StemSearch

The documentation for the program
StemSearch version 0.9, 1 July 2008

Niels Richard Hansen

This manual is for StemSearch version 0.9.

Table of Contents

1 Introduction

The program `StemSearch` can be used to search a DNA or RNA sequence for putative formation of local stems or stem-loops. Throughout this manual the program is described as searching RNA sequences only. DNA input is equally valid, but internally transcribed anyway, so for simplicity we will only operate with one type of sequence. This chapter provides a brief overview of how to invoke the program `StemSearch` from the command line, the options it takes and the formats of the input and output.

The current version of the program is 0.9. This is a beta-version. See Chapter 6 [To do], page 13 for a list of things that will be implemented before the program becomes version 1.0.

The details of the theory behind `StemSearch` can be found in the paper Statistical models of local RNA stem-loop scores.

1.1 Invoking StemSearch

You can call `StemSearch` from the command line and specify the file containing the sequence with the '`-f`' option. If the file '`foo`' contains the single line '`acuggucuau`' we get

```
> StemSearch -f foo
#StemSearch -f foo
>NN
Position: (1,6)          Anchor: (3,4)   Score: 2.246    E-value: 1.058
acuggu
((..))
```

The program returns by default a header line with the call and a four line output containing the identifier for the sequence and coordinates for the stem, the actual sequence of the highest scoring stem, and the corresponding structure in bracket notation. If the file contains several sequences in FASTA format, a stem-loop structure is found for each sequence separately. See Section 1.3 [IO-formats], page 2, for more information about the output format.

A default parameter file is used if no alternative is supplied. Use the '`-p foo.par`' to use the parameter file '`foo.par`'. See Section 1.2 [StemSearch Options], page 1, and in particular Chapter 3 [Parameters], page 7 for detailed information about the organization of the parameter files. Note that a good choice of parameters is problem specific. It depends on the kind of stems or stem-loops that you search for as well as the letter composition of the sequence you search within.

1.2 StemSearch Options

```
StemSearch [-f sequence file] [-I start end] [-L bandwidth]
[-t threshold] [-id] [-o s|nl] [-p parameter file] [-r nr of reported
sequences] [-nome] [-nolp] [-l] [-g] [-s seed]
```

'`-f`' The sequence file in FASTA format. See Section 1.3.1 [Input formats], page 2. If no file is specified with '`-f`', the program simulates a random sequence.

'`-I`' The substring of the sequence used specified by the starting and ending position relative to the entire sequence read. If '`-f`' is omitted, this option specifies the length of the simulated sequence.

'`-L`' The bandwidth of the upper triangular, dynamic programming matrix. By default, the matrix is not band limited. Default bandwidth is 300. The running time is linear in the bandwidth.

'`-t`' The threshold for the score of reported sequences. Only sequences with a score larger than the threshold are reported. Default value is 10. Threshold is ignored if '`-id`' is not used.

‘`-id`’	Enables the island declumping algorithm. With this option set, the program finds not only the highest scoring stem but all essentially different high scoring stems with scores above the threshold. The running time of the actual search algorithm with island declumping enabled is proportional (with a constant close to 1) to the running time without declumping. If, however, the threshold is low the handling of (the large number of) islands and island scores may take a considerable amount of extra time.
‘`-o s|nl`’	Specify the output format. There are two formats. The default, `standard`, format and a short format (‘`-o s`’). The standard format has the subformat standard-no-loop, specified by ‘`-o nl`’, which is identical to the standard output except that the entire sequence in the loop is not printed.
‘`-p`’	Specify the parameter file. Default value is ‘`default.par`’. See Chapter 3 [Parameters], page 7.
‘`-r`’	The number of reported sequences. By default, the highest scoring stem is reported, and with ‘`-id`’ set all stems with score > the threshold are reported. In the last case you can fix the number of sequences reported to be, say, 10 even if the threshold is low. Then only the 10 highest scoring stems are reported.
‘`-nome`’	No memory efficiency. The default algorithm is memory efficient and requires only memory proportional to the squared bandwidth. This option disables the memory efficient algorithm and uses one that stores the entire dynamic programming matrix. This requires memory proportional to the length of the sequence times the bandwidth. The option is mostly for testing purposes. It is also used implicitly in connection with ‘`-g`’.
‘`-nolp`’	No loop penalty. By default, the program penalizes the hairpin loop of the stem according to the parameter file. Setting ‘`-nolp`’ the hairpin loop does not contribute to the score. See Section 3.4 [Loop Penalty], page 9.
‘`-g`’	Greedy structure prediction. This option automatically enables ‘`-nome`’ and if ‘`-id`’ is set it will be ignored. This option computes the same dynamic programming matrix used to locate high scoring stems, but then, in a greedy way, it assembles the stems to form a prediction of secondary structure for the entire sequence.
‘`-l`’	Create log-file. Append warnings and other important information during IO and computations to the StemSearch.log file. Mostly for debugging.
‘`-s`’	Sets the seed for the random number generator. Useful for debugging. If no seed is provided the time (C++ time(0)) is used.

1.3 IO-formats

Essentially `StemSearch` reads a file in FASTA format with one or multiple RNA or DNA sequences and writes to `stdout` a list of putative stems in the sequences. The output is reported in one of two formats; standard or short. The details are described below in Section 1.3.2 [Output formats], page 3. Several of the options, Section 1.2 [StemSearch Options], page 1, also affect the precise content of the output.

1.3.1 Input formats

The format of a sequence is a plain text string containing the letters `a`, `c`, `g`, `t` or `u`, where the two last letters are treated identically by the program. Capital letters are not distinguished from small letters. Thus the following two sequences are from the programs point of view identical:

```
acgugucagguggaccggaucgaugaugcgau
```

```
ACgugtcaGGTggaCcggAtcgaugAugcgaT
```

A mixture of capital and small letters and a mixed use of `t` and `u` is not encouraged though. Use `t` if your sequence is a DNA sequence and `u` if it is an RNA sequence. Use capital letters for the whole or parts of the sequence if that is meaningful in another context. It will not affect the sequence output, which will always be a small letter, RNA sequence. All other letters occurring in the sequence are treated as an unknown letter represented by an `n`. Such a letter is not allowed to occur in a stem.

In a file, the sequence(s) should be organized in FASTA format. That is, each sequence is preceded by a single-line with an identification number and description starting with ‘`>`’.

When `StemSearch` reads from a file using ‘`-f`’ option, the program reads the description line (if it is present) and then the sequence (ignoring whitespaces) until it reaches `eof` or another description line. Then it performs the search and prints the output. Therefore, as is usually the case for FASTA format files, sequences spread over multiple lines can be used.

1.3.2 Output formats

The output is specified with the ‘`-o`’ option. See Section 1.2 [StemSearch Options], page 1. The two formats of output are:

`Standard`	The default. Produces a four line output for each stem containing the sequence identifier, coordinate information, score, E-value, sequence, and structure. As a subformat to the standard format, the loop can be ignored in the print of the sequence and structure.
`Short`	Produces a short version of the standard output containing score, E-value, and coordinate information only.

Standard format produces a four line output for each stem of the form

```
>identifier description
position: (i,j)   anchor: (k,l)   score: s    E-value: e
seq
str
```

The `identifier` is copied from the FASTA description line for the sequence, Section 1.3.1 [Input formats], page 2, `position` is the coordinates in the dynamic programming matrix where the score `s` is located and the anchor is the coordinates where the traceback terminates. This means that `i < k < l < j`, that `i` and `j` are the positions of the first and last letter respectively of the reported sequence `seq` within the searched sequence, and that `k` and `l` are the positions of the first and last letter of the hairpin loop. The score `s` is a normalized score and the E-value is the expected number of islands with a score `> s` that will be found in a sequence of the same length under the null model. See Section 3.3 [Normalization], page 8. The reported sequence, `seq`, is a small letter, RNA sequence and the stem structure, `str`, is in bracket notation. Since the structure is a stem, the sequence and structure is always of the form

```
[  5'-strand  ][hair-pin loop][  3'-strand   ]
[left-brackets]...............[right-brackets]
```

As the hair-pin loop can be large, especially when using the ‘`-nolp`’, Section 1.2 [StemSearch Options], page 1, it is possible, by ‘`-o nl`’, to get the standard output in the modified form

```
[  5'-strand  ]xx[n]xx[  3'-strand   ]
[left-brackets].......[right-brackets]
```

where `n` is the number of letters left out. The `x`’s represent the two first and two last letters in the hair-pin loop.

Short produces the following, one-line short version of the standard output:

```
identifier      i       j       k       l       s       e
```

A header line that reads

```
Seq.id    Pos.i      Pos.j      A.i       A.j       score   E.value
```

is printed first. This output can be read into `R` directly using the `read.table` command with `header=TRUE`.

2 Using StemSearch

StemSearch is made for searching long sequences or sequence databases for the occurrence of local stems or stem-loops. A main feature of the program is that it provides a normalized nat-score and corresponding E-value for each of the local stems that it finds. Those familiar with local alignment programs should feel comfortable with this, and the theoretical underpinning was also inspired by the results developed for local alignment.

The program locates one or more coordinate pairs (i, j) with $i < j$ in the sequence as closing coordinates for a stem. The maximal scoring stem with such a coordinate pair as closing coordinates is reported, and the dynamic programming algorithm used guarantees that the stem can not be extended without decreasing the score.

> **Note:** Even though StemSearch can report a secondary structure in bracket notation corresponding to each of the maximally scoring, local stems, it is **not** a secondary structure prediction program. This is important to understand in particular in relation to the parameters used, which are not energy-related but instead of a statistical/discriminative nature.

We sketch here three possible uses of StemSearch. **Finding local stems in several short sequences:** The file '`cel_miRNA`' contains 118 miRNA precursor sequences in FASTA format. We run `StemSearch` to locate one stem in each sequence using the parameter file '`cel.par`'.

```
>StemSearch  -f cel_miRNA -p cel.par
#StemSearch -f cel_miRNA -p cel.par
>cel-mir-245 MI0000321
Position: (8,92)          Anchor: (52,53)          Score: 6.608    E-value: 0.1308
caauguuggagagcuauuugcaagguaccuaauuguuugauuauugauucucaauugguccccuccaaguagcucuauugcauug
(((((.((((((((((((((..(((.(((.(((((...((..((..)))).))))))))..))).))))))))))).))))))))
>cel-lin-4 MI0000002
Position: (2,91)          Anchor: (42,44)          Score: 6.086    E-value: 0.2138
ugcuuccggccuguucccugagaccucaagugugaguguacuauugaugcuucacaccugggcucuccggguaccaggacgguuugagca
((((((((.((((.((((.((((((.((.(((((((((((...)..)))).)))))).)))).)))).))).).)))).)))...)))))
>cel-mir-258 MI0000335
Position: (20,78)         Anchor: (48,49)          Score: 8.396    E-value: 0.02167
uccuuuuacauauuuguugaaguuuucgcucgaauuuguggucgaauacuguagaagga
((((((((((((((((((((((..((((..))))))).))).)))))).))))))))))
>cel-mir-71 MI0000042
......
```

The output in standard format is four lines for each sequence in the file with a single stem-loop structure.

Finding all high-scoring stems in a segment of a long sequence: The file '`CHROMOSOME_I.fa`' contains the sequence for C.Elegans chromosome I in FASTA format. We want to locate all high-scoring stems in a segment from position 120000 to 350000 using a bandwidth of 200. The output below is slightly edited and lines are wrapped.

```
> StemSearch -f CHROMOSOME_I.fa -p cel.par -id -I 120000 350000 -L 200
#StemSearch -f CHROMOSOME_I.fa -p cel.par -id -I 120000 350000 -L 200
>I
Position: (336356,336554)    Anchor: (336454,336455)      Score: 54.02   E-value: 7.97e-19
guuagacaaaacuuuggcaaaacuuggauucaagcuuuaccaaggucuaacccaaguuucacccaacucuugccaaacuuuggcccaaac
uuuucuuauuucguuucaaauuugggccaaaguuuggcaagaguugggugaaacuuggguuagacuuugguaaagcuugaauccaaguuu
ugccaaagucuugccuaac
(((((.(((.(((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((.
.((((..(..).)....)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))).))).)))))
>I
Position: (315785,315983)    Anchor: (315884,315885)      Score: 53.48   E-value: 1.364e-18
aacucgggggauuggccccgcccacagaaccguggcuugcaauacgcccauuucugcaacugccgcacgguuuuaaaacuguauuuuucu
caauagagcgagaauuaacaagaaaaaauaauuuuaaaaccgugcggcaguugcagaaaugggcguauugcaagccacgguucugugggc
ggggccaaacucccgaguu
((((((((((.(((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((.(.(((((((((
...(.((((..)..))..)))))).)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))).))))))))))
>I
Position: (200924,201122)    Anchor: (201021,201022)     Score: 50.7    E-value: 2.205e-17
....
```

We may note that the sequences found seems to have a rather degenerate letter composition.

Finding the ten highest scoring stems in a simulated sequence We want to create a table of positions and scores of random stem-loops for the ten highest scoring stems in a simulated sequence of length 10000. We will use a first order Markov chain with transition probabilities estimated from the Human genome. We take the bandwidth to be 600 and a lower threshold to be 5.

```
> StemSearch -p hsa.par -I 0 10000 -L 600 -id -o s -r 10 -t 5
#StemSearch -p hsa.par -I 0 10000 -L 600 -id -o s -r 10 -t 5
Seq.id   Pos.i   Pos.j   Anc.i   Anc.j   score   E.value
"Random"        7618    7642    7629    7630    8.429   2.185
"Random"        8834    8918    8862    8885    7.838   3.946
"Random"        5721    5745    5732    5733    7.801   4.093
"Random"        4823    4853    4836    4837    7.72    4.438
"Random"        684     731     706     711     7.496   5.553
"Random"        4025    4059    4041    4047    7.325   6.59
"Random"        5670    5745    5707    5708    7.271   6.955
"Random"        1218    1279    1247    1250    7.151   7.845
"Random"        6178    6216    6191    6203    7.043   8.737
"Random"        2932    3008    2956    2977    7.028   8.869
```

Note that we have to set the lower threshold sufficiently small. The default is 10, and since none of the random stems obtain a score above 10, we will get an empty table if we don't set the threshold.

3 Parameters

The parameters used to score the stem is read from a parameter file. In contrast to secondary structure prediction programs, that rely on free energy considerations in the choice of parameters, `StemSearch` relies on parameters chosen for their discriminative power. This means that there is not one, globally “optimal” set of parameters, but instead a range of suitable parameters depending on the specific problem. If you for instance want to search for miRNA-precursors in C.Elegans and in Humans, two different parameter sets should be used.

3.1 Overview

The preferred name of a parameter file for `StemSearch` is ‘`foo.par`’. The central content of the file is a table of dinucleotide scores, that is, a table of scores $s(xy, zw)$ for (x, y) and (z, w) two consecutive nucleotide pairs in a stem. Formally, these scores correspond to stacking energies for free energy based, secondary structure prediction programs. The relation is, however, only formal, as the scores will take both positive and negative values with the most often occurring stacks, e.g. Watson Crick pairs, will receive a positive score and less frequent pairs will receive a negative score. See Section 3.2 [Organization], page 7.

The file also provides a specification of the internal loop, bulge and hairpin loop penalties. It is possible to use quite flexible, non-linear, hairpin loop penalty functions. See Section 3.4 [Loop Penalty], page 9, for details about the choice of loop penalties. In addition, the file contains parameters that are not used for the actual scoring of stems, but which defines a null model and the parameters needed for the normalization of the reported stem scores. The normalized scores are used to provide an assessment of the significance of the reported stems. It must be emphasized, however, that the normalization and hence the significance assessment is only valid if the null model used for the construction of the parameter file provides a reasonable approximation of the letter composition for the searched sequence. See Section 3.3 [Normalization], page 8.

For example, ‘`cel.par`’ can be used to search for stems similar to miRNA precursor stems within the C.Elegans genome. If it is used to search for stems in e.g. the human genome, it may first of all not be the most appropriate parameter set, but more important, the significance assessment is wrong due to a wrong normalization. If you want to use a parameter file on a sequence that you expect to have a different letter composition than the intentional one for the given parameter file, you can adapt the normalization parameters without changing the actual scoring parameters. Chapter 5 [Adaptation], page 11.

The default parameter file, ‘`default.par`’, is identical to the parameter file ‘`hum.par`’, which is based on a dataset of human miRNA precursors and the human genome for specification of the null model.

3.2 Organization

The default file, ‘`default.par`’, looks as follows:

```
###################################################################################
# Parameter file, default.par, for StemSearch. Created Tue Jun 24 00:50:13 2008 by
# the StemSearch R package 0.9.
###################################################################################
#
#Row and column order: aa ca ga ua ac cc gc uc ag cg gg ug au cu gu uu
#Stack and stack-entry transition scores:
* * * 0.8701 * * 1.1399 * * 3.0498 * 0.1305 0.441 * -0.6776 *
* * * 1.1223 * * 1.7685 * * 3.2484 * -0.4085 0.7871 * -0.4057 *
* * * 2.0804 * * 1.9847 * * 2.344 * 0.1644 1.2058 * * *
* * * 1.5391 * * 1.5054 * * 3.231 * 0.0042 1.0368 * 0.2406 *
* * * 1.2469 * * 1.9278 * * 2.7633 * * 1.3272 * -0.8464 *
* * * 1.5309 * * 1.378 * * 2.7754 * -0.5186 1.27 * -0.2281 *
* * * 1.8378 * * 1.8098 * * 1.8195 * -0.2658 1.5516 * 0.4393 *
* * * 1.3598 * * 1.4456 * * 2.7584 * -1.2287 0.9113 * -0.1273 *
* * * -0.311 * * 1.6933 * * 2.9101 * -0.4146 0.6268 * -1.105 *
* * * 3.039 * * 3.1212 * * 4.5859 * 1.4517 2.6342 * 1.9283 *
* * * 1.4394 * * 1.3494 * * 2.2966 * 0.8589 1.2071 * -0.0828 *
* * * 0.7321 * * 1.2664 * * 2.8432 * -0.2346 0.7028 * -1.3304 *
* * * 1.1065 * * 1.354 * * 3.1779 * -0.1612 1.0155 * 0.0383 *
* * * 1.0994 * * 1.6433 * * 3.2444 * 0.1691 0.9386 * 0.0177 *
* * * 1.8263 * * 1.8641 * * 2.9624 * 0.5389 1.7055 * 0.276 *
* * * 0.5763 * * 1.2465 * * 3.1043 * -0.2205 0.5051 * -0.2822 *

#Loop penalties

-4
-2.1
0
0
0
-0.4

#Null model

0.3307 0.1708 0.2377 0.2607
0.3537 0.2558 0.0487 0.3417
0.2891 0.2094 0.256 0.2455
0.222 0.2011 0.2462 0.3307

#Normalization

0.3535
-0.3099
1
0.3672
-2.7048
1
```

The first table specifies the allowed stacks in the RNA structure (those entries not marked by a '`*`') and the score for forming that particular stack. The file above allows canonical Watson-Crick pairs and GU-pairs. Next the loop penalties are defined. See Section 3.4 [Loop Penalty], page 9, provides details on the format. The null model is given by a table of transition probabilities for a first order Markov chain on the DNA/RNA alphabet. Last the parameters used for the normalization are given. The three first are used when the hairpin loop is penalized (the default) and the last three when the the option '`-nolp`' is specified, in which case there is no penalty on the hairpin loop.

3.3 Normalization

The output contains a score and an E-value. The score is a normalized score and referred to as the *nat-score*. We use the term *nat* to indicate that the normalization is based on the use of the natural logarithm. The choice is of course arbitrary. Dividing by log(2) (again with log being the natural logarithm) the score could be called a bit-score, as is preferred in the local alignment program BLAST.

For the normalization we need two parameters, λ and $\log(K)$. Internally, based on the parameters in the parameter file, the (maximal) score for a stem with closing coordinates (i, j) is $V(i, j)$. The reported nat-score for the stem is

$S(i, j) = \lambda V(i, j) - \log K.$

The nat-score is independent of the length of the sequence(s) searched as well as the bandwidth chosen. The value of the parameters λ and $\log(K)$ are specified in the parameter file and depends on whether the hairpin loop is penalized (the default) or not. Using the default parameter file with hairpin penalty $\lambda = 0.3535$ and $\log(K) = -0.3099$ and without hairpin penalty $\lambda = 0.3672$ and $\log(K) = -2.7048$.

The E-value for a stem with normalized score $S(i,j)$ gives the expected number of stems with a score exceeding $S(i,j)$ for a random sequence from the null model. The computation depends upon whether the hairpin loop is penalized. In both cases the E-value for the stem with closing coordinates (i,j) is computed as

$$E(i,j) = A\exp(-S(i,j)) = AK\exp(-\lambda V(i,j))$$

where A depends upon the length, n, of the sequence(s) searched and the bandwidth, L, chosen. If the hairpin loop is penalized,

$$A = n.$$

Thus in this case A does not depend upon the bandwidth. If the hairpin loop is not penalized,

$$A = n(L-1) - L(L-1)/2 = (L-1)(n - L/2).$$

In this case A is precisely the number of entries in the dynamic programming matrix that are used. We see that when L is small compared to n then A is roughly $n(L-1)$ and when $L = n$ then $A = n(n-1)/2$.

Usually L is much smaller than n, but if L is too small compared to n these computations of E-values may not be accurate. The parameter file supports a third parameter for each of the two situations – with or without hairpin loop penalty. This parameter is intended to be used to a currently unimplemented finite size correction. That is, a correction formula for A due to the fact that n and L are finite.

3.4 Loop Penalty

The program currently supports a penalization of the internal loops and bulges using an affine function and the hairpin loop using a combination of tabulation and linear extrapolation. For the internal loops and bulges the penalty function is

$$h(n) = \alpha + \beta(n-1)$$

for parameters $\alpha, \beta < 0$. The parameter α is a loop initiation penalty parameter and β is a loop extension penalty parameter. For the hairpin loop the small loop sizes are given an arbitrary penalty using a tabulation and for larger loop sizes than tabulated the penalty is extrapolated linearly from the two last tabulated values. That is, for n_1 tabulated values $g(0), g(1), \ldots, g(n_1)$ the extrapolation for $n > n_1$ is

$$g(n) = g(n_1) + (g(n_1) - g(n_1 - 1))(n - n_1)$$

In the default parameter file, ‘`default.par`’, this is given by the section

```
#Loop penalties

-4
-2.1
0
0
0
-0.4
```

which defines $\alpha = -4.0$, $\beta = -2.1$, $g(0) = g(1) = g(2) = 0$, $g(3) = -0.4$. The hairpin loop extrapolation has slope -0.4 and the entire hairpin loop penalty function is in this case piece-wise linear.

4 Algorithms

The algorithm that `StemSearch` employs is a dynamic programming algorithm using two dynamic programming matrices.

If we search a sequence $x(1)...x(n)$ let, for $i < j-1$,

$y(i,j) = s(x(i+1)x(j-1), x(i)x(j))$

where $s(xy, zw)$ for (x,y) and (z,w) dinucleotide pairs is the stack score from the table in the parameter file.

Initialization:

$V(i,i-1) = g(2)$ $V(i,i-2) = g(3)$

Recursion: $i < j-2$

$V(i,j) = \max\{V(i+1,j-1) + y(i,j), W(i+1,j-1) + y(i,j), g(i-j+1)\}$

$W(i,j) = \max\{W(i+1,j) + \beta, W(i,j-1) + \beta, V(i+1,j) + \alpha, V(i,j-1) + \alpha, g(i-j+1)\}$

Two traceback matrices are also stored and used for the final traceback of the stem responsible for the reported optimal stem(s). The stem with the maximal score may not be unique, but `StemSearch` will always report a single stem. In case of non-uniqueness the traceback algorithm chooses the stem to report.

If island declumping is used, two additional island matrices are stored to keep track of which island the given pair of coordinates belong to. The definition of the island matrix, I, corresponding to the score matrix V is that $I(i,j)$ is a pointer to the coordinate (i',j') such that if we start the traceback from (i,j) it will terminate at (i',j'). An island is by definition a set of coordinates pointing to the same traceback termination coordinate. By keeping track of the maximal score within each island the island declumped high scoring stems with a score exceeding a given threshold can be reported. The reported stems and stem-scores have two notable properties:

- Each stem has the maximal score within its corresponding island, and only one stem (the shortest maximal) is reported from each island.
- Two different, high scoring stems reported do not share any basepair.

The memory efficient algorithm is implemented by observing that for a given bandwidth, one only needs at any given time to keep a fraction of the matrices proportional to the square of the bandwidth. The tracebacks are then carried out "on the fly", as the matrices are filled instead of after the entire computation of the score matrices.

5 Adaptation

It is always possible to change the parameter file by hand to set stack score and penalty parameters or to change the null model. However, to get reliable nat-scores and E-values the normalization parameters should then also be updated.

Currently `StemSearch` comes with a small set of partly documented R-functions. The functions can be divided into three groups. The functions needed by the user are listed here. Some functions require the Bioconductor package Biostrings.

- Parameter IO-functions.
  - `read.parameters`
  - `write.parameters`
- Stack score computations.
  - `stack.trans.prob`
  - `null.trans.prob`
  - `update.stack.scores`
- Normalization parameter estimation.
  - `est.norm.lp`
  - `est.norm.nolp`

5.1 Parameter IO-functions

Two functions, `read.parameters` and `write.parameters` that can read and write parameters files in the format that `StemSearch` uses. The representation of the parameters in R is as a list with four entries;

- A matrix, `SScore`, of stack-scores
- a vector, `LoopPenalties`, of loop penalty parameters
- a matrix, `NullModel`, of transition probabilities for the null model
- a vector, `Norm`, with the normalization parameters.

5.2 Stack score computations

For the computation of stack scores as log-likelihood ratios using the first order Markov model one needs to specify a null model. The null model is specified by the matrix of transition probabilities. They can easily be estimated from one or more given DNA or RNA sequences, which can be regarded as representative null model sequences.

In addition to the matrix of null model transition probabilities we need a dataset with RNA structures. The function `ss.comp(parameters,file)` returns a parameter list, Section 5.1 [Parameter IO-functions], page 11,

5.3 Normalization parameter estimation

The normalization parameters can be estimated for the null model using simulations.

More details are found in Statistical models of local RNA stem-loop scores.

Assuming that the working directory is '`/StemSearch/R/`', the following example shows how to first compute the stack scores and the subsequently use the functions `est.norm.lp` and `est.norm.nolp`, see also '`ExScript1.r`', for estimating the normalization parameters.

```
source("parameterIO.r")
default.par <- read.parameters(file="../parameters/default.par")
st.tp <- stack.trans.prob(file="../data/hsa_miRNA_folds")
Ex1.par <- update.stack.scores(default.par,st.tp)
```

```
Ex1.par <- est.norm.lp(Ex1.par)
Ex1.par <- est.norm.nolp(Ex1.par)
write.parameters(file="Ex1.par",parameters=Ex1.par)
```

6 To do

The program is version 0.9 and is a beta-version. The following things will be changed before version 1.0 is released.

- The R-functions that are partly documented in Chapter 5 [Adaptation], page 11, will be embedded in a complete R package.
- The finite size corrections will be implemented.

A future version of `StemSearch` is expected to include the following components.

- More general secondary structures than stem-loops.
- Expansions of the R-package to contain R-functions for optimization of parameters and further statistical analyses.
- Heuristics to reduce computer time.

Index