PoSSuM software distribution

User manual

Version 2.0

Robert Homann, Michael Beckstette

June 8, 2009

Contents

1 Abstract

In biological sequence analysis, position specific scoring matrices (PSSMs) are widely used to represent sequence motifs. In this manual, we describe the *PoSSuM* software distribution implementing a new non-heuristic algorithm, called *ESAsearch* [BHGK06, BSH$^+$04], to efficiently find matches of such matrices in large databases. Our approach preprocesses the search space, e.g., a complete genome or a set of protein sequences, and builds an enhanced suffix array which is stored on file. The enhanced suffix array only requires 9 bytes per input symbol, and allows to search a database with a PSSM in sub-linear expected time. We also address the problem of non-comparable PSSM scores by implementing a method which allows to efficiently compute a matrix similarity threshold for a PSSM, given an E-value or a p-value. Our method is based on dynamic programming. In contrast to other methods it employs lazy evaluation of the dynamic programming matrix: it only evaluates those matrix entries that are necessary to derive the sought similarity threshold.

2 Introduction

Position specific scoring matrices (PSSMs) have a long history in sequence analysis (see [GME87]). A high PSSM score in some region of a sequence often indicates a possible biological relationship of this sequence to the family or motif characterized by the PSSM. There are several databases incorporating PSSMs, e.g., PROSITE [HSL$^+$04], PRINTS [ABF$^+$03], BLOCKS [HGPH00], or TRANSFAC [MFG$^+$03]. While these databases are constantly improved, there are only few improvements in the programs searching with PSSMs. E.g., the programs *FingerPrintScan* [SFA99], *BLIMPS* [HGPH00], and *MatInspector* [QFWW95] still use a simple straightforward $O(mn)$-time algorithm to search a PSSM of length m in a sequence of length n. The most advanced program in this field is *EMATRIX* [WNB00], which incorporates a technique called lookahead scoring. The lookahead scoring technique is also employed in the suffix tree based method of [DC00]. This method performs a limited depth first traversal of the suffix tree of the set of target sequences. This search updates PSSM scores along the edges of the suffix tree. Lookahead scoring allows to skip subtrees of the suffix tree that do not contain any matches to the PSSM. Unfortunately, the method of [DC00] has not found its way into a widely available and robust software system.

In this manual, we describe a new program for searching PSSMs. The overall structure of the implemented algorithm is similar to the method of [DC00]. However, instead of suffix trees we use enhanced suffix arrays, a data structure which is as powerful as suffix trees (cf. [AKO04]). Enhanced suffix arrays provide several advantages over suffix trees, which make them more suitable for searching PSSMs:

- While suffix trees require about $12n$ bytes in the best available implementation (cf. [Kur99]), the enhanced suffix array used for searching with PSSMs only needs $9n$ bytes of space.

- While the suffix tree is only computed in main memory, the enhanced suffix array is computed once and stored on file. Whenever a PSSM is to be searched with, the enhanced suffix array is mapped into main memory which requires no extra time.
- While the limited depth first traversal of the suffix tree suffers from the poor locality behavior of the data structure (cf. [GK95]), the enhanced suffix array provides optimal locality, because it is sequentially scanned from left to right.

One of our algorithmic contributions is a new technique that allows to skip parts of the enhanced suffix array containing no matches to the PSSM, implemented in *PoSSuM-search*. Due to the skipping, we achieve an expected running time that is sublinear in the size of the search space (i.e., the size of the nucleotide or protein database). As a consequence, the algorithm scales very well for large data sizes.

When searching with a PSSM it is very important to determine a suitable threshold for a PSSM match. Usually, the user prefers to specify a significance threshold (i.e., an E-value or a p-value) which has to be transformed into an absolute score threshold for the PSSM under consideration. This can be done by computing the score distribution of the PSSM, using well-known dynamic programming (DP, for short) methods, e.g., [Sta89, WNB00, Rah03, RMV03]. The program *PoSSuMdist* implements such an algorithm. Unfortunately, these methods are not fast enough for large PSSMs. For this reason, we have developed a new lazy evaluation algorithm that only computes a small fraction of the complete score distribution, speeding up the computation of the threshold significantly compared to standard DP methods. This makes our algorithm applicable for on-the-fly computations of the score thresholds.

In this manual, we first describe the file formats used by the *PoSSuM* software distribution, followed by descriptions of the programs *PoSSuMsearch*, *PoSSuMdist*, and *PoSSuMfreqs*. *PoSSuMsearch* and *PoSSuMdist* both generate output whose formats are also described in their respective sections. Finally, a few examples are shown how the *PoSSuM* software distribution can be used.

3 File formats

3.1 The *PoSSuM*-PSSM format

PoSSuMsearch and *PoSSuMdist* require PSSMs stored in an easy to read ASCII based file format, combining features supported by other PSSM formats.

3.1.1 PSSMs

In *PoSSuM*-PSSM format, each line begins with a *tag*, followed by *one* white space character, followed by some data for that tag. All strings are case-sensitive. There must be no white space before the beginning of any tag. Lines may be empty to separate things. Comments are allowed and introduced by a `#` character at the beginning of a line, the whole line is considered as a comment then.

These are the general rules. Now, a PSSM is defined in multiple lines, from which the first one reads

`BEGIN` *type*

followed by some other lines making up the PSSM, and the last line

`END`

indicating the end of a PSSM. The *type* can be one of `INT` or `FLOAT`, depending on the values used in the scoring matrix. If no floating point values occur in the matrix, then *type* should bet set to `INT` to speed up the search as integers can be processed much faster on most machines than floats can be.

Valid tags between a PSSM's `BEGIN` and `END` lines are (in any order):

`ID` The identifier of the PSSM. This tag is required.

`AC` The accession of the PSSM.

`DE` A description; any number of `DE` lines are allowed per PSSM. Multiple description lines are concatenated in order of occurence and separated by full stops when displayed by *PoSSuMsearch*.

`AL` An alphabet string. Each character in the string stands for one column of the PSSM, in given order and case-sensitive. The length of the alphabet string determines the width of the scoring matrix, that is, how many columns are expected to be defined.

`AP` A name of a predefined alphabet, one of `PROTEIN` or `DNA`. Specifying `DNA` here is equivalent to specifying an `AL` line with an alphabet string `ACGT`; `PROTEIN` is equivalent to `AL ACDEFGHIKLMNPQRSTVWY` (note that the exact order of characters is important, thus the explicit specification of the alphabet strings).

Only one of `AL` or `AP` can be used for a PSSM, of course, but one of them is required.

`TP` A predefined p-value cutoff, called trusted p-value cutoff. This is used when *PoSSuMsearch* is called with option `-trusted`.

`NP` A predefined p-value cutoff, called noise p-value cutoff. This is used when *PoSSuMsearch* is called with option `-noise`.

`LE` The "length" of the PSSM, that is the number of rows. This tag is required.

The values of a scoring matrix are defined using `MA` tags, one line per matrix row. There must be as many rows as specified in the `LE` line, each containing as many values as there are characters in the alphabet, in the order imposed by the alphabet. All values are given either as integers or reals as specified by the `BEGIN` line, separated by white space. After the first `MA` line, only `MA`, empty, or comment lines, or `END` are permitted.

The format requires matrices being *grouped* in so-called PSSM family models for later, optional chaining; see Section 3.1.2. A group of PSSMs must be surrounded by `BEGIN GROUP` and `END` lines. If your application context does not require grouping, just start the library file with a `BEGIN GROUP` line and end it with a final `END` after the `END` of the last PSSM. This declares all PSSMs in the file to belong to the same group.

Internally, each PSSM is identified by a tuple of *group identifier* and *group position*. The group identifier is the position of a group within the profile library file and the group position is the position of a PSSM within its group. Both quantities are counted up from 0 while the profile library file is read, where the group position counter is reset to 0 for every new group. Group identifiers and positions can be used for sorting the output (see description for `-sort` in Section 4.2) or for PSSM identification when post-processing the output by external programs.

A valid, artificial example for a PSSM library file is

```
BEGIN GROUP
BEGIN INT
ID Some matrix identifier
AC Some accession
DE A description describing the PSSM
DE Multiple description lines are possible
AP DNA
LE 3
#   A  C  G  T  was specified by "AP DNA"
MA  5 -1 -6  2
MA -4  4 -1 -5
MA  0 -3  3 -4
END

BEGIN FLOAT
ID Some other matrix identifier
DE Another description
AL AUCG
LE 2
#     A    U    C    G
MA  0.0 -3.5  3.2 -4.8
MA -4.2 -1.0  4.0 -5.8
END
END
```

3.1.2 PSSM family models

A PSSM family model is defined by the set of PSSMs in between `BEGIN GROUP` and `END GROUP` lines.

Valid tags between the PSSMs are (in any order):

`TL` This value gives the minimum length of a valid matching chain when using trusted p-value cutoffs (see *PoSSuMsearch* option `-trusted` and PSSM tag `TP`).

`NL` This value gives the minimum length of a valid matching chain when using noise p-value cutoffs (see *PoSSuMsearch* option `-noise` and PSSM tag `NP`).

The tags can be given before the first PSSM inside a group, after the last one, or anywhere in-between, but not inside a PSSM definition.

3.2 Frequency file format

A frequency file consists of simple character/value pairs, one pair per line. It serves for proper probability distribution calculation for E- and p-values for the PSSMs based on a specific input sequence, so its content should not just be a wild guess.

A line starts with a single character, followed by white space, followed by the relative frequency of that character in the input sequence. The relative frequency is a real number in the interval $[0, 1]$, the sum of all frequencies specified in one file should be 1.0, such that they constitute a sequence dependent character distribution.

Comments are allowed and introduced by a `#` character at the beginning of a line, the whole line is considered as a comment then. Empty lines are permitted.

A valid example for a uniform distribution on DNA data is

```
# This is a uniform distribution which is usually the wrong choice for real data.
A 0.25
C 0.25
G 0.25
T 0.25
```

Note that instead of "`T 0.25`" it would be equivalent to specify "`U 0.25`" or two separate lines reading

```
T 0.15
U 0.1
```

if the input sequence alphabet were to define "T" and "U" being the same. Frequencies of equivalent characters are summed up. See below for more information about symbol mappings.

See Section 6 on page 23 for the description of a tool for determining relative frequencies of characters from an input sequence.

3.3 Custom symbol mappings

To work on some sequence, *PoSSuMsearch*, *PoSSuMdist*, and *PoSSuMfreqs* all need to know the sequence's underlying alphabet. If the input sequence is an enhanced suffix array built by *mkESA*[1] or `mkvtree`[2], then this information is stored in the suffix array

[1]`http://bibiserv.techfak.uni-bielefeld.de/mkesa/`

[2]from the *Vmatch* package, see `http://www.vmatch.de/`

project. If the input sequence is a plain text format like FASTA, though, the user must either provide a symbol mapping file, or use the command line options `-dna` or `-protein` to specify a predefined (built-in) alphabet.

The format of symbol mapping files is the same as the format used in *Vmatch*, which is because the *PoSSuM* software distribution is based on the same libraries as *Vmatch*. Each line consists of a string of characters that should be regarded as equal. E.g., the file

```
aA
cC
gG
tT
*
```

defines a case-insensitive DNA alphabet. The last line specifies a group of special wildcard characters (only "`*`" in the example). Actually the wildcard is just an ordinary character treated in a special way internally. Note that the symbol mapping parser is quite picky and *requires* the last line to be terminated by a newline character.

Internally, all characters read from the input sequence are mapped to integers, and all characters that appear on the same line of the symbol mapping file are mapped to the same integer such that there is really no difference between them internally.

Use of symbol mappings is important for several reasons:

- Validation of the input sequence (invalid characters can be detected and therefore never occur during searching or in the output).

- Special treatment for case-(in)sensitivity, or more generally, character classes (like "t"="T"="u"="U"), is unnecessary because these cases are handled at the alphabet transformation level.

- The alphabet imposes an order on its characters by mapping them to integers (e.g., "a" and "A" may be mapped to 0, "c" and "C" to 1, etc.). Columns of PSSMs are ordered according to some alphabet, too (first column may stand for "A", second for "C", etc.). If the order of the input sequence alphabet is different from the PSSM's alphabet, then the columns of the PSSM can be reordered according to the order of the input sequence alphabet (otherwise, the user would be urged to provide his PSSMs with their columns in input sequence alphabet order), and this can be done with character classes being handled correctly (if the input sequence is encoded using the alphabet from the example above and the PSSM has some column for "U", then this column is read whenever a "t", "T", "u", or "U" appears in the input, additional columns for any of "t", "T", or "u" will be flagged as an error because of ambiguities).

4 *PoSSuMsearch*

4.1 Description

This is the main searching program. It implements *ESAsearch* for searching PSSMs in an enhanced suffix array, and the lazy dynamic programming evaluation algorithm for threshold derivation from E- and p-values. Additionally, other search algorithms *LAsearch* and simple search for plain text formats such like FASTA are implemented. As an alternative to the lazy dynamic programming evaluation algorithm, a precalculated probability distribution generated by *PoSSuMdist* (see Section 5 on page 21) can be used to derive PSSM thresholds.

4.2 Command line options

The searching program *PoSSuMsearch* is called as follows:

`possumsearch` [*options*]

Valid choices for *options* are

`-help`

Show options and terminate with error code 0.

`-db` *dbfile*

Name of a database to search in, which can be either an enhanced suffix array, or a FASTA, GENBANK, or EMBL file. The sequence must consist of characters over the alphabet as specified by the options `-dna`, `-protein`, or `-smap`, see below. This option is mandatory.

`-pr` *matrixfile*

Name of a profile library file. A “library” here is a collection of one or more PSSMs stored in the format as described in Section 3.1 on page 4. This option is mandatory.

`-protein`

This option is equivalent to the option `-smap` *mapfile* where *mapfile* stores exactly the following 21 lines:

```
L
V
I
F
K
R
E
D
A
G
```

```
S
T
N
Q
Y
W
P
H
M
C
XBZ*
```

This specifies an alphabet of size 20 with additional wildcard symbols on the last line. See Section 3.3 on page 7 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-dna`

This option is equivalent to the option `-smap` *mapfile* where *mapfile* stores exactly the following 5 lines:

```
aA
cC
gG
tTuU
nsywrkvbdhmNSYWRKVBDHM
```

This specifies an alphabet of size 4 with additional wildcard symbols appearing in the fifth line. See Section 3.3 on page 7 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-smap` *mapfile*

Specify the file storing the symbol mapping. If the given *mapfile* cannot be found in the directory where *PoSSuMsearch* is run, then all directories specified by the environment variable `MKVTREESMAPDIR` are searched. If defined correctly, this contains a list of directory paths separated by colons ("`:`").

If the file can't be found, an error message is reported and the program exits with error code 1. See Section 3.3 on page 7 or the *Vmatch* manual for a more detailed explanation of the format of symbol mapping files.

`-freq` *freqfile*

Specify the file storing the relative frequencies of characters in the input sequence. See Section 3.2 on page 7 for file format reference and Section 6 on page 23 for a description of *PoSSuMfreqs*, a simple program for generating frequency files from a database.

`-uniform`

If no frequency file is available, this option can be specified to assume characters being distributed uniformly. Note that this option is not meant for regular use—for accurate results, determining the real character distribution and specifying it via `-freq` is mandatory.

`-pdis` *distfile*
Specify the file storing a precalculated probability distribution as generated by *PoSSuMdist* for fast computation of E- and p-values for the PSSMs. The file must match the profile library specified by `-pr` and the alphabet of the input sequence. Because frequency information was already used when the distribution was precalculated by *PoSSuMdist*, options `-freq` and `-uniform` are prohibited when using this option. See Section 5.3 on page 23 and *PoSSuMdist* description for further information. Alternatively, `-lazy` can be used.

`-lazy`
Use lazy dynamic programming for fast computation of E- and p-values for the PSSMs as described in [BHGK06].

`-esa`
Search the PSSMs via *ESAsearch* as described in [BHGK06]. This option is only valid if the *dbfile* given to `-db` is an enhanced suffix array which must have been built by *mkESA* or `mkvtree` beforehand.

`-lahead`
Search the PSSMs via *LAsearch* as described in [WNB00, BHGK06]. This option can be used with all kinds of input sequences.

`-simple`
Search the PSSMs via the simple search algorithm as implemented in *FingerPrintScan*, *Blocksearch* [HH91], *BLIMPS*, *MatInspector*, and probably others. This option can be used with all kinds of input sequences, but should be used for debugging and benchmarking only due to its inferior efficiency.

`-eval` *E-value*
Specify E-value cutoff. This option must be combined with either `-lazy` or `-pdis`. E-value calculation is based on p-values, it is simply the p-value times database size. If combined with `-seqrange`, the database size is still assumed to be the size of the whole input sequence, not just the range's size. Use `-dbsize` to modify the database size for E-value calculation.

`-pval` *p-value*
Specify p-value cutoff. This option must be combined with either `-lazy` or `-pdis`.

`-trusted`
Use the trusted p-value cutoffs as specified in the PSSM library file. If this option is given, then a trusted p-value cutoff must be defined for each PSSM in the library file. This option must be combined with either `-lazy` or `-pdis`.

`-noise`
: Use the noise p-value cutoffs as specified in the PSSM library file, otherwise the same as `-trusted`.

`-mssth` *similarity*
: Specify a matrix similarity score (MSS) cutoff. MS-scores are PSSM scores rescaled to the interval $[0, 1]$ with the minimum reachable PSSM score corresponding to 0 and the maximum reachable PSSM score corresponding to 1. This scoring scheme is used in *MatInspector* and *Match* [KGR+03]. Note that because PSSM thresholds can be derived from *similarity* without use of probability distributions, they will not be calculated by default and E- and p-values will not be available in the results. If this information is required, also specify `-freq`, `-uniform`, or `-pdis` to tell *PoSSuMsearch* how the probability distributions for displaying E- and p-values should be obtained. If `-freq` or `-uniform` is used, a full probability distribution must be calculated for each matching PSSM which can be slow, using `-pdis` is the better choice then.

`-rawth` *threshold*
: Specify a raw, global threshold for all PSSMs. As PSSM thresholds are set directly and hence no probability distributions are required to do so, the same discussion about E- and p-values as for `-mssth` applies.

`-best` *k*
: Find the $k > 0$ best (meaning highest scoring) matches per PSSM. If there are less then k matches, only those are printed. This option can only be used in conjunction with `-esa` and `-lahead`. Searching with reduced alphabets on enhanced suffix arrays (options `-realpha` and `-esa`), however, does not work together with this option.

Note that since in general matches are found in different order for `-esa` and `-lahead`, their results may also slightly differ (e.g., this is the case when asking for, say, the best three matches, but there are actually a total of five best equally scoring matches in the database, then two of them never get reported—this is not a bug).

Also note that the threshold used for searching reported for each match (the value printed after “threshold” in human readable output, field 9 in tab delimited output, see Section 4.4 on page 19) is quite useless if `-best` is specified.

`-all`
: If a PSSM fails to code for the specified cutoff, e.g., if a p-value of 10^{-30} was specified, but the PSSM is only capable to code for a p-value of 10^{-20}, then that PSSM is not searched for by default and a warning is issued instead. If this option is specified, then in these cases the threshold is set to the maximum possible score the PSSM can yield, so it could match nonetheless (with a p-value higher than requested, though).

`-dbsize` *size*
: Assume the database size is *size* as basis for E-value calculation. This option af-

fects E-value calculation exclusively. By default, the original database size is used. Option `-seqrange` does not affect the default value.

`-realpha`

This option is specific to protein data/PSSMs. A problem with protein data is the large alphabet (when compared to DNA) involved which slows down the *ESAsearch* algorithm. A solution is to build an enhanced suffix array using a smaller custom alphabet which defines groups of amino acids as single representative characters, and to search the PSSMs on that reduced alphabet size index.

Usually PSSMs are converted to match the input sequence alphabet, such that an error would be issued when a protein PSSM was searched in an enhanced suffix array built with such a reduced alphabet. The problem then is that groups of distinct profile columns are mapped to the same sequence character representing a group, and when scoring that character there is no way to decide which of the PSSMs' columns should be used for scoring. So to handle this application case properly, the `-realpha` option must be specified. PSSMs are then read as if the input sequence was encoded by the standard protein alphabet, i.e., for enhanced suffix arrays as if they had been built using the `-protein` option, and for flat files (like FASTA) as if `-protein` had been passed to *PoSSuMsearch* (see description for `-protein` above). PSSMs are converted internally according to the reduced sequence alphabet and searched in the reduced sequence, the intermediate matches found are applied to the original PSSMs and original input sequence to calculate the correct match scores. Since reduced alphabets are specific to protein data, options `-rc` and `-fc` cannot be used together with `-realpha`, of course.

Note that for applying this option to an enhanced suffix array, it must have been built so to include the original input sequence (option `-g ois` for `mkesa`, option `-ois` for `mkvtree`). To specify a reduced alphabet, write a symbol map file as described in Section 3.3 on page 7 or in the *Vmatch* manual and pass that symbol map to the enhanced suffix array construction program (option `-a` for `mkesa`, option `-smap` for `mkvtree`). Using this option for flat files doesn't make much sense but is still supported, use *PoSSuMsearch*'s `-smap` option then. Also don't expect any speed-up when using reduced alphabets with the *LAsearch* algorithm.

All *PoSSuMsearch* options retain their original semantics even if `-realpha` is specified, e.g., `-pval` specifies a p-value cutoff for the PSSMs as if they were searched directly in the protein data, hence optionally passed frequencies or precalculated distributions must refer to the standard protein alphabet. See Section 7.1 for a complete example on how to use this option.

There is one drawback, though: if both `-esa` and `-best` are specified, then this option cannot be used. If `-best` is needed, use `-lahead` or a full alphabet size version of the index (i.e., no `-realpha`) instead.

`-sort` *keys*

Specify order in which matches should be reported. If this option is omitted, the output is not sorted in any special way. The *keys* argument is a string of keys the output is to be sorted by, priority in order of keys. Valid keys are

`i` PSSM identifier, sorted in lexical order. This is the string that is specified in the `ID` tag.

`a` PSSM accession, sorted in lexical order. This is the string that is specified in the `AC` tag.

`p` p-value of match, smallest first.

`e` E-value of match, smallest first.

`m` MSS of match, largest first.

`s` Score of match, largest first.

`n` Sequence number, smallest first.

`o` Position of match in sequence, smallest first.

`r` Group identifier.

`t` Group position.

`g` Group identifier and position, short for "`rt`".

E.g., to get results sorted by E-value in first place and sequence number in second place, specify "`-sort en`". Matches with both the same E-values and sequence numbers again are not sorted in any special way.

Note that a pair of group identifier and group position (sort key "g") always identifies exactly one PSSM, but a PSSM identifier together with its accession (sort keys "`ia`" or "`ai`") may not because multiple PSSMs with equal identifiers and accessions can be specified. If unsure, use "`iag`" instead of "`ia`" or "`aig`" instead of "`ai`" to make sure to have PSSMs with both the same identifier and accession separated in the output. Also note that specification of "`gia`" or "`gai`" is equivalent to only specifying "g" because if PSSMs are already sorted by group identifier and position in first place, then further sorting by PSSM identifier or accession is not possible (read: unnecessary). In other words, specification of "g" just separates matches by PSSM in order of occurence in the profile library, "`iag`" or "`aig`" arrange them in alphabetical order and then make sure to have distinct PSSMs with equal identifier and accession strings being separated.

`-modsearch` *count*

Search with PSSM family models, and report matching chains for up to *count* different models per sequence. The *count* best matching chains in a sequence are shown; if multiple chains are found for some model in the same sequence, then only the highest-ranking chain might be shown for that particular combination of model and sequence. That is, up to *count* chains are shown per sequence, and each chain belongs to a different model.

Typically, this option is combined with one of `-trusted` or `-noise`, and possibly also with `-mclen` or `-mrclen`. See also Sections 7.1.4 and 7.1.5.

`-seqclass` *count*

Search with PSSM family models, and for each model, report the best matching chains in up to *count* different sequences. The *count* best matching chains for a model are shown; if multiple chains are found for some model in the same sequence, then only the highest-ranking chain might be shown for that particular combination of model and sequence. That is, up to *count* chains are shown per model, and each chain belongs to a different sequence.

Typically, this option is combined with one of `-trusted` or `-noise`, and possibly also with `-mclen` or `-mrclen`. See also Sections 7.1.4 and 7.1.5.

`-mclen` *len*

Do not report chains shorter than *len*, i.e., make sure that a match to a PSSM family model consists of at least *len* different single PSSM matches defined in that model. By default, *len* is set to 1 so that there is no restriction on the minimum length. Note that this parameter can be defined for each individual PSSM family model by inserting `TL` and/or `NL` lines into the model definition. When given, this option overrides the value specified on `TL` and `NL` lines.

`-mrclen` *ratio*

Like `-mclen`, this option restricts the minimum chain length, but the restriction depends on the size of the PSSM family models; chains of length l found for a PSSM family model consisting of k PSSMs are accepted only if $\frac{l}{k} \geq \mathit{ratio}$ holds.

`-format` *fmt*

Specify output format, where *fmt* is one of

`human`	a human readable multiline format;
`cisml`	CisML [HW04], an XML-based format;
`tabs`	tab delimited output (see Section 4.4 on page 19);
`fasta`	FASTA format (see Section 4.5 on page 20); or
`null`	no output.

`-fn`

Search on forward strand (default). This option works with any alphabet and replaces the `-fwd` option of previous versions of *PoSSuMsearch*. See also `-rc`, `-rn`, and `-fc`. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions.

`-rc`

Search for reverse complementary matches. This option disables the default of searching on the forward strand—specify an extra `-fn` to search on both strands, or use option `-2` or even `-4`. This is a DNA specific option, i.e., the input alphabet must be a DNA alphabet (or in better words, a DNA *compatible* alphabet, not necessarily generated via the `-dna` option of *PoSSuMsearch* or `mkvtree`), and, of course, the PSSMs should encode DNA motifs.

The reason for the DNA specificity is that internally the PSSMs' columns are exchanged ("A"-column with "T"- or "U"-column and "C"-column with "G"-column), their rows are reversed in order, and then a usual search in forward direction is done. Because *PoSSuMsearch* supports arbitrary alphabets to be used for both, input sequences and PSSMs, the column exchange must be done carefully, the compatibility of both alphabets must be checked, and character classes must be recognized ("T" and "U" could be distinct characters in the input sequence). Note that *PoSSuMsearch* does not attempt to recognize whether the alphabets are strictly DNA or not, it just tries to find those columns unambiguously labelled with characters from the DNA alphabet and exchanges them. If columns cannot be exchanged for some reason, the program will tell so and exit with error code 1. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions. See also option `-ncompl`.

A more general, non-DNA specific approach could be implemented by requiring the user to explicitly specify the columns to be exchanged instead of autodetecting them, but it would be harder to use without any gain in practical use. Expect no problems when using option `-dna` throughout.

`-rn`
: Search for reverse matches. The PSSMs' rows are reversed in order and a usual search in forward direction is done. This option works with any alphabet. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions.

`-fc`
: Search for complementary matches. Alike `-rc`, this is a DNA specific option, i.e., the input alphabet must be a DNA alphabet and the PSSMs should encode DNA motifs. Internally, the PSSMs' columns are exchanged as with `-rc` and a usual search in forward direction is done, but the PSSMs' rows are *not* reversed in order. This option replaces the (misnamed) option `-rev` of previous versions of *PoSSuMsearch*. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions. See also option `-ncompl`.

`-2`
: Short for `-fn` and `-rc`. If searching for forward and reverse complementary matches, this option is usually what you want. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions.

`-4`
: Short for all of `-fn`, `-rc`, `-rn`, and `-fc`. See Section 4.3 on page 18 and Figure 1 for a more detailed explanation on the options concerning search directions.

`-ncompl`
: By default, the matching sequences for matches on the complementary strand (options `-rc` and `-fc`) are printed out complemented, i.e. *not* as they appear in

the input sequence. E.g., `ACG` is a matching sequence to the first PSSM in Section 3.1.1 on page 6 with a threshold of 12 on the forward strand. A reverse complementary matching sequence to the same PSSM with threshold 12 is `CGT`, possibly occuring somewhere else on the reverse complementary strand. In this case, the string that really occurs in the input sequence is `GCA` since the input sequence is always considered to be the forward strand. Without this option, the matching sequence will be printed as `CGT`, and as `GCA` otherwise. See Section 4.3 on the next page and Figure 1 for a more detailed explanation on the options concerning search directions.

`-seqrange` *range*

When using `-lahead` or `-simple`, search only in a range of sequences, not all. The *range* is specified as `min:max` pair, including the borders. A range of `30:39` will search only in sequences 30 to 39, including sequence 30 and 39. Sequence numbers always start at 0.

`-mult` *factor*

For probability distribution calculation, the values of the scoring matrices are scaled by the value of *factor*. The default value of *factor* is 1.0. To speed up calculation of E- and p-values at the price of loss of precision and to reduce disk space when writing the distribution to file using *PoSSuMdist*, choose a value from interval $(0, 1)$. This effects in a compression of PSSM score ranges and thereby a reduction of computation time for the probability distribution calculation. E.g., a value of 0.1 speeds up the probability distribution calculation for a PSSM by approximately a factor of 10 (because the PSSM's score range is only a tenth of the original range then), but this also means that every 10 consecutive score values achievable by a PSSM are condensensed into one single p-value, which is likely to produce false positives and false negatives.

To enhance precision in some cases, choose a value greater than 1, resulting into an expansion of score ranges. Since floating point PSSMs must be rounded to integers for our dynamic programming method, a value greater than 1 can help getting better E- and p-values for PSSMs containing very small values.

This option should be used with great care. PSSMs with smaller score ranges are more prone to rounding errors than those with larger ranges. Larger score ranges result into considerably more space consumption by the probability distribution calculation.

`-qm`

Do not print status messages to `stderr`.

`-qw`

Do not print warnings to `stderr`.

`-q`

Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm`

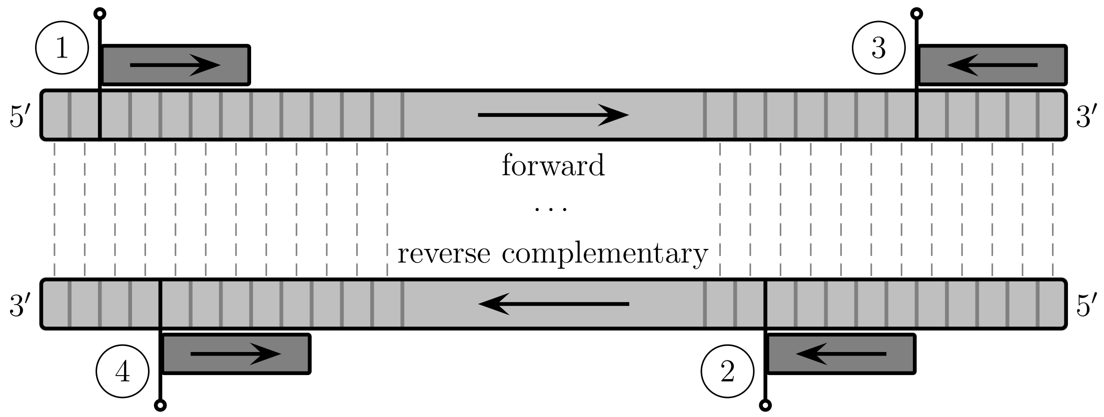

Figure 1: The search directions supported by *PoSSuMsearch*. If the data is DNA, then there are four cases to consider, namely searching with a PSSM (1) in forward direction on forward strand (option `-fn`, default), (2) in reverse direction on reverse complementary strand (option `-rc`), (3) in reverse direction on forward strand (option `-rn`), and (4) in forward direction on reverse complementary strand (option `-fc`). Note that the arrows denote directions in a biologically correct sense since DNA is commonly read from 5′- to 3′-end. The lower strand in the figure is the *complement* to the upper strand, not the reverse. In case of non-DNA data, the lower strand does not exist, and so then do cases (2) and (4) not.

and `-qw`. Matches are still written to `stdout`.

`-version`
Show program version. Option `-version` is a synonym for this option.

The program will terminate silently with error code 0 if no error occured. On error, it will terminate with error code 1 and print the error to `stderr`.

4.3 Search directions

On any data, *PoSSuMsearch* supports searching with PSSMs in forward and reverse directions in order to find matches to the provided PSSMs and their reverse. Additionally, on DNA data, searching on the complementary strand is possible. This sums up to a total of four cases, see Figure 1 for reference. *PoSSuMsearch* offers command line switches to choose any combination of these, the default is `-fn` (search for matches on the forward, non-complementary strand). Specifying one of the other options disables `-fn`, so `-fn` must be specified explicitly if this is also required. Cases (1) and (3) can be used independently of the alphabet, cases (1) and (2) are most commonly used on DNA data. Whether or not cases (3) and (4) are especially useful in practice is arguable, though, still we provide options for these since the user usually knows better what he wants than we do. For convenience, option `-2` can be used to search for matches falling into the categories of cases (1) and (2). Use option `-4` to search for matches in all four possible ways.

Within *PoSSuMsearch*, only the forward, non-complementary strand is known, as provided by the user, represented by the upper strand in Figure 1. Searching in reverse direction is implemented by reversing the PSSM's row order, covering cases (2) and (3). Searching on the complementary strand is accomplished by alphabet transformation, i.e., by permuting the PSSM's columns according to Watson-Crick base pairing, covering cases (2) and (4). The sequence itself remains unchanged. Since a match does not imply the existence of corresponding matches on the complementary strand nor does it imply the existence of reverse matches, *PoSSuMsearch* must search explicitly for every possible case, hence passing option `-4` results in roughly a 4-fold time consumption for the search phase compared to when searching for a single case.

There are various possible ways one could think of to report reverse and/or complementary matches. In *PoSSuMsearch*, matches are always reported with respect to the forward strand since this is the only sequence that is explicitly represented in the computer and stored in the database. The left-most position of a match relative to the beginning of the sequence the match occurs in is shown as the start position, starting with position 0, regardless of search direction. E.g., the matching positions for the instances of cases (1) to (4) in Figure 1 would be reported as marked as 2, $n - 10$, $n - 5$, and 4, respectively, where n is the number of characters in the sequence. The matching sequence is printed in forward direction as occuring in the database, so in particular, not in reverse even if the match was found on a reverse strand. For matches on the complementary strand, the *complementary sequence* is printed as matching sequence. To get the matching sequences as they occure in the database (i.e., on the non-complementary forward strand), specify option `-ncompl`.

4.4 Tab delimited output format

For each match a single line containing 18 entities is output. They are, separated by tabulators

1. matched PSSM's identifier (`ID`),
2. matched PSSM's accession (`AC`),
3. matched PSSM's description (`DE`, multiple lines separated by "`. `"),
4. group identifier, which is the position of the group in the profile library file that the PSSM belongs to, starting at 0,
5. position of PSSM within its group, starting at 0,
6. start position of the match with respect to the beginning sequence the match occurs in, starting at 0 (see Section 4.3 for more details on this),
7. length of the match,

8. search direction (“`fn`” for forward non-complementary, “`rc`” for reverse complementary, “`rn`” for reverse non-complementary, “`fc`” for forward complementary, see Section 4.3),
9. threshold used for searching (value is useless if `-best` was specified since there is no specific predefined threshold in that case),
10. match score,
11. minimum score the PSSM can achieve,
12. maximum score the PSSM can achieve,
13. p-value,
14. E-value,
15. MSS,
16. matching sequence number, starting at 0,
17. matching sequence description (multiple lines separated by “`. `”), and
18. matching substring (see also Section 4.3 and *PoSSuMsearch* option `-ncompl`).

Note that tabulators in string entities (descriptions, identifier, etc.) are *not* filtered and may therefore cause problems when parsing an output containing such a string.

If the probability distribution is not available during match evaluation (maybe because `-mssth` or `-rawth` was specified but neither `-pdis` nor `-freq`), the fields for E- and p-value will still be there, but left empty. The same is true for missing information due to lack of `AC` or `DE` tags in the PSSM specification or missing sequence description. A parser reading tab delimited *PoSSuMsearch* output should take this into account.

4.5 FASTA format

The FASTA format is very simple; it contains lines starting with a “`>`” character immediately followed by a sequence description, and lines containing sequence data attibuted to the nearest description line preceeding the sequence.

When selecting FASTA as output format for *PoSSuMsearch*, please beware of the fact that FASTA contains sequences only, and that whole sequences are reported as matches as they are stored in the sequence database. In particular, there will be no information about the nature of the matches found, their positions, their scores, etc., and a sequence will be reported multiple times when multiple matches are found in that sequence.

5 *PoSSuMdist*

5.1 Description

This is a supplementary program for *PoSSuMsearch*. It is used to precalculate the complete probability distribution which is used to derive PSSM thresholds from E- and p-values. This can be useful if the same PSSM library is searched multiple times and lazy evaluation within *PoSSuMsearch* should be circumvented. Note that a complete probability distribution may require a significant amount of space on file and can take a long time to calculate.

In *PoSSuMsearch*, it is not possible to use a generated probability distribution file with PSSM libraries different from that given to *PoSSuMdist* to generate the distribution. It is not even possible if PSSMs are only rearranged within, deleted from or inserted into that PSSM library. Hence the use of precalculated probability distributions decreases the grade of flexibility in favor of speed.

5.2 Command line options

The program for probability distribution calculation *PoSSuMdist* is called as follows:

`possumdist` [*options*]

Valid choices for *options* are

`-help`
: Show options and terminate with error code 0.

`-pr` *matrixfile*
: Name of a profile library file. A “library” here is a collection of one or more PSSMs stored in the format as described in Section 3.1 on page 4. This option is mandatory.

`-protein`
: Generate a probability distribution file for an input sequence encoded by the standard protein alphabet as described in the description of *PoSSuMsearch* option `-protein`.

`-dna`
: Generate a probability distribution file for an input sequence encoded by the standard DNA alphabet as described in the description of *PoSSuMsearch* option `-dna`.

`-smap` *mapfile*
: Generate a probability distribution file for an input sequence encoded by the symbol mapping defined in *mapfile*. See the description of *PoSSuMsearch* option `-smap` for more details.

`-db` *dbfile*
: Generate a probability distribution file for the enhanced suffix array *dbfile*. This option is used for convenience to just read the symbol mapping from an enhanced suffix array. The enhanced suffix array itself is not read.

`-freq` *freqfile*
: Specify the file storing the relative frequencies of characters in the input sequence. See Section 3.2 on page 7 for file format reference and Section 6 on the facing page for a description of *PoSSuMfreqs*, a simple program for generating frequency files from a database.

`-uniform`
: If no frequency file is available, this option can be specified to assume characters being distributed uniformly. Note that this option is not meant for regular use—for accurate results, determining the real character distribution and specifying it via `-freq` is mandatory.

`-pdis` *distfile*
: Specify the name of the output file storing the precalculated probability distribution. This file can be used later by *PoSSuMsearch* only in conjunction with the profile library specified by `-pr` and input sequences encoded by the specified alphabet. Note that the *factor* of `-mult` given to *PoSSuMdist* will be encoded into *distfile*, hence it can't be changed by a later invokation of *PoSSuMsearch*. See Section 5.3 on the next page for format description and more information.

`-mult` *factor*
: The values of the scoring matrices are always scaled by the value of *factor*, which defaults to 1.0. See the description of the *PoSSuMsearch* option `-mult` on page 17 for more details.

`-qm`
: Do not print status messages to `stderr`.

`-qw`
: Do not print warnings to `stderr`.

`-q`
: Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm` and `-qw`.

`-version`
: Show program version. Option `-version` is a synonym for this option.

The program will terminate silently with error code 0 if no error occured. On error, it will terminate with error code 1 and print the error to `stderr`.

5.3 Format of the probability distribution file

A probability distribution file contains the complete probability distributions of all PSSMs in the profile library in order of occurence, written as binary stream and compressed via *zlib* (see `http://www.gzip.org/zlib/`). This data is architecture dependent and can't be exchanged between different architectures because of different byte orders and eventually different sizes of data types. Exchanging probability distribution files between incompatible architectures will yield unpredictable results, from false matches to program crashes.

The distributions are written one after the other, containing

- *minscore*, the absolute of the minimum achievable score (unsigned integer),
- *maxscore*, the absolute of the maximum achievable score (unsigned integer),
- the absolute of the global matrix minimum multiplied by the matrix height (unsigned integer),
- the factor specified by `-mult` when the distribution was calculated (double), and
- an array of p-values of length $maxscore - minscore + 1$, ranging from *minscore* to *maxscore* (doubles).

All scores are scaled by the factor specified by `-mult` and rounded to integers. So, the general file layout is simply

1. *minscore*	1. *maxscore*	1. global minimum	1. factor	1. array of p-values
2. *minscore*	2. *maxscore*	2. global minimum	2. factor	2. array of p-values
...	...	...	...	...

in uncompressed form. To save space, *zlib* is used to compress the output transparently. Use *gunzip* (see `http://www.gzip.org/`) for manual decompression if needed.

6 *PoSSuMfreqs*

6.1 Description

For accurate results in score threshold calculation from significance thresholds, the relative frequencies of characters in the database need to be known. *PoSSuMfreqs* is a simple program to determine those frequencies and write them to `stdout` in the format described in Section 3.2 on page 7.

6.2 Command line options

The program for determining relative frequencies of characters *PoSSuMfreqs* is called as follows:

`possumfreqs` [*options*]

Valid choices for *options* are

`-help`
: Show options and terminate with error code 0.

`-db` *dbfile*
: Name of a database to determine the relative frequencies of characters from, which can be either an enhanced suffix array, or a FASTA, GENBANK, or EMBL file. The sequence must consist of characters over the alphabet as specified by the options `-dna`, `-protein`, or `-smap`, see below. This option is mandatory.

`-protein`
: Generate a frequency file for an input sequence encoded by the standard protein alphabet as described in the description of *PoSSuMsearch* option `-protein`.

`-dna`
: Generate a frequency file for an input sequence encoded by the standard DNA alphabet as described in the description of *PoSSuMsearch* option `-dna`.

`-smap` *mapfile*
: Generate a frequency file for an input sequence encoded by the symbol mapping defined in *mapfile*. See the description of *PoSSuMsearch* option `-smap` for more details.

`-qm`
: Do not print status messages to `stderr`.

`-qw`
: Do not print warnings to `stderr`.

`-q`
: Quiet, do not print anything to `stderr`. This is equivalent to specifying both `-qm` and `-qw`.

`-version`
: Show program version. Option `-version` is a synonym for this option.

7 Using the *PoSSuM* software distribution

7.1 Examples

7.1.1 Basic operation

Build an enhanced suffix array `sprot` from FASTA file `sprot.fas` containing protein data, using the predefined protein alphabet. To save disk space, not all possible tables are built, only those required by *PoSSuMsearch*.
Using *mkESA*:

```
$ mkesa -d sprot.fas -p sprot -b protein -g tis,suf,lcp,skp -v
```

Using `mkvtree`:

```
$ mkvtree -db sprot.fas -indexname sprot -protein -tis -suf -lcp -skp -v
```

Generate character distribution from the previously built enhanced suffix array `sprot` and write it to `frequencies.txt`, then search all PSSMs in `profiles.txt` in `sprot`, deriving PSSM thresholds via the lazy dynamic programming evaluation algorithm using an E-value of 10^{-15} and the character distribution from `frequencies.txt`. The size of the database and its alphabet are known from the `sprot` project.

```
$ possumfreqs -db sprot > frequencies.txt
$ possumsearch -pr profiles.txt -db sprot -esa -eval 1e-15 -lazy\
 -freq frequencies.txt
```

Precalculate probability distribution of PSSM library file `profiles.txt`, write distribution to `dist.gz`, and search all PSSMs with a p-value cutoff of 10^{-20} or less in FASTA file `sprot.fas` via *LAsearch*, which contains protein data with a character distribution stored in `frequencies.txt`. The alphabet of the database must be explicitly specified and match the alphabet used when `dist.gz` was created (`-protein` here). Perform the same search again on the previously built enhanced suffix array via *ESAsearch* and then via simple search to feel the difference.

```
$ possumdist -pr profiles.txt -protein -freq frequencies.txt -pdis dist.gz
$ possumsearch -pr profiles.txt -protein -db sprot.fas -pval 1e-20 -lahead\
 -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot -pval 1e-20 -esa -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot -pval 1e-20 -simple -pdis dist.gz
```

For a working example, take a look into the `share/PoSSuM/examples/` directory of the *PoSSuM* software distribution. Included are two Bourne shell scripts (`demo_mkesa.sh`, `demo_mkvtree.sh`), a FASTA sequence file containing two sequences (`demo.fas`), and 17 PSSMs in *PoSSuM*-PSSM format (`demo.lib`). The shell script builds an enhanced suffix array from the FASTA file and performs some searches in the enhanced suffix array and in the FASTA file. The output of the shell script can be redirected to a file and compared to the included file `results.txt` found in the examples directory.

7.1.2 Reduced alphabets

Here is a complete example on how to use the `-realpha` option of *PoSSuMsearch* to speed up *ESAsearch*. We use a custom symbol map, called `prot8.map`, containing eight character classes to build an enhanced suffix array `sprot8` from the protein data in FASTA file `sprot.fas`. The content of `prot8.map` reads (taken from [PNW99])

```
G
ALM
VI
ND
P
YFWC
KRQE
STH
BXZ*
```

PoSSuMsearch is used to search the profiles in `profiles.txt` in `sprot8`. Note that the distribution data previously generated can be used here again. Remember that internally the PSSMs are treated as if the input sequence was encoded by the standard protein alphabet, so the probability distribution must be, too. Hence when a precalculated probability distribution should be used in conjunction with `-realpha`, it must always refer to the standard protein alphabet. All relevant commands are shown below.
Construct enhanced suffix array using a custom symbol mapping:

```
$ mkesa -d sprot.fas -p sprot8 -a prot8.map -g tis,ois,suf,lcp,skp -v
```

or

```
$ mkvtree -db sprot.fas -indexname sprot8 -smap prot8.map -tis -ois -suf -lcp -skp -v
```

Then:

```
$ possumfreqs -db sprot.fas -protein > frequencies.txt
$ possumdist -pr profiles.txt -protein -freq frequencies.txt -pdis dist.gz
$ possumsearch -pr profiles.txt -db sprot8 -pval 1e-20 -esa -realpha -pdis dist.gz
```

The results should be the same as in the example before, except for the ordering.

7.1.3 Computing scores for all substrings

If the scores of all substrings of a sequence need to be known, the best method is to use simple search or *ESAsearch* in conjunction with a threshold of the PSSMs' *minscore*. To achieve this, use something like the following.

```
$ possumsearch -pr profiles.txt -db sprot -esa -mssth 0
```

Do not use *LAsearch* in this case. The reason to prefer simple search over *LAsearch* here is that every substring must be read to its full length anyway, simple search avoids the extra overhead of *LAsearch*. Still, *ESAsearch* might be even more preferable.

7.1.4 Sequence classification with PSSM family models

One of the innovations in the new version 2.0 of *PoSSuMsearch* is the support for PSSM family models. PSSM family models consist of several PSSMs in a defined order. This order reflects the order of occurence of the alignment blocks, from which the single PSSMs are determined, in the multiple alignment. When employing PSSM family models, *PoSSuMsearch* first finds all matches of the single PSSMs using, e.g., algorithm *ESAsearch*. Subsequently, these matches are assembled into chains using an efficient fragment chaining algorithm, and high scoring chains are reported to the user.

For sequence database searching with PSSM family models, *PoSSuMsearch* can be used in two modes of operation, namely *modsearch* and *seqclass* as described in Section 4.2 (options `-modsearch` and `-seqclass`). Mode *modsearch* mimics the *modus operandi* of program *hmmsearch* from the *HMMer* package, whereas mode *seqclass* is comparable to program *hmmpfam*.

To determine for each sequence in enhanced suffix array `trembl` the best matching chain for a PSSM family model stored in file `TIGR0001.pssmfm` fulfilling the model's trusted p-value cutoff (specified via the TP tag in the model file) and required minimal chain length (specified via the TL tag in the model file) using algorithm *ESAsearch*, call *PoSSuMsearch* as follows.

```
$ possumsearch -db trembl -pr TIGR0001.pssmfm -esa -lazy -freq frequencies.txt\
 -modsearch 1 -format human -trusted
```

Similarly, to compute for each model the best matching chain in (up to) 10 different sequences fulfilling the models noise p-value cutoff (specified via the NP tag in the model file) and minimal chain length (specified via the NL tag in the model file) match constraint, type

```
$ possumsearch -db trembl -pr TIGR0001.pssmfm -esa -lazy -freq frequencies.txt\
 -seqclass 10 -format human -noise
```

P-value cutoffs and minimal chain lengths stored in the PSSM family model file can be overridden with parameters `-pval` and `-mclen`, respectively. To compute all matching chains with a minimum length of 4 and a p-value cuttoff per PSSM of 10^{-5} for model `TIGR0001.pssmfm` in the enhanced suffix array `trembl`, call *PoSSuMsearch* as follows.

```
$ possumsearch -db trembl -pr TIGR0001.pssmfm -esa -lazy -freq frequencies.txt\
 -modsearch 1 -format human -pval 1e-5 -mclen 4
```

7.1.5 Speeding up *hmmsearch* with *PSfamSearch*

Profile hidden Markov models (pHMMs) are currently the most popular modeling concept for protein families. They provide very sensitive family descriptors, and sequence database searching with models from major pHMM collections has become a standard task in today's sequence analyses and genome annotation pipelines. On the downside, database searching for pHMMs with programs like *hmmsearch* or *hmmpfam* is computationally expensive.

The new *PoSSuMsearch* can be used to speed up *hmmsearch* by search space reduction with PSSM family models. This procedure, called *PSfamSearch*, is a two step approach that works as follows. First, we filter the search space (i.e., the sequences to be searched) as described in Section 7.1.4 (but with the output format set to `fasta`), using a PSSM family model constructed from a multiple alignment of a protein family. In the second step we apply the more time consuming *hmmsearch* with a pHMM corresponding to this family on the reduced sequence set only.

This two steps are combined in the Perl script `psfamsearch.pl`. Assume you have a PSSM family model stored in file `TIGR00001.pssmfm`[3], the corresponding pHMM in file `TIGR00001.hmm` in *HMMer 2* format, an enhanced suffix array `trembl`, and a frequency file `frequencies.txt`. Then, the following call performs the two step procedure described above.

```
$ psfamsearch.pl -pssmfm TIGR00001.pssmfm -hmm TIGR00001.hmm -index trembl\
 -freq frequencies.txt -trusted -seqtmp tmp.out
```

With parameter `-trusted`, trusted p-value cutoffs and required minimal chain length constraints from the PSSM family model file are used. Sequences passing the filter are written to temporary file `tmp.out` and serve as input for *hmmsearch* in step two. The final output of *hmmsearch* is written to `stdout`.

To use p-value cutoffs matching a pHMM's noise cutoff and write the output to file `TIGR000001.results`, type

```
$ psfamsearch.pl -pssmfm TIGR00001.pssmfm -hmm TIGR00001.hmm -index trembl\
 -freq frequencies.txt -noise -seqtmp tmp.out -output TIGR000001.results
```

7.2 Messages and warnings

Both programs, *PoSSuMsearch* and *PoSSuMdist*, print progress messages to inform the user about what the program is doing, and warnings if problems are detected. Messages and warnings are always printed to `stderr` and are therefore separated from the matches, which are always printed to `stdout`.

Messages can safely be ignored, but if any warnings occur, you should read them as they could point you to some undiscovered problem. Ignore them only if you know they are harmless.

Most warnings are self-explanatory, but there are some that can be confusing:

- Warnings concerning PSSM library files:

[3]PSSM family models with cutoffs adjusted to match *HMMer* trusted and noise cutoffs for the first 20 protein families of the TIGRFAM Rel. 6.0 database can be found on the *PoSSuMsearch* website.

Character ‘x’ is undefined/defined as a wildcard/defined as the special separator character in the input alphabet.
A PSSM defines a column for some character which is not defined as a valid character in the database alphabet and will therefore never contribute to a match score. Those columns are ignored for probability distribution calculation and during matching. This is a common warning for e.g., protein PSSMs defining a column for ‘B’ which is a wildcard in the predefined protein alphabet.

Character ‘x’ may occur in the input sequence, but is not defined for the PSSM.
A character may occur in the database that no column is defined for in a PSSM. This is a bad warning because this means that the missing column is inserted and filled with a *very* low, negative score. No problem for the searching algorithms, but a big problem for probability distribution calculation—the score range is enlarged artificially and the calculation is likely to abort due to insufficient memory. Expect to see this warning when e.g., trying to search DNA PSSMs on protein data.

Character class {. . . } may occur in the input sequence, but no column for any of its representatives is defined in the PSSM.
This is just the same like above, but instead of a single character ‘x’, the inserted column stands for a set of characters. The PSSM is expected to define a column for exactly one of them, otherwise the same will happen as described above.

- Warnings concerning frequency files:

Character ‘x’ is undefined in the input alphabet.
The frequency file defines a frequency for some character which is not defined in the database alphabet and will therefore never occur in a sequence. This frequency is ignored then.

Character ‘x’ is defined as a wildcard in the input alphabet.
The frequency file defines a frequency for a character that is defined as wildcard in the database alphabet and will therefore never match. This frequency is still accounted for.

Character ‘x’ is defined as the special separator character in the input alphabet.
The frequency file defines a frequency for some character that is mapped internally to a special separator which will never occur in a sequence. This frequency is ignored then.

Sum of relative frequencies is not 1.0.
The sum of all frequencies should be exactly 1.0 such that they constitute a probability distribution. If the sum is not 1.0, this warning is issued, but the frequencies are accepted as provided. Note that this warning can be an artifact due to rounding errors.

If any of the first three warnings occurs, chances are that you specified the wrong frequency file for the database alphabet.

- Searching on complementary strand with *PoSSuMsearch*:

Characters 'x' and 'X' are both undefined in the input alphabet.
Searching on the complementary strand, reverse or not, requires exchanging PSSM columns, in particular the "A"-column with "T"- or "U"-column and "C"-column with "G"-column. *PoSSuMsearch* tries to find these columns automatically, ignoring case (so the 'x' and 'X' above may stand for 'a' and 'A'). If there is no column for neither the lower nor the upper case letter of the to-be-exchanged columns, this warning will be issued, meaning that the search will still proceed but with the corresponding columns *unexchanged*.

7.3 Bugs

Searching for PSSMs in a sequence encoded by an alphabet that contains characters not contained in the PSSMs' alphabets (e.g., searching nucleotide PSSMs in an amino acid sequence) may crash the program. See also warning "**Character 'x' may occur in the input sequence, but is not defined for the PSSM**" described above.

Other bugs are not known to the authors. Please feel free to report bugs, suggestions or comments to `rhomann |at| techfak |dot| uni-bielefeld |dot| de`.

Please do not report bugs without reading and understanding warnings produced by `possumsearch` or `possumdist`. Warnings are not printed for no reason, indeed they bear a meaning, even if they can be switched off. They are intended to help the user tracking down certain problems, and in general that's what warnings are there for.

7.4 Release history

Version 2.0 Sequence classification with PSSM family models (local and global chaining of PSSM matches).

Filtering and sorting of chains.

FASTA output format for *hmmsearch* accelerator *PSfamSearch*.

Version 1.3 Replaced options `-fwd` and `-rev` by options `-fn`, `-fc`, `-rn`, and `-rc`. Two accompanying convenience options `-2` and `-4` have been added. Matching sequences are shown complementary if found on complementary strand and if not disabled by `-ncompl`.

Output formats *changed*: in tab delimited output, the search direction (field 8) is specified by two characters now (forward vs. reverse, non-complementary vs. complementary); human readable format was extended to print out that information, too.

Option `-mult` in *PoSSuMsearch* and *PoSSuMdist* accepts values greater than 1.0 now.

New sorting keys "`r`" and "`t`".

Various bug fixes.

Version 1.2a Fixed filtering of PSSMs that cannot satisfy the p-value cutoff. Thanks to Dustin E. Schones for reporting on this bug.

Version 1.2 Added support for alphabet transformations to *PoSSuMsearch* to possibly speed up the search for protein PSSMs (option `-realpha`).

Added option `-best` to *PoSSuMsearch* to find the k best matches per PSSM.

Fixed rounding errors when using floating point PSSMs.

Performance improvements.

Determine number of CPUs in multithreaded program versions when specifying `-j` without a number.

Minor bugfixes.

Version 1.1 Fixed *PoSSuMdist* crash on Intel Solaris.

Minor changes.

Version 1.0 Initial release.

Index

References

[ABF$^+$03] T. K. Attwood, P. Bradley, D. R. Flower, A. Gaulton, N. Maudling, A. L. Mitchell, G. Moulton, A. Nordle, K. Paine, P. Taylor, A. Uddin, and C. Zygouri. PRINTS and its automatic supplement, prePRINTS. *Nucl. Acids Res.*, **31**(1):400–402, 2003.

[AKO04] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing Suffix Trees with Enhanced Suffix Arrays. *Journal of Discrete Algorithms*, **2**:53–86, 2004.

[BHGK06] Michael Beckstette, Robert Homann, Robert Giegerich, and Stefan Kurtz. Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, 7:389, 2006.

[BSH$^+$04] M. Beckstette, D. Strothmann, R. Homann, R. Giegerich, and S. Kurtz. PoSSuMsearch: Fast and Sensitive Matching of Position Specific Scoring Matrices using Enhanced Suffix Arrays. In *Proceedings of the German Conference on Bioinformatics*, volume P-53, pages 53–64. GI Lecture Notes in Informatics, 2004. ISBN 3-88579-382-2.

[DC00] B. Dorohonceanu and Nevill-Manning C.G. Accelerating Protein Classification Using Suffix Trees. In *in Proc. of the International Conference on Intelligent Systems for Molecular Biology*, pages 128–133, Menlo Park, CA, 2000. AAAI Press.

[GK95] R. Giegerich and S. Kurtz. A Comparison of Imperative and Purely Functional Suffix Tree Constructions. *Science of Computer Programming*, **25**(2-3):187–218, 1995.

[GME87] M. Gribskov, M. McLachlan, and D. Eisenberg. Profile Analysis: Detection of Distantly Related Proteins. *Proc. Nat. Acad. Sci. U.S.A.*, **84**:4355–4358, 1987.

[HGPH00] J.G. Henikoff, E.A. Greene, S. Pietrokovski, and S. Henikoff. Increased Coverage of Protein Families with the Blocks Database Servers. *Nucl. Acids Res.*, **28**:228–230, 2000.

[HH91] S. Henikoff and J.G. Henikoff. Automated assembly of protein blocks for database searching. *Nucl. Acids Res.*, **19**:6565–6572, 1991.

[HSL$^+$04] N. Hulo, C.J.A. Sigrist, V. Le Saux, P. S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, and A. Bairoch. Recent improvements to the PROSITE database. *Nucl. Acids Res.*, **32**:134–137, 2004.

[HW04] Peter M. Haverty and Zhiping Weng. CisML: an XML-based format for sequence motif detection software. *Bioinformatics*, **20**(11):1815–1817, 2004.

[KGR$^+$03] A.E. Kel, E. Gößling, I. Reuter, E. Cheremushkin, O.V. Kel-Margoulis, and E. Wingender. MATCH: a tool for searching transcription factor binding sites in DNA sequences. *Nucl. Acids Res.*, **31**:3576–3579, 2003.

[Kur99] S. Kurtz. Reducing the Space Requirement of Suffix Trees. *Software—Practice and Experience*, **29**(13):1149–1171, 1999.

[MFG$^+$03] V. Matys, E. Fricke, R. Geffers, E. Gößling, M. Haubrock, R. Hehl, K. Hornischer, D. Karas, A. E. Kel, O. V. Kel-Margoulis, D.-U. Kloos, S. Land, B. Lewicki-Potapov, H. Michael, R. Munch, I. Reuter, S. Rotert, H. Saxel, M. Scheer, S. Thiele, and E. Wingender. TRANSFAC(R): transcriptional regulation, from patterns to profiles. *Nucl. Acids Res.*, **31**(1):374–378, 2003.

[PNW99] X. M. Pan, W. D. Niu, and Z. X. Wang. What is the minimum number of residues to determine the secondary structural state? *J. Protein Chem.*, **18**(5):579–584, 1999.

[QFWW95] K. Quandt, K. Frech, E. Wingender, and T. Werner. MatInd and MatInspector: new fast and versatile tools for detection of consensus matches in nucleotide data. *Nucl. Acids Res.*, **23**:4878–4884, 1995.

[Rah03] S. Rahmann. Dynamic programming algorithms for two statistical problems in computational biology. In *Proc. of the 3rd Workshop of Algorithms in Bioinformatics (WABI)*, pages 151–164. LNCS 2812, Springer Verlag, 2003.

[RMV03] S. Rahmann, T. Müller, and M. Vingron. On the Power of Profiles for Transcription Factor Binding Site Detection. *Statistical Applications in Genetics and Molecular Biology*, **2**(1), 2003.

[SFA99] P. Scordis, D.R. Flower, and T.K. Attwood. FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics*, **15**(10):799–806, 1999.

[Sta89] R. Staden. Methods for calculating the probabilities for finding patterns in sequences. *Comp. Appl. Biosci.*, **5**:89–96, 1989.

[WNB00] T.D. Wu, C.G. Nevill-Manning, and D.L. Brutlag. Fast Probabilistic Analysis of Sequence Function using Scoring Matrices. *Bioinformatics*, **16**(3):233–244, 2000.