Efficient Multiple Genome Alignment Manual

Michael Höhl *

March 18, 2003

About This Document

This manual is not a technical description. As such it does not include formal definitions but instead refers to other documents containing them. It tries to avoid the excessive use of technical termini except in places where the author considers them necessary.

1 Introduction

`mga` is a software tool for efficiently aligning two or more sufficiently similar genomic sized sequences [HKO02]. It belongs to the category of anchor-based multiple alignment methods. `mga` uses *multiMEMs* (or MEMs, MUMs as special cases) to anchor the alignment. In essence, they are (hopefully long) stretches or regions of identical bases occuring in all input sequences.

The first step in producing an alignment is to find *multiMEMs* (also referred to simply as matches). In order to ensure a consistent alignment, these matches have to be chained. Without going into detail, this is done by an algorithm that gives the optimal chain of matches out of all matches found, according to some criterion. The chained (and therefore selected) matches are considered aligned.

In regions between these matches (and possibly at the start and end of the sequences) are gaps. If the gaps are "long", `mga` can recursively repeat the first step, searching for matches of shorter but significant length. This directly results in a higher number of aligned matches. As a side effect, more and shorter gaps are produced, allowing the following process to be employed more often.

*Technische Fakultät, Universität Bielefeld, Postfach 100 131, 33501 Bielefeld, Germany, Email: mhoehl@TechFak.Uni-Bielefeld.DE

In the second step, if the remaining gaps are “short”, a conventional alignment method is used to align these non-identical regions. The meaning of “long” and “short” is sequence dependent, of course. Still, at the end, some unaligned long gaps may remain. This makes sense biologically: since there are no matches exceeding the length threshold, there is no detectable similarity and the gaps are not forced into an alignment. This way, `mga` can cope with long insertions, deletions, etc., retaining its overall efficiency.

2 A Word of Caution

`mga` assumes the input sequences to be sufficiently similar. Then it can find “enough” and “long enough” matches, on which the algorithm depends. If this is not the case, `mga` will fail to produce meaningful alignments. Keep this in mind when choosing your sequences. For example, it makes sense to input only syntenic regions of long, divergent genomes.

3 Preparing the Input Sequences

`mga` efficiently locates anchors, i.e., matches in the input sequences. To do so, these sequences need to be preprocessed. This preprocessing step has to be done only once, as the created index is stored on secondary memory (usually a hard disk) and resides there until deleted by the user. The tool to index the sequences is `mkvtree`. If you have three or more sequences you must proceed as described in a). If you have two sequences you can choose from two alternatives. Either you proceed as in a) or as in b). The implications are: a) the index is selfcontained, faster to use once it is built, and you are able to use the recursive strategy. b) the index uses minimal space (it contains only one sequence), so you have to supply the second sequence with options `-mem` or `-mum`. Because of that you can supply a different sequence every time you call `mga` and you can compute MUMs, i.e., unique matches.

Here are the two ways to index the sequence(s) (assuming appropriately named sequences are to be aligned):

a)
```
mkvtree -dna -lcp -suf -tis -indexname 2ormoreseqs\
   -db seq1.fna seq2.fna [seq3.fna ...]
```

You can use the index by calling

```
mga [options] 2ormoreseqs
```

b)
```
mkvtree -dna -suf -tis -sti1 -pl 8 -db seq1.fna
```

You can use the index by calling

```
mga [options] -mem|mum seq2.fna seq1.fna
```

This is all you need to know to get started. For more information on `mkvtree` consult [Kur02].

4 The Program

`mga` assumes that the sequences to be aligned have been indexed by the program `mkvtree`. The commandline has the following syntax:

The program is called as follows:

`mga` [*options*] *indexname*

And here is a description of the options:

`-l` $\ell_1 \ldots \ell_n$
: Specify the length threshold(s) ℓ_i. These must be positive integers. At least one length threshold has to be specified. If there are several ℓ_i, they have to be enumerated in strictly decreasing order. Only matches at least of length ℓ_i are considered at each recursion step i. This option is mandatory.

`-min` *n*
: Specify the minimal number of matches that have to be found in order to be chained. This must be a positive integer. It prevents `mga` from forming a backbone of anchors based on too few evidence, i.e., matches.

`-last`
: By default, when the number of matches found in a recursion step is less than specified by `-min`, they are not considered. In order to increase sequence coverage during the last recursion step they can be taken into account using this switch. This option requires option `-min` to be used.

`-max` *n*
: Specify the maximal number of matches that are to be chained. This must be a positive integer. If this number is exceeded the matches are not taken into account. As explained in Section 10.1 for $k \geq 3$ sequences a *kd*-tree based algorithm is chosen which is slow from a certain number of matches onwards. Using this option is recommended. A reasonable value for *n* is $100,000$ or $200,000$.

`-mum` *sequence2*
: Specify the filename of the second sequence to be used in order to compute MUMs, that is, maximal unique matches. The first sequence has to be indexed and its indexname has to be delivered as usually. See Section 3 for a more detailed description.

`-mem` *sequence2*

Specify the filename of the second sequence to be used in order to compute MEMs, that is, maximal exact matches. The first sequence has to be indexed and its indexname has to be delivered as usually. See Section 3 for a more detailed description.

`-gl` ℓ

Specify the maximal length for a gap to be still considered alignable. This must be a non-negative integer. Since there are *k* sequences, each gap consists of *k* parts. If one or more of them are longer than ℓ, the gap is considered unalignable. Larger values for ℓ increase the sequence coverage. The default value for ℓ is $1,000$.

`-clustalw`

Align short gaps using Clustal W [THG94] and show the resulting aligned sequence data. The multiple sequence alignment program `clustalw` must be in the search path of `mga` for this option to work. The running time for this phase is dependent on the parameter ℓ specified by option `-gl`. For values of $\ell > 1,000$ this gets excessively slow.

`-msascript` *scriptname*

This option is similar to option `-clustalw` but allows you to use multiple alignment tools and parameters of your choice by invoking script `scriptname`. See Section 5 for a detailed description.

`-greedy`

Align short gaps using a greedy approach and show the resulting aligned sequence data. This option works only for two sequences.

`-identity` *n*

Specify the minimal percent identity that an aligned gap must reach or exceed in order for its alignment to be displayed. This must be an integer $\in [0,100]$. The default value for *n* is 60. This option only works for two sequences and in combination with option `-maxdist`.

`-maxdist` *d*

Specify the maximal edit distance for a gap to be still considered alignable. Larger values for *d* increase the sequence coverage. This option only works for two sequences and in combination with option `-greedy`. A reasonable value for *d* is $1,000$.

`-xdrop` *x*

Align short gaps using the X-drop approach and show the resulting aligned sequence data. Specify the X-drop parameter to determine how much difference is tolerated in the alignment. Larger values for *n* increase the sequence coverage but may lead to degenerated alignments. A reasonable value for *x* is 1 or 2.

`-nofast`

Indicate that matches are to be chained using the general $O(k \cdot m^2)$ time algorithm where

k is the number of sequences and m the number of matches. Normally, one of the two following, asymtotically faster algorithms is used depending on k: For $k = 2$ an $O(m \cdot \log m)$ time algorithm is chosen; for $k > 2$ a kd-tree based algorithm is chosen if $m \geq 1000$. This option serves mainly comparison purposes. Using this option is *not* recommended.

`-always`
: Indicate to always recurse into a gap, even if the gap has been considered previously without success: Either no matches of the specified length threshold were found or their number was lower than required by option `min`. Using this option is recommended.

`-volume`
: Indicate to use the volume weight function. Normally, the weight of a match equals its length ℓ. Using this option the weight equals ℓ^k where k is the number of sequences.

`-chain`
: Produce a file containing chained matches only. See Section 6 for a detailed description.

`-gap`
: Produce files containing the sequence data of unaligned gaps, one file per gap and sequence. See Section 6 for a detailed description. *Caution:* This option may produce thousands of files if the partial alignment is highly fragmented. Check the number of gaps which appears in the summary before using this option.

`-alignedseqs`
: Produce files containing the sequence data of aligned regions and matches, one file per continuous stretch. See Section 6 for a detailed description. *Caution:* This option may produce thousands of files if the partial alignment is highly fragmented. Check the number of gaps which appears in the summary before using this option.

`-match`
: Show the sequence data of the aligned *multiMEMs*. The default is to only report a summary of where the matches are located in a single line. This option and option `-bl` exclude each other.

`-bl` ℓ
: Show the first and last ℓ bases of the sequence data of the aligned matches. Specify the boundary length as a positive integer. This option is useful for highly similar sequences, when only the differences in the gaps and the bases at their boundaries are of interest. The default is to only report the matches in a single line. This option and option `-match` exclude each other.

`-absolute`
: Show absolute positions, i.e., positions in the concatenated sequences. The default is to show relative positions, i.e., positions relative to each sequence.

`-width` *n*
: Specify the width of a single line of an alignment. If $n = 0$ then put every chunk of the alignment data in a single line. If this option is not supplied then the default line width of 60 is used.

`-xml`
: Produce a file containing *all* `mga` output in XML format. Using this option is recommended for postprocessing the output of `mga`.

`-o` *prefix*
: Specify the output file prefix. Normally, all output is sent to `stdout`. Using this option writes the output to various files. The number of created files and their names depend on the options, see Section 6 for a detailed description. Existing files will be overwritten without warning.

`-v`
: Be verbose, that is, report the number of matches found in each recursion step as well as the resource requirements of the computation. Using this option is recommended.

`-help`
: Show a summary of all options and terminate.

Each of the options may be specified only once.

5 Scripts

As explained in the introduction short gaps between matches are aligned by conventional alignment methods. Three of them are built-in and ready to use but there is no need to limit oneself to these methods. Any external tool can be used for that purpose as long as the resulting alignment is presented in a format `mga` can parse.

The idea is to have a short script that takes care of several steps:

1. If the alignment tool can not read in FASTA files the script must call a conversion tool like `readseq` [Gil01] to provide the sequences in a suitable format.
2. The alignment tool is called with appropriate parameters and is passed on the name of the file containing the sequence data.
3. The resulting alignment must be stored either in GDE nucleotide format or in FASTA format. If the alignment tool is not able to output one of these formats the script must convert the sequences.
4. The order of the sequences must be unchanged from the input file: `mga` depends on it.

5. The filename of the aligned sequences must have the suffix `.fna` replaced by `.gde`—even if they are in FASTA format.

6. The script carries the responsibility of deleting any intermediate files. Otherwise these files will accumulate.

After completion of the script `mga` will delete the input (`.fna`) and output (`.gde`) file.

Option `-msascript` expects the name of the script to be run. In order for this to work two things must be ensured: First, the "x" permission must be set. Second, the script must be in the search path of `mga`; alternatively you can specify the complete path.

`mga` is distributed along with an example script called `myclustalw.sh`. Its content is listed below:

```
#! /bin/sh
if [ $# -ne 1 ] ; then
  echo "Usage: $0 <inputfile>"
  exit 1
fi
inputfile=$1
options="-type=DNA -dnamatrix=IUB -output=GDE -outorder=input"
clustalw ${options} -infile=${inputfile}
if [ $? -ne 0 ] ;  then
  echo "failure: clustalw ${options} -infile=${inputfile}"
  exit 1
fi
dnd=‘echo ${inputfile} | sed -e ’s/fna\$/dnd/’‘
rm -f ${dnd}
```

Inside this `sh`-script the filename can be accessed by refering to variable $1. Clustal W is called with several options, the last one specifying the location of the sequence data. The first option forces Clustal W to expect nucleotide data to prevent its recognition heuristics from failing in some cases. The second option chooses the IUB matrix for scoring the alignments. This is actually the default in recent versions. The options starting with `-out` are essential: they ensure that a) the resulting alignments are stored in a format `mga` can parse and b) the sequences are in the assumed order.

The last two lines of the script delete an intermediate file produced by Clustal W, the guide tree. Otherwise the files would accumulate.

This script does exactly what using `-clustalw` was doing in `mga` before version 2003-03-18. Now, this option has been slightly modified by omitting `-dnamatrix=IUB` in the call of Clustal W. This improves compatibility with older versions of that tool which do not have the IUB matrix built in.

Sometimes gaps appearing at the start and end of an alignment should be scored in a different way from gaps in the middle of an alignment. This can be achieved by parsing the FASTA headers of the sequences. `mga` writes out sequence number, length, start and end position of the sequence data. The prototypical FASTA header looks as follows (variables are enclosed by '<' and '>'):

```
> Seq=<seqnum> length=<length>, start=<start>, end=<end>
```

6 Output

The results of `mga` are written to a number of files whose prefix can be specified by option `-o`. The following files may be generated:

prefix.**`summary`** This file contains the parameters used during the computation, a description of the sequences and a short statistics on the number and length of the various regions.

prefix.**`align`** This file contains the data of the alignment. By default, only the positions of all matches, aligned and unaligned gaps are given. This results in a short output which is useful for getting a brief overview of the structure of the alignment. The actual sequence data of the matches and aligned gaps are presented if requested using options `-match` / `-bl` and `-clustalw` / `-msascript` / `-greedy` / `-xdrop`.

prefix.**`reg`***X* These files contain the sequence data of all aligned regions and matches in FASTA format. There will be a file for every continous region number X. The files will only be created if option `-alignedseqs` is specified.

prefix.**`gap`***X*.**`seq`***Y* These files contain the sequence data of all unaligned gaps in FASTA format. There will be a file for every gap number X and every sequence number Y. The files will only be created if option `-gap` is specified.

prefix.**`chain`** This file contains all chained matches and is generated by option `-chain`. The file can, for example, be parsed by a graphical user interface that visualizes the backbone of anchors and computes alignments of short gaps on demand.

prefix.**`xml`** This file contains all information in a structured way using XML-format. It can be used to postprocess the output of `mga`, e.g., to generate HTML pages containing the alignment. This is demonstrated by `mga2html.pl` which is included in the distribution.

If option `-o` is not specified the output will be written to `stdout`. This does not work for all options, e.g., option `-gap` requires option `-o` to be present.

7 Format of Textual Output

In the following, the textual output of `mga` is presented prototypically for all options that influence it. Variables are enclosed by '<' and '>'. The input consists of $k \geq 2$ sequences. All relative positions start at base zero. This is the default if option `-absolute` is not given.

Matches are output in the following format:

```
<length> <position 1> ... <position k>
```

Option `-match` displays the complete sequence data once, as it is the same in all sequences.

```
Exact: <sequence data>                                    <length>
```

Option `-bl` displays the first and last bases of the sequence data once.

```
Exact: <sequence data>                                    <length>
       ...skipping bases...
Exact: <sequence data>                                    <length>
```

Every gap consists of k parts: one for each of the k sequences. Depending on its length a particular part of an *aligned* gap is displayed in one of three different formats. There is one format for the general case and two additional formats for special cases. The general case occurs when a part is at least two bases long. Then, the length and the start position of that part is shown. For better readability, the end position is given as well.

```
... <length i>:<start position i>-<end position i> ...
```

The first special case occurs when a particular part consists of exactly one base. This part is considered a putative SNP. It is output using a special format to simplify extracting it.

```
... !<SNP position i> ...
```

The second special case occurs when a particular part has no length at all. This is possible and means that the other (non-empty) parts are insertions: there is no corresponding sequence data in this sequence. This is indicated by a dash.

```
... - ...
```

An unaligned gap is always indicated as follows. Note that a short explanation in the actual output was cut off in order to fit the line on the page. It is displayed in the corresponding example in the next section.

```
Gap <gap number>: <min. length> + <difference> = <max. length>
```

The sequence data of aligned gaps from two sequences as computed by options `-greedy` and `-xdrop` can contain dashes '`-`' to indicate gap characters. Mismatches are indicated by exclamation marks '`!`'. The lengths 1 and 2 are identical as they refer to the (same) length of the line.

```
Sbjct: <sequence data 1 with gaps>                    <length 1>
       <mismatches>
Query: <sequence data 2 with gaps>                    <length 2>
```

The sequence data of aligned gaps as computed by the option `-clustalw` can contain dashes '`-`' to indicate gap characters. The first sequence is always shown completely. The other sequences are displayed in a different way: To enhance readability, matching bases w.r.t. the first sequence are marked as dots '`.`'. The remaining sequence data is shown as usual.

```
Seq <1>: <sequence data 1 with gaps>                    <length>
...
Seq <k>: <mismatches of sequence data k with gaps>
```

8 Examples

8.1 Multiple Sequence Alignment

Suppose you want to align three strains auf *S. aureus*. You follow instructions a) in Section 3 and build an index named `3staph`:

```
mkvtree -dna -lcp -suf -tis -db NC_002745.fna NC_002758.fna MRSA.dbs\
 -indexname 3staph
```

You are now able to run `mga` and create your alignments. You decide to turn on verbose output and to always recurse into gaps. Using these two options is always a good idea for multiple sequences, though in this example, the second one does not make a difference. The output will be written to files whose names start with `example`.

```
mga -v -l 1000 20 -always -o example 3staph
```

The file `example.summary` contains the command line parameters of `mga`. Also, the description of the sequences as present in their FASTA headers is given. Then, some simple statistics follow: more than 90% of the sequences are aligned.

```
The program call arguments were

-v -l 1000 20 -always -o example 3staph

Sequence description:

Seq 1: gi|15925705|ref|NC_002745.1| Staphylococcus aureus subsp. aureus N315, \
complete genome
Seq 2: gi|15922990|ref|NC_002758.1| Staphylococcus aureus strain Mu50, \
complete genome
Seq 3: Staphylococcus aureus (EMRSA-16) chromosome 2,902,619 bp

Number of matches / aligned / unaligned gaps is 23054 / 22960 / 93

        length / all al. = matches + aligned / unaligned gaps

Seq 1: 2813641 / 2654636 = 2348504 +  306132 /  159005
Seq 2: 2878040 / 2652323 = 2348504 +  303819 /  225717
Seq 3: 2902619 / 2653076 = 2348504 +  304572 /  249543

Avg. : 2864766 / 2653345 = 2348504 +  304841 /  211421

Coverage in percent

Seq 1: 2813641 /    94.3 =    83.5 +    10.9 /     5.7
Seq 2: 2878040 /    92.2 =    81.6 +    10.6 /     7.8
Seq 3: 2902619 /    91.4 =    80.9 +    10.5 /     8.6

Avg. : 2864766 /    92.6 =    82.0 +    10.6 /     7.4

Please cite the following paper:

            Michael Hoehl, Stefan Kurtz, Enno Ohlebusch
            Efficient Multiple Genome Alignment
            Bioinformatics, Vol. 18 (S1):S312-S320, 2002
```

If `mga` is run as above the file `example.align` contains only the structure of the alignment, not the (aligned) sequence data itself. You can see a *multiMEM* in the first line, starting at position zero in all three sequences, then a gap which is five bases short, etc.:

```
61 0 0 0
5:61-65 5:61-65 5:61-65
85 66 66 66
72:151-222 71:151-221 72:151-222
131 223 222 223
```

When `mga` is run with the additional parameter `-match` it outputs the sequence data of matches. Recall that a match consists of bases that are identical in all sequences. Therefore, the bases are shown only once. The last line from the previous example is displayed as follows:

```
131 223 222 223
Exact: attagaaattacacacaaagttatactatttttagcaacatattcacaggtatttgacat        60
Exact: atagagaactgaaaaagtataattgtgtggataagtcgtccaactcatgattttataagg       120
Exact: atttatttatt                                                        131
```

When your are only interested in, say, 20 bases next to a gap, you can get this context information by supplying `mga` with parameter `-bl 20` instead:

```
131 223 222 223
Exact: attagaaattacacacaaag                                               20
       ...skipping bases...
Exact: tttataaggatttatttatt                                               20
```

You might want to know why the match starts one base earlier in sequence two. The preceding gap is one base shorter in that sequence. You can view the aligned sequence data by calling `mga` with parameter `-clustalw`. This outputs the Clustal W alignment of short gaps:

```
72:151-222 71:151-221 72:151-222
Seq 1: tcactaacagatattctatagaaggaaaagttatccacttatgcacatttatagttttca        60
Seq 2: ............................atg.....agca..-g............
Seq 3: a...........c..............................c.....c....t.

Seq 1: gaattgtggata                                                       72
Seq 2: ..........ct
Seq 3: ............
```

8.2 Pairwise Sequence Alignment

Now you are interested in aligning two strains of *E. coli*. In order to save about half the amount space, you decide to build an index comprised of the first sequence only according to instructions b) of Section 3.

```
mkvtree -dna -suf -tis -sti1 -pl 8 -db ecoli.fna
```

Using this index (which has the same name as the sequence it contains) means every time you run `mga` you have to specify the missing second sequence.

```
mga -l 20 -greedy -o example -mem ecoli_O157.fna ecoli.fna
```

The last `mga` run created a greedy alignment of short gaps which you can inspect in the file `example.align`:

```
73:229-301 90:229-318
Sbjct: ------------------ttaccacaggtaacggtgcgggctgacgcgtacaggaaacac        60
       !!!!!!!!!!!!!!!!!
Query: ccaccatcaccattaccattaccacaggtaacggtgcgggctgacgcgtacaggaaacac        60

Sbjct: agaaaaaagcccgcacctgacagtgcgggct                                    91
                                     !
Query: agaaaaaagcccgcacctgacagtgcgggc-                                    91
```

The query sequence, i.e. strain O157:H7 is approximately 900k bases longer than strain K-12 MG1655. This difference is reflected in a number of long, unaligned gaps where the region or part of the query sequence is substantially longer than the other one. The most extreme case is presented below:

```
2707:1634868-1637574 172666:2138604-2311269
Gap67:    2707 (min. length) +  169959 (diff.) =  172666 (max. length)
```

Suppose you want to align two strains of *M. tuberculosis* using MUMs as anchors. This is only possible when creating an index as before:

```
mkvtree -dna -suf -tis -sti1 -pl 8 -db mtub.fna
```

Now you want to align not just short gaps but indeed every gap that is similar in both sequences:

```
mga -l 50 -maxdist 1000 -greedy -o example -mum mtub_CDC1551.fna mtub.fna
```

You find several alignments that are not computed without option `-maxdist`. An excerpt of the longest one is shown below; note that it contains more than 10k bases.

```
10722:4134709-4145430 10721:4126967-4137687
Sbjct: gagagcggcgcagatgatcctaaccggacgcaccggcttgctggccctgatctgcgtcct      60
       !
Query: -agagcggcgcagatgatcctaaccggacgcaccggcttgctggccctgatctgcgtcct      60

Sbjct: gccgatagcgctgtccccttggccggcaagggctttcgtgatgttgctggtggcgcttgc     120
Query: gccgatagcgctgtccccttggccggcaagggctttcgtgatgttgctggtggcgcttgc     120
```

In order to obtain the sequence data of putative SNPs you either have to include option `-identity 0` in the previous call or you remove option `-maxdist` from it. Here is one putative SNP between two particularly long MUMs:

```
19138 3270781 3265104
!3289919 !3284242
Sbjct: c                                                                  1

Query: a                                                                  1

6919 3289920 3284243
```

9 DTD for XML Output

This section can be safely skipped if you are not interested in parsing the XML file. If you are interested, read on. The next section presents the DTD at one glance, the remaining sections explain it in detail.

9.1 The Complete DTD

```
<!ELEMENT MgaOutput (Summary, PartialMultipleAlignment)>
<!ELEMENT Summary (CommandLineArguments, SequenceDescription+,
                   SequenceInformation+, TotalNumber+)>
<!ELEMENT CommandLineArguments (#PCDATA)>
<!ELEMENT SequenceDescription (#PCDATA)>
<!ELEMENT SequenceInformation (Length+)>
<!ELEMENT TotalNumber (#PCDATA)>
<!ELEMENT Length (#PCDATA)>
<!ELEMENT PartialMultipleAlignment (Block+)>
<!ELEMENT Block (OneSequence+)>
<!ELEMENT OneSequence (Region, SequenceData)>
<!ELEMENT Region (Start, End)>
<!ELEMENT SequenceData (#PCDATA)>
<!ELEMENT Start (#PCDATA)>
<!ELEMENT End (#PCDATA)>
<!ENTITY % type "(match|aligned|gap)">
<!ATTLIST SequenceDescription seqid ID #REQUIRED>
<!ATTLIST SequenceInformation seqref IDREF #REQUIRED>
<!ATTLIST Length type (sequence|%type;) #REQUIRED>
<!ATTLIST TotalNumber type %type; #REQUIRED>
<!ATTLIST Block type %type; #REQUIRED>
<!ATTLIST OneSequence seqref IDREF #REQUIRED>
<!ATTLIST SequenceData type (gapfree|gapped) #REQUIRED>
```

9.2 Explanation of Element Rules

The XML output of `mga` consists of a summary of this particular run and of the computed partial multiple alignment.

```
<!ELEMENT MgaOutput (Summary, PartialMultipleAlignment)>
```

The summary itself is subdivided into the parameters `mga` was called with, the description or names of the sequences as found in their FASTA headers, other information about the sequences and finally some numbers.

```
<!ELEMENT Summary (CommandLineArguments, SequenceDescription+,
                   SequenceInformation+, TotalNumber+)>
```

The command line arguments are some sort of character data and not structured into subunits any further.

```
<!ELEMENT CommandLineArguments (#PCDATA)>
```

The sequence names are also unstructured.

```
<!ELEMENT SequenceDescription (#PCDATA)>
```

The additional information is comprised of several lengths.

```
<!ELEMENT SequenceInformation (Length+)>
```

The total numbers and the lengths are natural numbers.

```
<!ELEMENT TotalNumber (#PCDATA)>
```

```
<!ELEMENT Length (#PCDATA)>
```

A partial multiple alignment consists of at least one block. An extreme case is a single unaligned block, i.e., all sequences are in fact unaligned.

```
<!ELEMENT PartialMultipleAlignment (Block+)>
```

A block is comprised of all sequences which means at least two sequences, or else an alignment would not make sense.

```
<!ELEMENT Block (OneSequence+)>
```

A particular sequence (fragment) carries region information describing where it resides on the genome. Furthermore, its sequence data is stored.

```
<!ELEMENT OneSequence (Region, SequenceData)>
```

A region simply consists of a start and an end point.

```
<!ELEMENT Region (Start, End)>
```

Sequence data can contain any kind of character as `mga` per se does not restrict the alphabet to consist of A, C, G, T only.

```
<!ELEMENT SequenceData (#PCDATA)>
```

Start and end points are natural numbers.

```
<!ELEMENT Start (#PCDATA)>

<!ELEMENT End (#PCDATA)>
```

9.3 Explanation of Parameter Entity

In Section 9.4, several entities receive an attribute named “type”. The values “match” and “aligned” denote anchors and closed gaps: sequence fragments of both types are considered aligned. The remaining value “gap” denotes an unaligned gap.

```
<!ENTITY % type "(match|aligned|gap)">
```

9.4 Explanation of Attribute Rules

Every sequence respective its name receives a unique identifier.

```
<!ATTLIST SequenceDescription seqid ID #REQUIRED>
```

All information can be assigned to its corresponding sequence by way of a reference to the previously mentioned identifier.

```
<!ATTLIST SequenceInformation seqref IDREF #REQUIRED>
```

Length information is given for the complete sequence and for every kind of type.

```
<!ATTLIST Length type (sequence|%type;) #REQUIRED>
```

For every type, the total number of bases is stored which are contained in that particular type.

```
<!ATTLIST TotalNumber type %type; #REQUIRED>
```

As was shown in Section 9.2, a block groups together excerpts of sequences. These excerpts can be considered aligned or unaligned as denoted by their type.

```
<!ATTLIST Block type %type; #REQUIRED>
```

Every sequence (fragment) references one of the descriptions.

```
<!ATTLIST OneSequence seqref IDREF #REQUIRED>
```

All sequence data are annotated whether they may contain gaps as indicated by the gap character '-' or whether they are guaranteed to be free of gaps. Anchors and unaligned gaps are of type "gapfree".

```
<!ATTLIST SequenceData type (gapfree|gapped) #REQUIRED>
```

10 Version History and Release Notes

10.1 Version 2003-03-18: Release of Mar 18 2003

- Speed-up of chaining phase
  Previously, *m multiMEMs* occuring in $k > 2$ sequences were chained in time $O(k \cdot m^2)$, i.e., the running time of the algorithm was quadratic in the number *m* of *multiMEMs*. This can be a serious drawback if *m* is large. To overcome this obstacle, we implemented a variant of an algorithm devised by Zhang et al. [ZRHM94]. This algorithm takes advantage of the geometric nature of the problem. It is based on *kd*-trees, a data structure known from computational geometry [Ben90]. As is typical with *kd*-tree methods, no rigorous analysis of the running time of the algorithm is known. However, our experiments show that it gives a considerable speed-up in practice.

- Using arbitrary alignment tools
  Previously, only Clustal W could be used to close short gaps between *multiMEMs* occuring in $k > 2$ sequences. Furthermore, its parameters were fixed. Now, virtually any multiple (or pairwise) alignment tool and parameter setting can be used with the help of wrapper scripts which are passed to option `-msascript`.

- Same chaining output between Linux and Solaris
  Previously, the output of the chaining phase could differ depending on the operating system. This could happen when there was a choice between several equally long subchains. Furthermore, this could lead to a different overall number of chained *multiMEMs*—especially when the resulting gaps were aligned recursively. Consequently the alignment could differ substantially. Now the chaining phase will always get the same results independant of the operating system. Using the same parameter settings for `mga` will aways result in the same chain and alignment. Note that because of the changes the alignment produced by `mga` may differ from that of earlier versions. Furthermore, chains may differ depending on the (not recommended) use of option `-nofast` for three or more sequences.

- Sequence limit displayed
  The command line output now shows the maximum number of sequences `mga` can handle.

- Changes to options
  new: options `-msascript`, `-alignedseqs` and `-width`
  modified: options `-xml` and `-nofast`
  consistent: options `-clustalw` and `-greedy` cannot be combined
  no limit: options `-gl` and `-maxdist` can receive values higher than $5,000$

10.2 Version 2002-08-26: Release of Aug 26 2002

- Citation of `mga` paper changed to display volume and page numbers.

10.3 Version 2002-07-25: Release of Jul 25 2002

- First public release of `mga`

References

[Ben90] J.L. Bentley. K-d trees for Semidynamic Point Sets. In *Proceedings of the 6th Annual ACM Symposium on Computational Geometry*, pages 187–197, 1990.

[Gil01] D. Gilbert. Readseq, May 2001. The latest version is available from `http://iubio.bio.indiana.edu/soft/molbio/readseq/java/`.

[HKO02] M. Höhl, S. Kurtz, and E. Ohlebusch. Efficient Multiple Genome Alignment. In *Proceedings of the Tenth International Conference on Intelligent Systems for Molecular Biology.* Bioinformatics, volume 18 supplement 1, pages S312–S320, 2002.

[Kur02] S. Kurtz. *Construction of Virtual Suffix Trees*. Technische Fakultät, Universität Bielefeld, May 2002. Manual.

[THG94] J.D. Thompson, D.G. Higgins, and T.J. Gibson. *CLUSTAL W*: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position Specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Research*, 22:4673–4680, 1994. The latest version is available from `ftp://ftp.ebi.ac.uk/pub/software/unix/clustalw/`.

[ZRHM94] Z. Zhang, B.R. Raghavachari, R.C. Hardison, and W. Miller. Chaining Multiple Alignment Blocks. *J. Comp. Biol.*, 1(3):217–226, 1994.