Structator: fast index-based search for RNA sequence-structure patterns User's Manual

Fernando Meyer
Center for Bioinformatics
University of Hamburg
Bundesstr. 43, 20146 Hamburg, Germany

April 7, 2011

Contents

1 Introduction

Structator is a software package for time efficient matching of RNA sequence-structure patterns in sequence databases. Its main features are:

- Persistent construction of an index data structure of the target database. The index, called affix array, only needs to be constructed once and is stored on disk.
- Flexible alphabet handling, including predefined DNA, RNA, and protein alphabets and the possibility to use personalized ones.
- Matching on forward and reverse complement strands. Matching in plain FASTA files is also possible.
- Support of a variety of patterns with ambiguous IUPAC symbols.
- Standard and user-defined base pairing rules.
- Integrated fast global and local chaining algorithms.
- Output of results in different formats, including BED for visualization in the UCSC Genome Browser.

Structator consists of two command line programs: *afconstruct* and *afsearch*. *afconstruct* allows the construction of tables that constitute the affix array index data structure of the target database. *afsearch* allows fast sequence-structure pattern matching in a precomputed affix array or in the plain database.

This software is available as open source under the GNU General Public License Version 3.

2 *afconstruct*

afconstruct constructs the affix array index data structure of a given database. In summary, the process of construction includes reading the database in FASTA format, mapping the sequences of the database to an alphabet, selecting the desired tables of the index to be constructed, and saving the index to files on disk. This process is performed smoothly by *afconstruct*, where the user only needs to set a few options. An overview of all possible options is given in Table 2.1 and their detailed description is given below.

2.1 Index construction options

- `<file>`
  `<file>` is the path and name of the FASTA file for which the is index is to be constructed. The file may contain one or more sequences and all are selected for index construction. Note that index-based search in the forward and reverse complement sequences only requires the construction of a single index.
- `-alph <file>`
  `-alph` takes as parameter the path and name of the text file specifying an alphabet. The sequences' characters are mapped to this alphabet and the sequences are then said to be alphabetically transformed. The index is constructed for the alphabetically transformed sequences. This option also allows alphabet reduction. Each line in the file specifies a class of characters, which means that all characters of a class are not distinguished between each other. Below is an example of an alphabet file.

  ```
  Aa A
  Cc
  Gg
  TtUu U
  *BbDdHhNnYyRrSsVvWwKkMmXx
  ```

  Lines beginning with *, like the last one, imply a class of wildcards (i.e. ambiguous characters). Wildcards in the database indicate unknown or unsequenced regions, hence such regions cannot be matched against any pattern. Furthermore, characters must be given without spaces in each line. A space and a character imply that the first character after the space is a so-called *class representative*. The class representative

<file>	Load FASTA file
`-alph <file>`	Use alphabet defined in file
`-dna`	Use 4-letter DNA alphabet (default)
`-rna`	Use 4-letter RNA alphabet
`-protein`	Use 20-letter protein alphabet
`-a`	Construct all tables
`-suf`	Construct suf table
`-lcp`	Construct lcp table
`-skp`	Construct skp table
`-aflk`	Construct aflk table
`-sufr`	Construct sufr table
`-lcpr`	Construct lcpr table
`-skpr`	Construct skpr table
`-aflkr`	Construct aflkr table
`-s <index>`	Save constructed structures to given index name
`-x`	Do not save alphabetically transformed sequence
`-c`	Output constructed structures to screen
`-t <file>`	Output constructed structures to text file
`-time`	Display elapsed times

Table 2.1: Overview of options of program *afconstruct*.

is shown in place of the original character when outputting transformed sequences to file or screen. If no representative is explicitly specified, the first character of the line is chosen as the representative. In summary, in the example above we have 5 character classes, whose representatives are A, C, G, U, and *.

As a remark, although ambiguous IUPAC character such as N, R, Y, etc. indicate unknown regions in the database, they can be used for defining patterns. It is noted here that the user does not have to create character classes for such characters since they are already recognized by Structator. More about this is discussed in the section about program *afsearch*.

- `-dna`, `-rna`, `-protein`
  These options allow transforming the input sequences to predefined alphabets. The alphabet for DNA, RNA, and protein sequences has size 4, 4, and 20, respectively. More precisely, the characters of each alphabet option are the following:
  `-dna`: A, C, G, T
  `-rna`: A, C, G, U
  `-protein`: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y
  Uppercase and lowercase characters are not distinguished. If the sequences contain characters other than the ones above, one can create a new alphabet in a text file and use it with the option `-alph`.

- `-a`
  `-a` selects all eight tables of the affix array for construction. The tables are listed next.

- `-suf`, `-lcp`, `-skp`, `-aflk`,
  `-sufr`, `-lcpr`, `-skpr`, `-aflkr`
  These options allow the user to individually select the desired tables of the index to be constructed. Each option corresponds, as expected, to the table of the same name, that is:

  `-suf`: suffix array
  `-lcp`: longest common prefix
  `-skp`: skip
  `-aflk`: affix link
  `-sufr`: reverse prefix (i.e. suffix array of the reverse sequences)
  `-lcpr`: longest common prefix of the reverse sequences
  `-skpr`: skip of the reverse sequences
  `-aflkr`: affix link of the reverse sequences

  Note that, because certain tables depend on another for their construction, a table may be constructed even if it is not selected. For example, table suf will automatically be constructed if the user only selects table lcp. For constructing table aflk (aflkr) there are two possibilities. By selecting aflk (aflkr) only, tables suf, lcp (lcpr), and sufr are automatically selected as well, and binary search method is used in the construction of aflk (aflkr). If the user additionally selects skp (skpr), the construction of aflk (aflkr) is sped up by the additional use of this table. The skip tables skp and skpr can be deleted by the user after the construction of the affix link tables aflk and aflkr because they are not required for pattern matching.

- `-s <index>`
  By using the option `-s` along with an index name, each table that is constructed is stored on disk in its own file. The name of each file is `[index name].[table name]`. Additional files are also stored. One file with extension `.alph` stores the alphabet, one with extension `.base` stores basic information about the sequences such as their length, and one with extension `.des` stores the description of each sequence. The sequences and alphabetically transformed sequences are stored in a file with extension `.seq` and `.tseq`, respectively. Note that all the generated files are binary.

- `-x`
  This option prevents *afconstruct* from saving alphabetically transformed sequences to file. This is useful for saving disk space, but note that the sequences of the index will be transformed each time program *afsearch* (see next section) is executed.

- `-c`
  `-c` outputs the constructed tables and the corresponding suffixes (or reverse prefixes)

to screen. For ease of readability, the strings of the reverse prefixes are printed in reverse order. This option is only recommended for small databases, say, with sequence length up to 100.

- `-t <file>`
  `-t` works like the option `-c`, but it directs the output to the specified file.

- `-time`
  With this option the time required to construct each selected table is displayed.

Be aware that the generated files may overwrite existing ones without warning!

2.2 Using *afconstruct*

We show an example for constructing all tables of the affix array, including also tables skp and skpr, for the Rfam database release 9.1. The database is stored in the file `Rfam.fas`. Because the sequences contain characters different from the 4-character RNA alphabet, we use with option `-alph` the same alphabet file that is exemplarily described above with 5 character classes. This file is here called `myrna.alph`. Below is the program call and its screen output.

```
$ ./afconstruct /path/to/fasta_file/Rfam.fas -alph
/path/to/alphabet_file/myrna.alph -a -s /path/to/save/index/Rfam

Fasta file:          Rfam.fas
Number of sequences: 1149685
Total length:        179030400

Computing suf... done

Computing lcp... done

Computing skp... done

Computing sufr... done

Computing lcpr... done

Computing skpr... done

Computing aflk with skpr... done

Computing aflkr with skp... done
```

The program execution produces these files:

```
$ ls -goh
total 5.0G
-rw-r-r- 1 688M 2010-01-04 16:28 Rfam.aflk
-rw-r-r- 1 688M 2010-01-04 16:39 Rfam.aflkr
-rw-r-r- 1 68 2010-01-04 16:13 Rfam.alph
-rw-r-r- 1 4.4M 2010-01-04 16:13 Rfam.base
-rw-r-r- 1 29M 2010-01-04 16:13 Rfam.des
-rw-r-r- 1 172M 2010-01-04 16:15 Rfam.lcp
-rw-r-r- 1 116M 2010-01-04 16:15 Rfam.lcpe
-rw-r-r- 1 116M 2010-01-04 16:17 Rfam.lcper
-rw-r-r- 1 172M 2010-01-04 16:17 Rfam.lcpr
-rw-r-r- 1 172M 2010-01-04 16:13 Rfam.seq
-rw-r-r- 1 688M 2010-01-04 16:15 Rfam.skp
-rw-r-r- 1 688M 2010-01-04 16:17 Rfam.skpr
-rw-r-r- 1 688M 2010-01-04 16:15 Rfam.suf
-rw-r-r- 1 688M 2010-01-04 16:17 Rfam.sufr
-rw-r-r- 1 172M 2010-01-04 16:13 Rfam.tseq
```

3 *afsearch*

afsearch is the program for matching RNA sequence-structure patterns in a precomputed index or directly in a plain FASTA file. In case an index is used, matching patterns containing no base pairs and no ambiguous IUPAC characters requires only the precomputation of the suf table, otherwise tables suf and sufr, lcp and lcpr, and aflk and aflkr are required. An overview of the options of *afsearch* is given in Table 3.1 and are explained in more detail below.

3.1 Pattern search options

- `<data>`
  `<data>` is the path and target FASTA file or the path and prefix name of the files (i.e. file name without extension) storing an index. Remember to map the desired tables in case the target is an index.
- `-alph`
  `-alph` takes as parameter the path and name of the text file specifying an alphabet. See the full description of alphabet files above on the section about *afconstruct*. Note that this option is only effective if the target data is a FASTA file. Otherwise, if it is an index, the alphabet used is obtained from the index.

  If the FASTA file or the pattern file (see below option `-pat`) contains ambiguous IUPAC characters, e.g. N, R, S, Y, etc., these must be specified in the alphabet file. However, while in the FASTA file they indicate unknown or unsequenced regions and hence cannot be searched, these characters can be used to define patterns. It is noted that the use of ambiguous characters in patterns does not require the user to create the corresponding character classes, e.g. N denoting A, C, G, or U or R denoting A or G, since all standard IUPAC classes are already recognized by Structator.
- `-dna, -rna, -protein`
  Alphabet option for the respective kind of sequence data. For more details on the predefined DNA, RNA, and protein alphabets see the section about *afconstruct*. As with the `-alph` option, these are only effective if the target data is a FASTA file.
- `-pat <file>`
  `-pat` takes as parameter a text file containing one or multiple sequence-structure pat-

`<data>`	Index name or FASTA file
`-alph <file>`	Use alphabet defined by file (option applies only to FASTA file)
`-dna`	Use 4-letter DNA alphabet (default) (option applies only to FASTA file)
`-rna`	Use 4-letter RNA alphabet (option applies only to FASTA file)
`-protein`	Use 20-letter protein alphabet (option applies only to FASTA file)
`-pat <file>`	Search for (structural) patterns
`-for`	Search in the forward sequence (default)
`-rev`	Search in the reverse complement sequence. For searching in the forward sequence as well, combine it with -for
`-comp <file>`	Load base-pair complementarity rules from file
`-a`	Map all index tables
`-suf`	Map suf table
`-lcp`	Map lcp table
`-aflk`	Map aflk table
`-sufr`	Map sufr table
`-lcpr`	Map lcpr table
`-aflkr`	Map aflkr table
`-bed`	Output matches in BED format
`-allm`	Report all matches of variable length patterns, i.e. not only the longest ones
`-match <k>`	Report only sequences matching at least k different patterns
`-t <file>`	Write matches to text file instead of to screen
`-seqdesc`	Include sequence description in the results, otherwise tag each pattern match with the sequence id
`-time`	Display elapsed times
`-silent1`	Do not output matches
`-silent2`	Do not output anything
Chaining options:	
`-global`	Perform global chaining
`-local`	Perform local chaining
`-wf <wf>`	Apply weight factor > 0.0 to fragments
`-maxgap <width>`	Allow chain gaps with up to the specified width
`-minscore <score>`	Report only chains with at least the specified score
`-minlen <length>`	Report only chains with number of fragments $>=$ length
`-top <#>`	Report only top # scoring chains of each sequence
`-chainrep <file>`	Write chaining report to text file instead of to screen
`-show`	Show chains in the report

Table 3.1: Overview of options of program *afsearch*.

terns. Each pattern is specified in three consecutive lines. The first line begins with the symbol > followed by the description of the pattern. Optionally, the description may be followed by pipe symbols | separating these supplemental options:
`weight`: a weight that is assigned to a chain fragment corresponding to a match of the respective pattern. If no weight is provided, value 1 is assumed by default.
`startpos`: this option denotes the expected starting match position of the pattern in the searched sequences and is used for computing the score of local chains. Furthermore, it must be specified for none or all patterns. If not used, the starting position of the patterns are automatically computed in a stacked way, i.e., the starting position of a pattern is the sum of the length of all patterns defined before it +1.
`instance`: the instance is the number that defines the allowed order of occurrence of a chain fragment in a chain of matches. Patterns of equal instance are equivalent w.r.t. the chaining position. This option must be specified for none or all patterns. If not specified, the order of occurrence of chain fragments respects the top-bottom order in which the respective matching pattern is defined in the patterns file. For instance, a chain fragment of a pattern defined in the beginning of the file must occur at a position prior to a chain fragment of a pattern defined in the end of the file.
`maxstemlength`: maximum length (i.e. number of base pairs) of the stem region of the pattern. The minimum length is derived from the dot-bracket sequence structure. For example, if the pattern has structure `((((...))))`, the minimum stem length is 4 and `maxstemlength` must be at least 4. The pattern characters for base pairs occurring in number above the minimum stem length are assumed to be ambiguous characters N.
`maxrightloopextent` (alternatively `mrlex`): number of positions by which to extend the beginning (from left to right) of the loop region. The extended pattern positions are assumed to be characters N. See the example below for the usage of this option.
`maxleftloopextent` (alternatively `mllex`): number of positions by which to extend the end (from left to right) of the loop region. The extended pattern positions are assumed to be characters N. See the example below for the usage of this option.
`maxmispair`: maximum number of base pairs that may not obey the chosen complementarity rules, say, the Watson-Crick (A, U), (U, A), (C, G), (G, C).

Supplemental options must be provided between two pipe symbols and its keyword, say, *weight*, is followed by the equal sign (=) and a value.

The second line of the pattern definition contains the sequence information, i.e., a sequence of bases possibly containing ambiguous IUPAC characters. It is noted that Structator automatically recognizes ambiguous characters and tries to match the corresponding base, e.g. A or G in place of an R. The third line contains the structure information in dot-bracket notation. In this notation, unpaired bases are represented by dots `.` and paired bases are represented by `(` and `)`. Note that positions specified by dots are not strictly unpaired, i.e., they may form a base pair with another position

although this is not required. Supported structures are hairpins with bulges and/or internal loops and also single strands. Observe that for specifying a single stranded pattern it is necessary to provide a sequence of dots.

As an example, a patterns file may contain the following text.

```
>p0|maxleftloopextent=1|maxrightloopextent=1|maxstemlength=6
RNSNGKUNGCNHNSCY
(.((((....)))).)
```

The pattern above represents a set of patterns, namely:

```
>p0
RNSNGKUNGCNHNSCY
(.((((....)))).)
>p1
RNSNGKNUNGCNHNSCY
(.((((.....)))).)
>p2
RNSNGKUNGCNNHNSCY
(.((((.....)))).)
>p3
RNSNGKNUNGCNNHNSCY
(.((((......)))).)
>p4
NRNSNGKUNGCNHNSCYN
((.((((....)))).))
>p5
NRNSNGKUNGCNNHNSCYN
((.((((.....)))).))
>p6
NRNSNGKNUNGCNHNSCYN
((.((((.....)))).))
>p7
NRNSNGKNUNGCNNHNSCYN
((.((((......)))).))
```

- `-for`
  Option for searching in the forward sequences. This option is selected by default.
- `-rev`
  Option for searching in the reverse complement sequences. If used in combination with

the option `-for`, search is performed in both the forward and reverse complement sequences, otherwise search is only performed in the reverse complement sequences. Observe that searching in reverse complement sequences of a database does not require computing an index for the reverse complement sequences. *afsearch* handles this by automatically computing the reverse complement of the patterns and by using these patterns for search.

- `-comp <file>`
  The parameter of the option `-comp` is a file specifying complementary bases. A line with two bases, given without any spaces or punctuation, implies that matches to the patterns can contain such a base pair. It is not necessary to specify the pairing rule twice. For example, for pairs (C, G) and (G, C) it suffices to have a line `CG`. Below is a sample file.

  ```
  AU
  CG
  GA
  GU
  ```

  According to this file, these base pairs are possible: (A, U), (U, A), (C, G), (G, C), (A, G), (G, A), (U, G), (G, U). Note that if the option `-comp` is not used, Watson-Crick base pairs are allowed by default.

- `-a`
  `-a` maps all six tables of the index (see the next options) to memory. Mapping means that they are made available to *afsearch*, but are not immediately loaded into memory. Blocks of data are only effectively loaded into memory as parts of the tables are read during pattern matching operations.

- `-suf`, `-lcp`, `-aflk`
  `-sufr`, `-lcpr`, `-aflkr`
  These options allow the individual selection of the tables that are mapped to memory. Matching single-stranded patterns containing no ambiguous characters requires only table suf. Otherwise, it is additionally mandatory the selection of tables sufr, lcp, lcpr aflk, and aflkr.

- `-bed`
  Option for printing out the matches in BED format. Otherwise, if not used, the matches are printed out in a format similar to BED, but including the matched substring and its secondary structure.

- `-allm`
  This option is only effective when matching patterns of variable length. By using it, all matches of all possible different pattern lengths are reported. Otherwise, if not used and there are matches embedded in other matches of the same pattern, embedded

matches are ignored. For example, consider a pattern with minimum length 6 and maximum length 10 and an arbitrary sequence. If the pattern matches with length 6 at sequence position 5 and with length 10 it matches at position 2, then the match at position 5 is ignored because it is embedded in the match at position 2.

- `-match <k>`
  `-match` with parameter `k` neglects sequences and pattern matches occurring in them if the matches are of not of at least `k` different patterns.
- `-t <file>`
  `-t` writes the matches to the specified file instead of to screen. The matches are sorted by sequence and, within a sequence, by ascending matching position.
- `-seqdesc`
  Option `-seqdesc` includes the sequences' description in the list of pattern matches. If this option is not used, the sequence is identified by a number that corresponds to its order of definition in the database, beginning from 0.
- `-time`
  Option to display the time needed to search for each pattern.
- `-silent1`
  `-silent1` avoids the output of matches and chains. Note that also the output to text file by the use of option `-t` is neglected.
- `-silent2`
  Option for not outputting anything.

3.2 Chaining options

- `-global`
  Option to perform global chaining of matches. It is the default option.
- `-local`
  Option to perform local chaining of matches.
- `-wf <wf>`
  `-wf` takes as parameter a positive weight factor that is applied to all chain fragments. For instance, if a chain fragment of a pattern has weight 2, a weight factor of 10 implies that the chain fragment will have weight 20.
- `-maxgap <width>`
  `-maxgap` takes as parameter the maximum distance (i.e. number of bases) allowed between chain fragments.
- `-minscore <score>`
  Report only chains with at least the specified score.

- `-minlen <len>`
  Report only chains with at least the specified number of chain fragments.
- `-top <#>`
  Report only top # scoring chains. If this option is not used, all chains are reported.
- `-chainrep <file>`
  `-chainrep` writes to the specified file the chaining report, otherwise the chains are written to screen. Chains are reported in descending order of their chain score.
- `-show`
  Show chain fragments and their coordinates (i.e. start and end matching position and weight) in the chaining report.

3.3 Using *afsearch*

We use *afsearch* in this example to search with three patterns derived from the consensus structure of the Rfam family *OxyS RNAs* (Acc.: RF00035). The patterns, shown below, are assigned a weight of 1 for computing global chains of matches. The patterns are stored in a file called `oxyS.pat`. We search in the index of Rfam release 10, here called `Rfam10`, which was preconstructed with *afconstruct*. The allowed base pairs are (A, U), (U, A), (C, G), (G, C), (G, U), and (U, G), which are specified in a text file and used with the option `-comp`. We also set *afsearch* to report global chains of matches with at least score 2 by using the option `-minscore`. The pattern matches and the chains are written to files `matches.txt` and `chains.txt`, respectively. The patterns file is as follows.

```
>HP1|maxrightloopextent=1|maxleftloopextent=1|maxmispair=6|weight=1
NNNNNNNNNNNNNNNNNNNNNACCCNUNANNNNNNNNNNNNNNNN
(((((((.((((.((.((..........)).)).)..)))))))))))
>HP2|maxrightloopextent=5|weight=1
GNNNNNCUCACNN
((((.....))))
>HP3|maxmispair=2|maxrightloopextent=2|weight=1
NNGGANCUNNNNNNNNNNNN
(((((((.....)))))))
```

The command to call *afsearch* and the screen output are:

```
$ ./afsearch /path/to/index/Rfam10 -pat /path/to/patterns_file/oxyS.pat
-comp /path/to/comp_file/wcgu.comp -a -t matches.txt -minscore 2 -show
-chainrep chains.txt
```

```
Number of sequences: 1149685
Total length:        179030400

!Searching for pattern HP1 in the forward sequence(s)... done
!#Matches: 8619

!Searching for pattern HP2 in the forward sequence(s)... done
!#Matches: 1699

!Searching for pattern HP3 in the forward sequence(s)... done
!#Matches: 142219

!#Total matches: 152537
```

The first 10 lines of the matches file are:

```
$ head -n 15 matches.txt
![matched substring/structure] [seq. id] [matching pos.] [pattern id] [weight]
[strand]
ACGGAUCUCUUGGUUCUGG 119 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCUGG 122 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCUGG 124 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCUGG 125 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCUGG 126 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCCGG 132 11 2 1 f
(((((((.....)))))))
ACGGAUCUCUUGGUUCUGG 136 11 2 1 f
(((((((.....)))))))
```

Observe that the matches are sorted by ascending sequence id. The id corresponds to the order of occurrence of the sequence in the database. Below are the first 26 lines of the chaining report showing 5 chains. There are in total 316 chains with at least score 2.

```
$ head -n 26 chains.txt
head -n 26 chains.txt
![sequence] [chain score] [chain length] [strand]
>CP000468.1+4477379-4477488 3 3 f
```

```
0 47 0 46 1
48 65 49 62 1
66 86 90 108 1
GAAACGGAGCGGCACCUCUUUUAACCCUUGAAGUCACUGCCCGUUUC GAGUUUCUCAACUC GCGGAUCUCCAGGAUCCGC
>CP000034.1+3532296-3532405 3 3 f
0 47 0 46 1
48 65 49 62 1
66 86 90 108 1
GAAACGGAGCGGCACCUCUUUUAACCCUUGAAGUCACUGCCCGUUUC GAGUUUCUCAACUC GCGGAUCUCCAGGAUCCGC
>AAJW02000005.1+188036-188145 3 3 f
0 47 0 46 1
48 65 49 62 1
66 86 90 108 1
GAAACGGAGCGGCACCUCUUUUAACCCUUGAAGUCACUGCCCGUUUC GAGUUUCUCAACUC GCGGAUCUCCAGGAUCCGC
>ABHW01000012.1+10515-10624 3 3 f
0 47 0 46 1
48 65 49 62 1
66 86 90 108 1
GAAACGGAGCGGCACCUCUUUUAACCCUUGAAGUCACUGCCCGUUUC GAGUUUCUCAACUC GCGGAUCUCCAGGAUCCGC
>AE014073.1+3594803-3594912 3 3 f
0 47 0 46 1
48 65 49 62 1
66 86 90 108 1
GAAACGGAGCGGCACCUCUUUUAACCCUUGAAGUCACUGCCCGUUUC GAGUUUCUCAACUC GCGGAUCUCCAGGAUCCGC
```

The chains are sorted by descending chain score. In this example, 3 is the maximum score possible. Each chain contains the description of the sequence where it occurs, the fragments' coordinates (i.e. expected or "stacked" start and end matching positions of the fragment, actual start and end matching positions of the fragment, and fragment weight), and the matching substring of the fragments.