15 13 13 13 0 16 17 13
6 17 15 15 16 16 0 19 14
7 14 13 19 16 17 19 0 15
8 15 15 15 16 13 14 15 0

Edit(compl.) 1 2 3 4 5 6 7 8
1 13 13 12 13 12 15 13 13
2 13 10 12 12 12 13 13 11
3 12 12 10 10 12 10 13 13
4 13 12 10 14 14 14 13 12
5 12 12 12 14 14 13 12 15
6 15 13 10 14 13 11 15 14
7 13 13 13 13 12 15 13 14
8 13 11 13 12 15 14 14 10
```

See section 3.1.10 for more measure string examples.

3.1.12 thermo (version 1.01)

This program reads DNA sequences and writes some thermodynamic properties to standard output. These are melting temperature T_m, as well as differences in enthalpy ΔH, entropy ΔS, and Gibbs free energy ΔG of the hybridization reaction between each sequence and its Watson-Crick complement.

The sequences are usually read from an input file. If no filename is given, thermo works in interactive mode. In this mode the user is prompted to type in a sequence, and the thermodynamic data are put out immediately. You can leave the interactive mode (and exit the program) by entering `@`.

Usage

`thermo [-{h|?}] [-Na c] [-Mg c] [-Fo c] [-Sample c] [-p {W|P|B|S|L|U|T}] [-s {W|L|O|U}] [-t] [-i] [<input_file>]`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-i	Input file contains sequence IDs	

-t	Table mode	
-Na or -na	Na^+ concentration (default 0.12 M)	real number ≥ 0
-Mg or -mg	Mg^{++} concentration (default 0.0 M)	real number ≥ 0
-Fo or -fo	formamide concentration (default 0.0 % vol.)	real number ≥ 0
-Sample or -sample	DNA strand concentration (default 1e-9 M)	real number > 0
-p	Calculation method and parameter set (default U)	$\in$ {W,P,B,S,L,U,T}
-s	Salt correction method (default U)	$\in$ {W,L,C,U}

Arguments explained

-h and -?: thermo writes a short description of what it does and an overview of its arguments to standard output.

-i: This switch tells thermo that the first column of the input file contains sequence identifiers (names or numbers). The sequences are then expected in the second column. The identifiers are also used in the output. See also the paragraphs on input and output below. In interactive mode, this switch is ignored.

-t: This switch activates the table mode, i.e. thermo does not write a five-line mini-report for each sequence, but resumes all data for a sequence in a single line. See the paragraph on output below for more details.

-Na or -na: This argument must be followed by a non-negative real number, identifying the molar Na^+ concentration in the solution in which melting or hybridization shall take place. [Na+] and [Mg2+] may not both be 0. If this switch is omitted, Na^+ concentration is set to 0.12 M.

-Mg or -mg: This argument must be followed by a non-negative real number, identifying the molar Mg^{++} concentration in the solution. [Na+] and [Mg2+] may not both be 0. If this switch is omitted, Mg^{++} concentration is set to 0.0 M.

-Fo or -fo: This argument must be followed by a non-negative real number, identifying the formamide concentration (in percent by volume) in the solution. If this switch is omitted, formamide concentration is set to 0.0 % vol. T_m is adjusted by subtracting $0.72 \cdot (\%formamide)$ [McConaughy et al., 1969].

-Sample or -sample: This argument must be followed by a positive real number, identifying the molar DNA strand concentration in the solution. If this switch is omitted, sample concentration is set to 10^{-9} M. This argument is only used for the nearest-neighbor method, and is ignored when the Wallace rule or the percent GC method are chosen.

-p: This argument must be followed by one of the characters W, P, B, S, L, U, or T, encoding the melting temperature calculation method and the parameter set used. The methods and parameter sets are:

W Wallace rule [Suggs et al., 1981]. For each `G` or `C` in the sequence, 4 ℃ are added, for each `A` or `T` 2 ℃. If this method is chosen, only T_m is calculated, but not ΔS, ΔH, or ΔG.

P Percent GC method [Wetmur, 1997]. $T_m = 81.5 + 0.41 \cdot r_{GC} - 500/n$, where n is sequence length and r_{GC} is the GC-ratio $\frac{\text{number of G and C in sequence}}{n}$. If this method is chosen, only T_m is calculated, but not ΔS, ΔH, or ΔG.

B Nearest-neighbor method with Breslauer parameter set [Breslauer et al., 1986]. First,

ΔS, ΔH, and ΔG are calculated by summing up fixed values for each pair of neighboring base pairs. Then, T_m is derived from enthalpy and entropy, using $T_m = \frac{\Delta H}{\Delta S + R \cdot \ln(f \cdot c)}$, where $R = 1.987$ cal/mol·K is the gas constant, c is the molar DNA strand concentration, and factor f equals 1.0 if the sequence is self-complementary, and 0.25 else. Gibbs free energy ΔG is calculated for 25 ℃.

S Nearest-neighbor method with Sugimoto parameter set [Sugimoto et al., 1996]. The method is described above. Gibbs free energy ΔG is calculated for 37 ℃.

L Nearest-neighbor method with SantaLucia parameter set [SantaLucia et al., 1996]. The method is described above. Gibbs free energy ΔG is calculated for 37 ℃.

U Nearest-neighbor method with Unified parameter set [SantaLucia, 1998]. The method is described above. Gibbs free energy ΔG is calculated for 37 ℃.

T Nearest-neighbor method with Tanaka parameter set [Tanaka et al., 2004]. The method is described above. Gibbs free energy ΔG is calculated for 37 ℃.

If this switch is omitted, the nearest-neighbor method with the unified parameter set is used (parameter U).

-s: This argument must be followed by one of the characters W, L, C, or U, encoding the salt concentration correction method used. Melting temperature of nucleic acid strands depends on the salt concentrations in solution. The model used here regards Na^+ and Mg^{++} ion concentrations, where Mg^{++} concentration is transformed into an equivalent Na^+ concentration: $[salt] = [Na^+] + 4 \cdot \sqrt{[Mg^{++}]}$ [Wetmur, 1997]. The methods are:

W Wetmur method [Wetmur, 1997]. T_m is corrected by adding $16.6 \cdot \log(\frac{[salt]}{1+0.7[salt]})$.

L SantaLucia method [SantaLucia et al., 1996]. T_m is corrected by adding $12.5 \cdot \log[salt]$.

C Cantor/Schimmel method [Cantor and Schimmel, 1980]. T_m is corrected by adding $16.6 \cdot \log[salt]$.

U Method from unified parameter set [SantaLucia, 1998]. Here, calculation of ΔS already takes salt concentrations into account. ΔS is adjusted by adding $0.368(n-1) \cdot \ln[salt]$, where n is the sequence length. This also influences ΔG, which is adjusted by subtracting $0.114(n-1) \cdot \ln[salt]$.

If this switch is omitted, the method published with the unified parameter set is used (parameter U).

Input

The input file is supposed to be a normal text file, with one nucleic acid sequence in each line. Empty lines are ignored. Whitespaces and other characters not representing bases are also ignored.

If the -i switch is used, everything from the beginning of each line up to the first space or tabulator is treated as a sequence identifier. The rest of the line is then read as the nucleic acid sequence.

See appendix B.3 for examples of input files.

Output

No files are written, the thermodynamic data are written to standard output.

If the -t switch is used, all results are put in a table. The first line contains the column headers. Each following line contains the data for one examined sequence. It starts with the base sequence itself, eventually preceeded by the sequence identifier if the -i switch is used, followed by T_m, and, if the nearest-neighbor method is chosen, ΔH, ΔS, and ΔG. The columns are seperated by tabulators, which does not look very nice in a text editor, but allows for easy import in MSExcel or other spreadsheet programs.

If the -t switch is not used, a block of two or five lines is printed for each examined sequence. The first line contains the base sequence itself, eventually preceeded by the sequence identifier if the -i switch is used. The next line shows the calculated T_m. If the nearest-neighbor method is used, three additional lines show ΔH, ΔS, and ΔG, respectively. Each value is preceeded by the name of the calculated variable and an equal sign, and is followed by the appropriate dimension. Two consecutive blocks are seperated by an empty line.

The output format is the same in interactive mode (when no input file is given). See the next paragraph for some example outputs.

Examples

`thermo example_seqs.txt`
reads sequences from the file *example_seqs.txt*, calculates T_m, ΔH, ΔS, and ΔG with the nearest-neighbor model using the unified parameter set and the according salt correction term. The results are then written to standard output, in five line blocks. The output starts like this:

```
ttcgccctgctactaacacg
Tm = 53.2752 deg C
DeltaH = -158.4 kcal / mol
DeltaS = -441.325 cal / K per mol
DeltaG = -21.4175 kcal / mol, at 37 deg C

agataacagcaggatttctt
Tm = 44.8461 deg C
DeltaH = -147.6 kcal / mol
DeltaS = -420.225 cal / K per mol
DeltaG = -17.1275 kcal / mol, at 37 deg C

ggatcgcaggatctcagtca
Tm = 52.0266 deg C
DeltaH = -154.9 kcal / mol
DeltaS = -432.425 cal / K per mol
DeltaG = -20.6775 kcal / mol, at 37 deg C
...
```

`thermo -i -p W example_seqs_w_IDs.txt`
also reads sequence identifiers from the input file and uses the simple Wallace method. Thus, only T_m is calculated:

```
Karl    ttcgccctgctactaacacg
```

```
Tm = 62 deg C

Willy   agataacagcaggatttctt
Tm = 54 deg C

seq03   ggatcgcaggatctcagtca
Tm = 62 deg C
...
```

`thermo -Na 0.3 -Mg 0.1 example_seqs.txt`
raises the salt concentrations in the solution (compared to the default values), thus stabilizing the duplexes and raising also the melting temperature.

```
ttcgccctgctactaacacg
Tm = 65.8182 deg C
DeltaH = -158.4 kcal / mol
DeltaS = -434.918 cal / K per mol
DeltaG = -23.4022 kcal / mol, at 37 deg C

agataacagcaggatttctt
Tm = 57.643 deg C
DeltaH = -147.6 kcal / mol
DeltaS = -413.818 cal / K per mol
DeltaG = -19.1122 kcal / mol, at 37 deg C

ggatcgcaggatctcagtca
Tm = 64.7641 deg C
DeltaH = -154.9 kcal / mol
DeltaS = -426.018 cal / K per mol
DeltaG = -22.6622 kcal / mol, at 37 deg C
...
```

`thermo -p L -s L -t example_seqs.txt`
uses the nearest-neighbor method with parameter set and salt correction term taken from the 1996 paper of SantaLucia [SantaLucia et al., 1996]. The results are then printed in a table:

```
sequence Tm [deg C] DeltaH [kcal/mol] DeltaS [cal/K mol] DeltaG_37 [kcal/mol]
ttcgccctgctactaacacg 52.3596 -151.4 -405.3 -25.29
agataacagcaggatttctt 43.7015 -141.6 -387.3 -21.58
ggatcgcaggatctcagtca 51.7963 -142.5 -379.6 -24.96
gtgaccctccttccagtccg 57.5841 -141.7 -370.1 -26.51
gaattccatatcccttccaa 44.735 -137.5 -373.5 -21.49
agaccgggctccgcacctgt 62.919 -150.2 -388.2 -29.22
cttcacatacaaaattaatc 37.2746 -143.4 -401.5 -18.77
gtccttcccgcggtttctac 55.496 -150.7 -399.1 -26.37
```

An interactive session using default settings can look like this:

```
>thermo
```

```
Enter sequence (@ to exit):
aaaaaaaaaa
Tm = 1.01574 deg C
DeltaH = -66.5 kcal / mol
DeltaS = -198.622 cal / K per mol
DeltaG = -4.76461 kcal / mol, at 37 deg C

Enter sequence (@ to exit):
gcgcgcgcgc
Tm = 45.7317 deg C
DeltaH = -91.2 kcal / mol
DeltaS = -244.822 cal / K per mol
DeltaG = -15.3446 kcal / mol, at 37 deg C

Enter sequence (@ to exit):
@

>
```

3.2 Miscellaneous Tools

I also wrote some tools that have nothing to do with DNA sequences, but that I needed for statistical analysis. Since they might be useful for someone else, I just put them into CANADA.

3.2.1 corr_coeff (version 1.0)

This program reads a table of measurements from an input file where each column contains the measurements for one variable and each row contains the measurements for one sample. The linear correlation coefficient for each pair of variables is written in tabular form to standard output.

Usage

`corr_coeff {[-{h|?}] | -r <number> -c <number> [-R <number>] [-C <number>] <input_file>}`

Arguments overview

Argument	Description	Parameters
-h	Print help text	
-?	Print help text	
-r	Number of rows (samples)	integer ≥ 1
-c	Number of columns (variables)	integer ≥ 1
-R	Number of lines to be ignored (default 1)	integer ≥ 0
-C	Number of columns to be ignored (default 1)	integer ≥ 0

Arguments explained

-h and -?: corr_coeff writes a short description of what it does and an overview of its arguments to standard output.

-r: This argument must be followed by a positive integer, specifying the number of samples, i.e. the number of rows in the input file containing data. This number does not include the lines ignored by using the -R switch. This argument is mandatory.

-c: This argument must be followed by a positive integer, specifying the number of variables that shall be compared, i.e. the number of columns in the input file. This number does not include the columns ignored by using the -C switch. This argument is mandatory.

-R: This argument must be followed by a non-negative integer, specifying the number of leading lines in the input file that do not contain data and are to be ignored (e.g. the table header line). If this switch is omitted, the first line will be ignored.

-C: This argument must be followed by a non-negative integer, specifying the number of leading columns in the input file that do not contain data and are to be ignored (e.g. identifiers or other row headers in a table). If this switch is omitted, the first column will be ignored.

Input

The input file is supposed to be a normal text file, containing the measured samples for the variables that shall be compared in a table. Each row contains the measurements for one sample, each column contains the measurements for one variable. Columns are seperated by tabulators. The table may also contain extra rows and columns, e.g. for the names of the variables and samples. These can be ignored by using the -R and -C arguments.

See appendix B.4 for examples of input files.

Output

No files are written, the table of correlation coefficients are written to standard output.

The first line of the table begins with r, the usual symbol for correlation coefficients, followed by the numbers 1 through n, where n is the number of variables specified with the -c argument. Each following line begins with a variable's consecutive number, followed by the correlation coefficients.

See the next paragraph for some example outputs.

Examples

`corr_coeff -r 20 -c 4 corr_example.txt`
reads 21 rows with 5 columns from the file *corr_example.txt*, ignores the first row and column, and treats the rest as 20 measurements of 4 variables. It then prints the linear correlation coefficients for all pairs of variables to standard output in a table that looks like this:

```
r       1       2       3       4
1       1       -1      1       0.27
2       -1      1       -1      -0.27
3       1       -1      1       0.27
4       0.27    -0.27   0.27    1
```

`corr_coeff -r 20 -c 4 -R 0 -C 0 corr_example_data_only.txt`
does the same, but here the input file contains only measurements, so no row or column shall

be ignored.

3.2.2 rank_corr (version 1.0)

This program reads a table of measurements from an input file where each column contains the measurements for one variable and each row contains the measurements for one sample. The Spearman rank correlation coefficient for each pair of variables is written in tabular form to standard output.

Actually, everything written about corr_coeff (see section 3.2.1 above) is also valid for rank_corr. The only difference is the meaning of the results you get. So I will not repeat the complete stuff from the section on corr_coeff, but will only describe the differences.

Usage

```
rank_corr {[-{h|?}] | -r <number> -c <number> [-R <number>] [-C <number>]
<input_file>}
```

Output

In addition to the output described for corr_coeff (see 3.2.1), rank_corr also produces a file named *ranks.txt*. This plain text file contains a table with n rows and m columns, where n and m are the numbers of measurements and variables specified by the -r and -c arguments, respectively. This table contains the ranks each measurements gets, which are used for calculating the Spearman rank correlation coefficient.

See the next paragraph for some example output.

Examples

`rank_corr -r 20 -c 4 corr_example.txt`
reads 21 rows with 5 columns from the file *corr_example.txt*, ignores the first row and column, and treats the rest as 20 measurements of 4 variables. It then prints the Spearman rank correlation coefficients for all pairs of variables to standard output in a table that looks like this:

```
r       1       2       3       4
1       1       -1      1       0.28
2       -1      1       -1      -0.27
3       1       -1      1       0.28
4       0.28    -0.27   0.28    1
```

In addition, a file named *ranks.txt* containing the ranks of the measurements is produced, which looks like this:

```
1       20      1       2
2       19      2       14.5
3       18      3       5.5
4       17      4       8
5       16      5       2
6       15      6       12.5
7       14      7       10.5
8       13      8       18.5
9       12      9       8
10      11      10      18.5
11      10      11      5.5
12      9       12      4
13      8       13      18.5
14      7       14      8
15      6       15      16
16      5       16      12.5
17      4       17      14.5
18      3       18      10.5
19      2       19      2
20      1       20      18.5
```

Appendix A

Tables

A.1 IUPAC-IUB Recommendation for Degenerate Base Symbols

Symbol	Bases	Description
A	A	**A**denin
C	C	**C**ytosin
G	G	**G**uanin
T	T	**T**hymin
R	G, A	Pu**r**ine
Y	T, C	P**y**rimidine
M	A, C	A**m**ino
K	G, T	**K**eto
S	G, C	**s**trong bond (3 H-bridges)
W	A, T	**w**eak bond (2 H-bridges)
H	A, C, T	not G (H follows G in the alphabet)
B	C, G, T	not A
V	A, C, G	not T (for RNA: not U)
D	A, G, T	not C
N	A, C, G, T	a**n**y base

Table A.1: Notation for incompletely specified (degenerate) bases, following the IUPAC-IUB recommendation [Cornish-Bowden, 1985]. The three columns show the recommended symbols, the bases represented by each symbol, and a short hint why these symbols are chosen.

Appendix B

Example Input Files

B.1 DeLaNA Files

B.1.1 all_features.dln

The file *all_features.dln* is a DeLaNA file containing all objects and statements with their properties.

```
// This is a comment line.

/* This is a comment
running over
several lines. */

SEQUENCE fixed_oligo {
  seq_mask = "aattggcc"; }

SEQUENCE oligo_A, oligo_B {
  NA_type = RNA;
  length = 10;
  GC_ratio = 0.3;
  Tm = [55;60];
  DG = [-10.0;0.0];
  seq_mask = "nnaunncgnn";
  forbidden = fixed_oligo, "ggg"; }

SEQUENCETYPE my_20mer {
  NA_type = DNA;
  length = 20;
  GC_ratio = [0.1;0.9];
  Tm = [55;65];
  DeltaG = [-100.0;-10.0];
  seq_mask = "yyyyyyyyyyrrrrrrrrrr";
  forbidden = "ata"; }
```

```
my_20mer oligo_C;

my_20mer oligo_D {
  Tm = [0.0;100.0];
  forbidden += "atg", "gtg", "ttg"; }


CONCAT oligo_A, oligo_B;

CONCAT Complement(oligo_A), oligo_C;

CONCAT oligo_B, Complement(oligo_D);

3WJ oligo_A, Complement(oligo_B), oligo_D;

4WJ oligo_A, complement(oligo_D), complement(oligo_B), oligo_C;


POOL p1 {
  sequences = oligo_A, oligo_B, oligo_C, oligo_D;
  n_uniqueness = 6;
  Hamming = [5;20]
  H_distance = [5;20];
  sample_conc = 0.005;
  Na_conc = 0.123;
  formamide_conc = 0.001;
  Tm_method = Sugimoto;
  salt_method = CantorSchimmel;
  violation_tolerance = 1;
  forbidden = "ttt", fixed_oligo, complement(fixed_oligo);
  no_shorties = false;
  no_GGG = false;
  no_AUG = false;
  no_GUG = false;
  no_UUG = false;
  no_fraying = true;
  base_strand_GC = [0.3;0.7];
}

DESIGNTOOL {
  random_seed = 12345;
  ID_in_table = true;
  analyze_uniqueness = false;
}
```

B.1.2 dsg_small_pool.dln

This input file for dsg is used to generate a 4-unique pool containing different oligomers. No more than two consecutive guanine bases may occur.

```
// 4-unique pool of oligomers
// without three or more G's in a row

SEQUENCE oli1 {
  length = 10;
  }

SEQUENCE oli2 {
  length = 20;
  GC_ratio = 0.5;
  }

SEQUENCE oli3, oli4 {
  length = 15;
  Tm = [50;55];
  }

POOL mypool {
  n_uniqueness = 6;
  Na_conc = .05;
  sample_conc = 2e-7;
  Formamide_conc = 0;
  forbidden = "ggg";
  }

DESIGNTOOL {
  Random_Seed = 0;
  }
```

B.1.3 dsg_big_pool.dln

This input file for dsg is used to generate a 6-unique pool containing 100 oligomers of length 20.

```
// 6-unique pool of 100 20mers

SEQUENCETYPE twentymer {
  length = 20; }

twentymer x[100];

POOL mypool {
  n_uniqueness = 6;
```

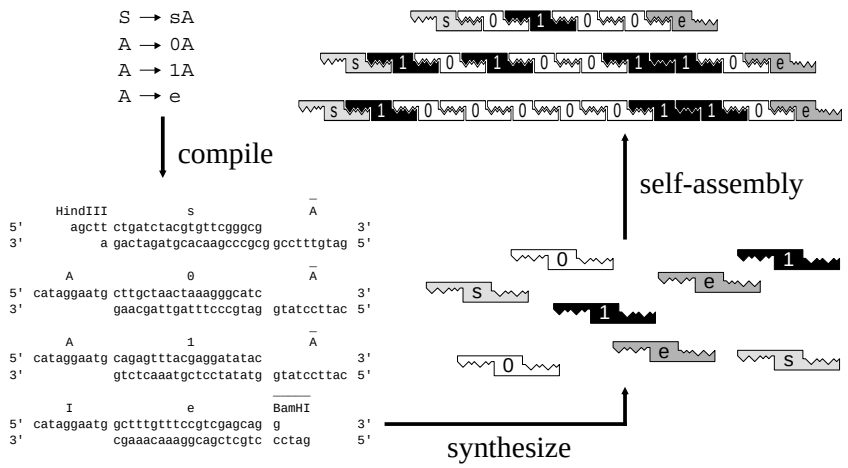

Figure B.1: Binary random number generator with DNA molecules [Rauhe et al., 2000, Feldkamp et al., 2003].

```
  Na_conc = .05;
  sample_conc = 2e-7;
  Formamide_conc = 0;
  }

DESIGNTOOL {
  Random_Seed = 0;
  }
```

B.1.4 dsc_RNG.dln

This input file for dsc is used to generate oligomers for a binary random number generator as described in [Rauhe et al., 2000, Feldkamp et al., 2003] (see Fig. B.1).

```
// binary random number generator

SEQUENCETYPE terminal {
  length = 20;
  GC_ratio = 0.5;
}

SEQUENCETYPE variable {
  length = 10;
}

terminal s, e, t0, t1;

variable A;

SEQUENCE HindIII {  // restriction site
```

```
  seq_mask = "aagctt";
  }

SEQUENCE BamHI {    // restriction site
  seq_mask = "ggatcc";
  }

CONCAT HindIII, s;
CONCAT s, A;

CONCAT A, t0;
CONCAT t0, A;

CONCAT A, t1;
CONCAT t1, A;

CONCAT A, e;
CONCAT e, BamHI;

POOL p {
  N_uniqueness = 4;
  Violation_tolerance = 1;
  Sample_conc = 2e-7;
  Na_conc = 1.0;
  Tm_method = NNSantaLucia;
  }

DESIGNTOOL  {
  Random_Seed = 0;
}
```

B.1.5 dsc_4WJ.dln

This input file for dsc is used to generate oligomers for a four-way junction structural motif as described in [Seeman, 1982] (see Fig. B.2) without using the 4WJ statement.

```
// 4 way junction motif
// without using the 4WJ statement

SEQUENCETYPE arm {
  length = 8;
}

SEQUENCETYPE sticky_end {
  length = 5;
}

// one arm for each direction
```

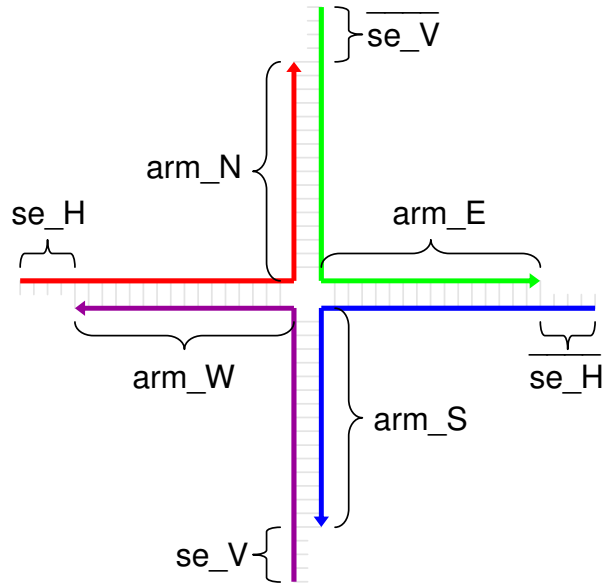

Figure B.2: Sketch of a four-way junction motif. Colored arrows represent the strands' backbones, with the arrow point indicating the 3'-end. Bases are indicated by grey bars. The number of bases/base pairs shown here does not have to match the number in the DeLaNA file.

```
arm arm_N, arm_E, arm_S, arm_W {
  GC_ratio = 0.5; }

// one sticky end for horizontal assembly,
// one for vertical assembly
sticky_end se_V, se_H {
  GC_ratio = [0.6;1.0]; // a little bit stability
  }


// red strand
CONCAT se_H, complement(arm_W);
CONCAT complement(arm_W), arm_N;

// green strand
CONCAT complement(se_V), complement(arm_N);
CONCAT complement(arm_N), arm_E;

// blue strand
CONCAT complement(se_H), complement(arm_E);
CONCAT complement(arm_E), arm_S;

// violet strand
CONCAT se_V, complement(arm_S);
CONCAT complement(arm_S), arm_W;
```

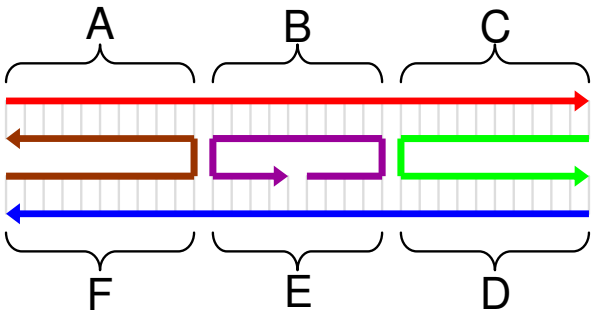

Figure B.3: Sketch of a DAE-DX motif [Fu and Seeman, 1993]. Colored arrows represent the strands' backbones, with the arrow point indicating the 3'-end. Bases are indicated by grey bars. The number of bases/base pairs shown here does not have to match the number in the DeLaNA file.

```
POOL p {
  N_uniqueness = 4;
  Violation_tolerance = 0;
  Sample_conc = 2e-7;
  Na_conc = 1.0;
  Tm_method = NNSantaLucia;
  }

DESIGNTOOL  {
  Random_Seed = 0;
}
```

B.1.6 dsc_DX.dln

This input file for dsc is used to generate oligomers for a double-crossover (DX) structural motif as described in [Fu and Seeman, 1993] (more precisely: a DAE-DX tile) (see Fig. B.3).

```
// DAE double crossover motif without sticky ends
// [fu1993], Fig. 4

// upper helix
SEQUENCE A {
  length = 11; }
SEQUENCE B {
  length = 10; }
SEQUENCE C {
  length = 11; }

// lower helix
SEQUENCE D {
  length = 11; }
SEQUENCE E {
```

```
  length = 10; }
SEQUENCE F {
  length = 11; }

/*
// If one would not use the 4WJ statement, it could look like this:

// red strand
CONCAT A, B;
CONCAT B, C;

// blue strand
CONCAT D, E;
CONCAT E, F;

// brown strand
CONCAT complement(F), complement(A);

// violet strand with no nick
CONCAT complement(E), complement(B);
CONCAT complement(B), complement(E);

// green strand
CONCAT complement(C), complement(D);
*/

// left crossover point
4WJ complement(A), B, complement(E), F;

// right crossover point
4WJ complement(B), C, complement(D), E;


POOL p {
  N_uniqueness = 4;
  Violation_tolerance = 0;
  Sample_conc = 2e-7;
  Na_conc = 1.0;
  Tm_method = NNSantaLucia;
  }

DESIGNTOOL  {
  Random_Seed = 0;
}
```

B.2 Configuration Files

B.2.1 dsc_config.cfg

The file *dsc_config.cfg* contains parameter settings specific for the design tool dsc.

```
// sample config file
random_seed = 12345;
ID_in_table = true;
analyze_uniqueness = false;
RTF = true;
seperate_entries = true;
group_size = c;
group_shuffling = true;
successor_choice = f;
SC_parameter = 0.1;
/* closing comment */
```

B.3 Sequence Pools

This sections shows some files containing DNA sequences, serving as input for several tools.

B.3.1 example_seqs.txt

The file *example_seqs.txt* contains eight randomly generated 22mers, one sequence per line, in lower case.

```
gtacttccttaaacgacgcagg
ggcggtaaaagaatcttggctg
catatctcggcacacatgatgg
cttatcgctttatgaccggacc
gcttcggattaacagtgacgtg
caatgaaacactaggcgaggac
cttcacgattgccactttccac
cgtgtagcctttgtattcgtcc
```

B.3.2 example_seqs_w_IDs.txt

The file *example_seqs_w_IDs.txt* contains the same eight 22mers as *example_seqs.txt* (see above), but with an additional identifier preceding the actual sequence. The two columns are tabulator-seperated.

```
eins    gtacttccttaaacgacgcagg
Karl    ggcggtaaaagaatcttggctg
23      catatctcggcacacatgatgg
x       cttatcgctttatgaccggacc
Willy   gcttcggattaacagtgacgtg
```

```
ABC     caatgaaacactaggcgaggac
Lemmy   cttcacgattgccactttccac
17      cgtgtagcctttgtattcgtcc
```

B.4 Tables with Numbers

This section shows some files containing tab-seperated tables comprised of numerical data, serving as input files for the statistical analysis tools.

B.4.1 corr_example.txt

The file *corr_example.txt* contains contains 20 samples or measurements of four viariables. The variables are called 'first', 'second', 'third', and 'last'. The samples are labelled with the lower case letters 'a' through 't'. The columns are tabulator-seperated.

```
cc      first   second  third   last
a       1       20      2       21
b       2       19      4       27
c       3       18      6       23
d       4       17      8       24
e       5       16      10      21
f       6       15      12      26
g       7       14      14      25
h       8       13      16      29
i       9       12      18      24
j       10      11      20      29
k       11      10      22      23
l       12      9       24      22
m       13      8       26      29
n       14      7       28      24
o       15      6       30      28
p       16      5       32      26
q       17      4       34      27
r       18      3       36      25
s       19      2       38      21
t       20      1       40      29
```

B.4.2 corr_example_data_only.txt

The file *corr_example_data_only.txt* contains the same numbers as *corr_example.txt* (see above), but without the first line and column containing identifiers of variables and samples.

```
1       20      2       21
2       19      4       27
3       18      6       23
4       17      8       24
5       16      10      21
```

6	15	12	26
7	14	14	25
8	13	16	29
9	12	18	24
10	11	20	29
11	10	22	23
12	9	24	22
13	8	26	29
14	7	28	24
15	6	30	28
16	5	32	26
17	4	34	27
18	3	36	25
19	2	38	21
20	1	40	29

Bibliography

[Ackermann and Gast, 2003] Ackermann, J. and Gast, F.-U. (2003). Word design for biomolecular information processing. *Zeitschrift für Naturforschung*, 58a:157–161.

[Allawi et al., 1997] Allawi, H. T., Peyret, N., Seneviratne, P. A., and SantaLucia, Jr., J. (1997). DNA mismatch thermodynamics and structure. *Abstracts of Papers of the American Chemical Society*, 213:270.

[Allawi and SantaLucia, 1998a] Allawi, H. T. and SantaLucia, Jr., J. (1998a). Nearest neighbor thermodynamic parameters for internal G·A mismatches in DNA. *Biochemistry*, 37(8):2170–2179.

[Allawi and SantaLucia, 1998b] Allawi, H. T. and SantaLucia, Jr., J. (1998b). Nearest-neighbor thermodynamics of internal A·C mismatches in DNA: Sequence dependence and pH effects. *Biochemistry*, 37(26):9435–9444.

[Allawi and SantaLucia, 1998c] Allawi, H. T. and SantaLucia, Jr., J. (1998c). Thermodynamics of internal C·T mismatches in DNA. *Nucleic Acids Research*, 26(11):2694–2701.

[Breslauer et al., 1986] Breslauer, K. J., Frank, R., and Blöcker, H. (1986). Predicting DNA duplex stability from the base sequence. *Proceedings of the National Academy of Sciences*, 83(4):3746–3750.

[Cantor and Schimmel, 1980] Cantor, C. R. and Schimmel, P. R. (1980). *Biophysical Chemistry Part III: The Behavior of Biological Macromolecules*. W. H. Freeman and Company.

[Cornish-Bowden, 1985] Cornish-Bowden, A. (1985). Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic Acids Research*, 13(9):3021–3032.

[Durbin et al., 1998] Durbin, R., Eddy, S., Krogh, A., and Mitchinson, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.

[Feldkamp, 2000] Feldkamp, U. (2000). Ein DNA-Sequenz-Compiler. Technical Report of the Systems Analysis Research Group SYS–2/00, University of Dortmund, Department of Computer Science.

[Feldkamp and Niemeyer, 2006] Feldkamp, U. and Niemeyer, C. M. (2006). Rational design of DNA nanoarchitectures. *Angewandte Chemie International Edition*, 45:1856–1876.

[Feldkamp et al., 2003] Feldkamp, U., Rauhe, H., and Banzhaf, W. (2003). Software tools for DNA sequence design. *Genetic Programming and Evolvable Machines*, 4(2):153–171.

[Fu and Seeman, 1993] Fu, T.-J. and Seeman, N. C. (1993). DNA double-crossover molecules. *Biochemistry*, 32:3211–3220.

[Garzon et al., 1997] Garzon, M. H., Deaton, R., Neathery, P., Franceschetti, D. R., and Murphy, R. (1997). A new metric for DNA computing. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Conference on Genetic Programming, GP-97*, Stanford University, Stanford, California. Special Track on DNA computing.

[Gibas and Jambeck, 2001] Gibas, C. and Jambeck, P. (2001). *Developing Bioinformatics Computer Skills*. O'Reilly.

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

[Hofacker et al., 1994] Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M., and Schuster, P. (1994). Fast folding and comparison of RNA secondary structures. *Chemical Monthly*, 125:167–188.

[Lempel and Ziv, 1976] Lempel, A. and Ziv, J. (1976). On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81.

[McConaughy et al., 1969] McConaughy, B. L., Laird, C. D., and McCarthy, B. J. (1969). Nucleic acid reassociation in formamide. *Biochemistry*, 8(8):3289–3295.

[Needleman and Wunsch, 1970] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

[Otu and Sayood, 2003] Otu, H. H. and Sayood, K. (2003). A new sequence distance measure for phylogenetic tree construction. *Bioinformatics*, 19(16):2122–2130.

[Peyret et al., 1999] Peyret, N., Seneviratne, P. A., Allawi, H. T., and SantaLucia, Jr., J. (1999). Nearest-neighbor thermodynamics and NMR of DNA sequences with internal A·A, C·C, G·G and T·T mismatches. *Biochemistry*, 38:3468–3477.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge University Press, 2 edition.

[Rauhe et al., 2000] Rauhe, H., Vopper, G., Feldkamp, U., Banzhaf, W., and Howard, J. C. (2000). Digital DNA molecules. In Condon, A. E. and Rozenberg, G., editors, *Preproceedings of the 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 2000*, page 271. Leiden center for natural computing. Poster, manuscript available under http://ls11-www.informatik.uni-dortmund.de/molcomp/Publications/publications.html.

[SantaLucia, 1998] SantaLucia, Jr., J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95:1460–1465.

[SantaLucia et al., 1996] SantaLucia, Jr., J., Allawi, H. T., and Seneviratne, P. A. (1996). Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry*, 35(11):3555–3562.

[Seeman, 1982] Seeman, N. C. (1982). Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247.

[Suggs et al., 1981] Suggs, S. V., Hirose, T., Miyake, T., Kawashima, E. H., Johnson, M. J., Itakura, K., and Wallace, R. B. (1981). Use of synthetic oligodeoxyribonucleotides for the isolation of specific cloned DNA sequences. In Brown, D. D., editor, *ICN-UCLA Symp. Developmental Biology Using Purified Genes*, volume 23, pages 683–693. Academic Press, New York.

[Sugimoto et al., 1996] Sugimoto, N., Nakano, S., Yoneyama, M., and Honda, K. (1996). Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes. *Nucleic Acids Research*, 24(22):4501–4505.

[Tanaka et al., 2004] Tanaka, F., Kameda, A., Yamamoto, M., and Ohuchi, A. (2004). Thermodynamic parameters based on a nearest-neighbor model for DNA sequences with a single-bulge loop. *Biochemistry*, 43:7143–7150.

[Vinga and Almeida, 2003] Vinga, S. and Almeida, J. (2003). Alignment-free sequence comparison — a review. *Bioinformatics*, 19(4):513–523.

[Welsh and Powell, 1967] Welsh, D. J. A. and Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86.

[Wetmur, 1997] Wetmur, J. G. (1997). Physical chemistry of nucleic acid hybridization. In Wood, D., editor, *DNA3*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, pages 1–23, Providence, RI. American Mathematical Society.

[Wu et al., 1997] Wu, T.-J., Burke, J. P., and Davison, D. B. (1997). A measure of DNA sequence dissimilarity based on Mahalanobis distance between frequencies of words. *Biometrics*, 53:1431–1439.