Using MUMmer to Identify Similar Regions in Large Sequence Sets

The MUMmer sequence alignment package (Delcher et al., 1999, 2002) is a suite of programs to detect similar regions between biological sequences. The programs are specifically designed to rapidly detect regions of very high similarity in very long sequence sets, such as complete bacterial genomes or entire eukaryotic chromosomes.

The MUMmer package is set of programs, all of of which are designed to be as independent as possible so that the user can employ them as parts of other scripts. Most of the programs have simple text input and output formats that can easily be edited or modified by other scripts. Several Unix shell scripts are provided to run the system in the modes described below.

The methodology employed by all of the MUMmer scripts is to first identify a set of identical regions between the sequences being compared, and then cluster/combine the exact matches into larger regions of similarity. Alignments are provided for regions where the match is inexact. The fundamental type of exact match is a maximal unique match (MUM). This is an exact match between two sequences that occurs exactly once in each sequence (uniqueness) and is not contained within any larger match (maximality). The MUMmer programs use suffix trees to find exact matches (Gusfield, 1997).

In what follows, the terms *query sequence* and *reference sequence* refer to the two sequences (or sets of sequences) being compared. Depending on the protocol, there may be restrictions on the format of one of these, and in general the algorithms are not symmetric with respect to these sequences, so that different results may be obtained if the roles of the query and reference sequences are reversed.

The Basic Protocol describes the use of the MUMmer2 script to identify regions of similarity between a collection of DNA sequences and a single reference sequence. Three alternate protocols are then described. The NUCmer protocol (Alternate Protocol 1) relaxes the restriction that requires the reference to be a single sequence, thus allowing a set of multiple DNA sequences to be compared to another such set. The PROmer protocol (Alternate Protocol 2) compares DNA sequences by performing a 6-frame protein translation and then searching for matches in the resulting protein space. PROmer generally is able to detect more divergent matches that are too dissimilar to be detected directly at the DNA level by NUCmer. Alternate Protocol 3 uses the original MUMmer1 script to compute the longest single consistent alignment between two single DNA sequences. Finally, the Support Protocol describes how to obtain the MUMmer package from the TIGR Web site. Each of these protocols runs in the Unix environment. Users unfamilar with Unix should consult *APPENDIX 1C*.

MUMmer2: COMPARING A SET OF SEQUENCES TO A SINGLE REFERENCE SEQUENCE

BASIC PROTOCOL

This protocol takes a collection of query sequences and searches for unambiguous alignments between them and a single reference sequence. For example, a user could use this protocol to find alignments between shotgun sequence fragments and a reference genome, or to compare the genomes of closely related species, or to quickly compare alternate assemblies of the same genome.

It is important to note that not all matches are found—if there are repeat regions in the reference sequence that cannot be distinguished, then matches to neither will be reported.

Contributed by Arthur L. Delcher, Steven L. Salzberg, and Adam M. Phillippy
Current Protocols in Bioinformatics (2003) 10.3.1-10.3.18

Thus, this protocol should not be used when a comprehensive list of all possible matches is desired. Rather, its purpose is to identify high-fidelity matches as quickly as possible.

The protocol presents two ways of running MUMmer. The preferred method requires the user to invoke each of the programs from the Unix command line in turn. As an alternative, a Unix shell script is provided with the system.

This script uses only default parameter settings; however, the user is strongly encouraged to understand and modify the options of the different commands to suit his or her situation. Users unfamiliar with Unix should consult *APPENDIX 1C*.

Necessary Resources

Hardware

Unix or Linux workstation. The largest program used in this protocol requires main memory of approximately 20 bytes per base of reference sequence plus 1 byte per base of query sequence. Thus, to compare 2 million bases of query sequence to 3 million bases of reference sequence, the computer should have at least (20 × 3 Mb) + (1 × 2 Mb) = 62 Mb of main memory.

Software

MUMmer 2.12 package (see Support Protocol for download and installation)

Files

A multi-FASTA query file and a single-FASTA reference file (see *APPENDIX 1B* for information on FASTA). The files used in this example are complete genomic sequences from two strains of *Helicobacter pylori*—known as 26695 and J99. These sequences can be downloaded from TIGR's Comprehensive Microbial Resource at *http://www.tigr.org/CMR*, from the NCBI at *http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/micr.html*, or from the *Current Protocols in Bioinformatics* Web site at *http://www3.interscience.wiley.com/c_p/cpbi_sampledatafiles.htm*.

Running MUMmer Interactively from the Command Line

Find MUMs between the query and reference sequences

1. Determine the minimum-length MUM to be reported. In general, large values should be used for highly similar sequences and for long sequences, whereas lower values are better for less similar and shorter sequences. A reasonable starting value for bacterial genomes is 20, which is the value that will be used in this example.

   Because this program runs very quickly, the user can run the program with different values of this parameter and determine which is best based on the resulting output. Generally, there is little point in using a value less than the length of match one would expect for random sequence, which for DNA sequences is ~$\log_4 R$, where R is the length of the reference sequence. For example, for a 4-million base genome, $\log_4 R = 11$ and the minimum MUM length should be significantly longer than 11 bp.

2. Determine the orientations of the sequences to be compared. In the comparison, the user has three choices: use only the forward orientation of the query sequences (default); use only the reverse orientation of the query sequences (`-r` option); or use both forward and reverse orientations (`-b` option). This example will use both forward and reverse orientations.

3. Run the program `mummer2` from the Unix command line. For this example, the user would type:

```
mummer2 -b -c -l20 hp26695.seq hpj99.seq >j99vs26695.out
```

```
> gi|12057207|gb|AE001439.1|AE001439
     183        17        22
     238        72       108
     347       181        92
     458       292        50
     509       343        35
     545       379        94
     640       474        25
     666       500        38
     705       539        44
     750       584        38
     807       641        23
     834       668        35
     870       704        24
     921       755        26
     966       800        29
    1014       848        20
       .         .         .
       .         .         .
       .         .         .
```

Figure 10.3.1 Beginning of output from program `mummer2` for *H. pylori* strains 26695 and J99. The first line is the FASTA header line from the query sequence in the file `hpj99.seq`. The following lines list the locations of all MUMs between this sequence and the reference sequence, which was in the file `hp26695.seq`. The first column is the beginning position of the MUM in the reference sequence 26695; the second column is the beginning position of the MUM in the query sequence J99; and the third column is the length of the MUM. The MUMs are listed in order by position in the query sequence.

`hp26695.seq` and `hpj99.seq` are names of the files containing the genomes of strains 26695 and J99 in FASTA format.

The MUMs found by the `mummer2` *program are identical sequences contained in both the query and reference strings, such that these sequences are not contained in any larger matches and are unique in the reference string. Note that these matches may not be unique in the query string, so that it is possible that two or more locations in the query string may match the same position in the reference string. If the user desires MUMs that are unique in both sequences, the program* `mummer1` *can be used instead of* `mummer2`*. However, this program uses substantially more memory than* `mummer2`*.*

The `-b` *option means that both the forward and reverse-complement orientations will be used (independently) to find MUMs. The* `-c` *option means that the positions of MUMs on the reverse-complement strand will be reported relative to the original string, instead of relative to the reverse-complement string, which is the default without the* `-c` *option. For example, a MUM of length 20 reported at position 20 on the reverse strand using the* `-c` *option is at positions* `1..20` *of the original reference string, whereas without the* `-c` *option, a MUM of length 20 reported at position 20 on the reverse strand is at positions* `20..39` *from the end of the original reference string. The* `-l20` *option means that only MUMs of length at least 20 will be output. Note that this option is a lower-case letter "ell" (*`l`*). The end of the command,* `>j99vs26695.out`*, directs the output of the command to a file named* `j99vs26695.out`*. The beginning of this file is shown in Figure 10.3.1, and the beginning of the portion of the file showing MUMs on the reverse-complement strand is shown in Figure 10.3.2. For each sequence in the query file (and its reverse-complement if the* `-b` *option is used), the locations of all MUMs between it and the reference sequence are given. The format is one MUM per line, where the first column is the starting position of the MUM in the reference sequence, the second column is the starting position of the MUM in the query sequence, and the third column is the length of the MUM.*

```
> gi|12057207|gb|AE001439.1|AE001439 Reverse
 1314091   1637275        22
 1348297   1626178        20
 1612843   1623008        23
 1283825   1615114        22
 1615880   1611836        31
 1615915   1611801        38
 1616062   1611654        37
 1616125   1611591        23
 1616149   1611567        25
 1616175   1611541        30
 1616236   1611480        74
      .         .         .
      .         .         .
      .         .         .
```

Figure 10.3.2 Beginning of the second part of the output from program `mummer2` for *H. pylori* strains 26695 and J99 showing the matches for the reverse-complement strand (indicated by the word `Reverse` at the end of the header line) of the query sequence J99. The format is the same as in the previous figure. If the query file contained more than one sequence, the matches for each sequence would follow in the same format, with each set of matches preceded by the FASTA header line of the sequence in the query file.

Cluster MUMs into consistent sets of matches

4. Determine parameters for grouping MUMs into clusters. Because MUMs are exact matches, to find similar regions that are not exact, MUMs that are likely to belong in the same region of similarity are grouped together. There are several parameters that determine how clusters are formed, all of which are specified as options to the program `mgaps`:

   a. Maximum separation between consecutive MUMs in a cluster, as measured by the number of bases between the MUMs in the query sequence. This is set by the `-s` option. For highly similar sequences use a small value; for dissimilar sequences use a large value.

   b. Maximum diagonal difference between consecutive MUMs in a cluster. The diagonal of a MUM is the difference between its position in the query and its position in the reference. The diagonal difference between consecutive MUMs is a measure of the size of insertions or deletions between them. There are two ways to specify the diagonal difference. The first is to set a fixed maximum diagonal difference with the `-d` option. The second is to set the maximum diagonal difference as a fraction of the separation, which is done with the `-f` option. If both options are used, then clusters meeting either criterion will be output.

   c. Minimum cluster length. This is set by the `-l` option. By default, the length of a cluster is defined as the sum of the lengths of the MUMs within it. If the `-e` option is specified, then the length of a cluster will be the distance from the start of the first MUM in the cluster to the end of the last MUM in the cluster as measured in the reference sequence.

5. Run the program `mgaps` from the Unix command line. For this example, the user would type:

```
mgaps -l100 -f0.12 -s600 <j99vs26695.out >j99vs26695.gaps
```

The `-l` option sets the minimum cluster length to 100. The `-f` option sets the maximum diagonal difference between consecutive MUMs to 12% of the separation. The `-s` option

```
#
   131579   124441     74    none      -      -
   131654   124516     30    none      1      1
   131685   124547    100    none      1      1
   131785   124710     17      -6      0     63
   131843   124747     29    none     41     20
   131893   124797    111    none     21     21
   132296   125201     20    none    292    293
   132316   125222     16      -5      0      1
   132377   125283    182    none     45     45
   132560   125466     41    none      1      1
      .        .      .       .       .      .
      .        .      .       .       .      .
      .        .      .       .       .      .
```

Figure 10.3.3 Part of the output from program `mgaps` showing the beginning of one of the clusters of MUMs. As before, the FASTA header lines from the query file are reproduced and clusters for the same query are separated by lines containing a single # character. The first three columns are the MUMs. The fourth column indicates any overlap between MUMs. The final two columns are the number of characters between the start of the MUM and the end of the preceding MUM in the reference and query sequence, respectively.

sets the maximum separation between consecutive MUMs to 600. Input is taken from the file `j99vs26695.out` *(the output of the preceding step) and output is directed to the file* `j99vs26695.gaps`*. The output file has the FASTA header lines from the input file separating the clusters for each query string and its reverse complement. Different clusters for the same query string are separated by a line containing a single pound (#) character. The beginning of one of the clusters of MUMs in file* `j99vs26695.gaps` *is shown in Figure 10.3.3. The first three columns are the same format as the input file: the start position of the MUM in the reference string, the start in the query string, and the length of the MUM. The fourth column is an indication of whether the MUM overlaps the preceding MUM in the cluster. If not, the word* `none` *appears in this column; otherwise, the number of characters of overlap is indicated as a negative number (see the next section for an example). The final two columns show the number of characters between the MUM and the preceding MUM in the cluster—the fifth column is in the reference sequence, the sixth is the query sequence.*

Compute alignments in regions between MUMs in clusters

6. Determine the error-rate parameter for computing alignments. This is set by the `-e` option of program `combineMUMs`.

   When computing alignments in gaps between consecutive MUMs in a cluster, the program computes the alignment from the ends of the MUMs bounding the gap. The alignment stops when the error rate exceeds the value specified, and the cluster is split into separate clusters at that point.

7. Run the program `combineMUMs` from the Unix command line. For this example, the user would type:

```
combineMUMs -x -e0.10 -W j99vs26695.witherrs hp26695.seq
hpj99.seq j99vs26695.gaps >j99vs26695.align
```

   The `-e` *option sets the alignment error rate to 10%: when the number of errors in the alignment exceeds 10% of the length of the alignment, the alignment is aborted. The reference sequence is* `hp26695.seq`*, the query sequence is* `hpj99.seq` *as before, and* `j99vs26695.gaps` *is the output from the prior step. The* `-w` *option specifies the name (*`j99vs26695.witherrs` *in this example) of an additional output file described below. The alignment output is directed to the file* `j99vs26695.align`*. The* `-x` *option*

```
#
  131579   124441     74    none      -      -
  131654   124516     30    none      1      1
     Errors = 1
A: gtgtattaatggtcttttgtt
B: gtgtattaattgtcttttgtt
             ^

  131685   124547    100    none      1      1
     Errors = 1
A: taactctatcgcactcttatt
B: taactctatctcactcttatt
             ^

  131785   124710     17      -6      0     63
     Errors = 63
A: tgtttgtttt----------------------------------------------------
B: tgtttgtttttttgtctttcttgttctaactctatctcactcttatttgtcttctgtcttt
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

A: -------------ctgtctttct
B: cttgttctgttttctgtctttct
   ^^^^^^^^^^^^^

  131843   124747     29    none     41     20
     Errors = 24
A: tcttgttctagttctatcccactcttgttcgctctatccctagcgttttcagcttccttt
B: tcttgttctaactttatcccactctt---------at---t----t-----gcttccttt
           ^^ ^            ^^^^^^^^^  ^^^ ^^^^ ^^^^^

A: t
B: t

  131893   124797    111    none     21     21
     Errors = 3
A: ctagttctatctgtttatcagtatcaccagcgctacaagcg
B: ctagttctatttgtttatcaatatcaccagtgctacaagcg
           ^          ^          ^

  132296   125201     20    none    292    293
     Errors = 36
A: ctttatcgtctaattctttcgctttcttgcacgcaatatgggtcaatactactaatgccg
B: ctttatcgtccaattcttttgccctcttgcacgcaatatgggtcaatactactaatgcca
           ^         ^  ^^                                    ^
A: cactgcttttcttggggtgtttttttacaagtcctctaactctctttgcaatttttgtaa
B: cactgcttttcttgggatgtttttttcacaagttctctaactttctttgcaatttttgtaa
                ^              ^          ^          ^
A: aaaacccaccaactgatctgattatatttgtaatggcgctcacttgtttaaaatgagtct
B: aaatcccaccaattgatcggagcgtatttgtaatggcactcgcttgtttaaaatgagtct
     ^         ^     ^  ^^^             ^   ^
A: ctttatttgcctttgtattagccatctctgtattcttgccaaccttatttgtttttcctg
B: ctttacttgcctctgtgttagctgtctctgtgtttttgccaactttatttgtttttactg
      ^      ^  ^      ^^        ^  ^         ^            ^
A: tttttactgattccattaccaacctttytttaamaaacaatcttactttwcaaaatyata
B: tttttactgattccattaccaacctttttttcaaaaaacaatcttacccatcaaaactata
                           ^  ^  ^            ^^^^     ^^
A: tc-atggaaacta
B: tcaatggaaacta
    ^

  132316   125222     16      -5      0      1
     Errors = 1
A: atacgaaaaa-tccactcaag
B: atacgaaaaaatccactcaag
           ^

  132377   125283    182    none     45     45
     Errors = 12
A: caagttagattgatttatttattatcaatgttcagttaagccatgttcttttagttcttt
B: caagttagatcaacccctttattatcaacgtgcagttaagccatggtctttcaactcttt
           ^^ ^^^^           ^  ^              ^     ^ ^^
A: ttcta
B: ttcta

  132560   125466     41    none      1      1
              .
              .
              .
Region:     131579 .. 139347        124441 .. 132253         301 / 7813
3.85%
```

Figure 10.3.4 Part of the output from program `combineMUMs` showing the beginning of the same cluster as in the previous figure. Note the alignment above for MUM `131785 124710 17`, which overlaps the preceding MUM by 6 characters (indicated by the `-6` in the column that has `none` when there is no overlap). The overlap can be seen in that the last 6 characters before the gap match the last 6 characters in the gap. The phenomenon of overlapping MUMs generally indicates a variation in the number of occurrences of a tandem repeat in the two sequences. In this instance, the 63-character insertion occurred within a repeated run of `gtttt`, where the B sequence had one more occurrence of `gtttt` than did the A sequence.

suppresses the output of two coverage files, which simply show the percentage of N's in each cluster and the regions between them in the two input files.

The alignment output for the same cluster shown previously is shown in Figure 10.3.4. This output duplicates the information in the `.gaps` *file, but interlaces it with alignment information. Specifically, after each MUM (except the first) in a cluster, the alignment of the gap separating that MUM from the preceding MUM is shown. Each alignment begins with a line stating the number of errors in the alignment and errors are indicated by carets (*`^`*) in the alignment itself. The alignment is then shown; for clarity, it begins with the last 10 characters from the preceding MUM and includes the first 10 characters from the current MUM as a point of reference. The* `A` *sequence in the alignment is the query sequence and the* `B` *sequence is the reference sequence. Additionally, at the end of a cluster, the extent of the cluster and an overall alignment score for it is given. Note that if an alignment fails to span a gap in a cluster, then the cluster will be split. Thus, the clusters in the* `.gaps` *file will not necessarily match those in the* `.align` *file.*

An additional file, in this example named `j99vs26695.witherrs`*, is also produced by the* `combineMUMs` *program. This file is identical to the input* `.gaps` *file, with the exception that a seventh column is added, which indicates the number of errors in the alignment between the MUM and its predecessor. If an alignment fails to span a gap (because of the error-rate threshold or because the gap is too large), no value is reported in the seventh column and there is simply a dash there. The name of this file is specified by the* `-W` *option. If this option is not used, the default name of this file is* `wither-rors.gaps`.

Running MUMmer via the Shell Script

As an alternative to performing each of the above steps separately, a Unix shell script named `run-mummer2.csh` is provided with the system. It is invoked with three parameters: the reference file name, the query file name, and a tag for outputs. In the example above, the user would simply type:

```
run-mummer2.csh hp26695.seq hpj99.seq j99vs26695
```

This script uses only default parameter settings, however, and the user is strongly encouraged to understand and modify the options of the different commands to suit the particular situation.

NUCmer: COMPARING A SET OF SEQUENCES TO ANOTHER SET OF SEQUENCES

ALTERNATE PROTOCOL 1

The NUCmer package is a wrapper around the basic MUMmer programs that extends MUMmer by permitting the reference sequence to be a set of strings, in the form of a multi-FASTA file (*APPENDIX 1B*). NUCmer accomplishes this by first concatenating all the reference strings together into a single string, then running MUMmer on that single string, then splitting the results back to appropriate coordinates on the separate strings.

Necessary Resources

Hardware

Unix or Linux workstation. The largest program in the suite requires main memory of approximately 20 bytes per base of reference sequence plus 1 byte per base of query sequence. Thus, to compare 2 million bases of query sequence to 3 million bases of reference sequence, the computer should have at least (20 × 3 Mb) + (1 × 2 Mb) = 62 Mb of main memory.

Software

NUCmer is included in the MUMmer 2.12 package (see Support Protocol for download and installation)

Files

A multi-FASTA query file and a multi-FASTA reference file (see *APPENDIX 1B* for information on FASTA). The files used in this example are sequences extracted from alignment regions of the *H. pylori* genomes used in the Basic Protocol. File `26695parts.seq` has five 2-kb sequences extracted in order from file `hp26695.seq`, and file `j99parts.seq` has five corresponding 2-kb sequences from file `hpj99.seq` but in permuted order, with 2 sequences reversed. The positions of the sequences in the files from which they were extracted are indicated in the FASTA header lines in the files. These files can be obtained from the *Current Protocols in Bioinformatics* Web site at *http://www3.interscience.wiley.com/c_p/cpbi_sampledatafiles.htm*. The first field after the `>` on the FASTA header line of each sequence will be used to identify each sequence; therefore these field should be unique both within and between the query and reference files.

Edit and run the NUCmer script

1. Determine the desired alignment parameters. The most important parameters correspond to those used in the Basic Protocol and are set by options on the NUCmer script command line. Specifically they are:

   `-l` the minimum MUM size to use (default value, 20)
   `-g` the largest separation in the `mgaps` program (default value, 90)
   `-c` the minimum cluster length in `mgaps` (default value, 65)
   `-d` the diagonal difference fraction in `mgaps` (default value, 0.12).

   There are several additional parameters that can be set by command-line options.The `-a` *option can be used to specify the program that finds the initial matches. Possible values for this parameter are* `mummer2` *and* `max-match`*. The first is the default and finds matches as described previously.* `max-match` *finds all maximal exact matches between the query and reference sequences with no regard for uniqueness. The* `-b` *option can be used to specify the length that alignments are extended into poor-scoring regions before being abandoned. Its default value is 200. The* `-f` *option specifies that only the forward strand of the query sequence should be used. The* `-r` *option specifies that only the reverse strand of the query sequence should be used. The default behavior is to use both the forward and reverse strands. The* `-p` *option can be used to specify the file name prefix to use for outputs. The* `-o` *option directs NUCmer to automatically generate the* `.coords` *file described below and should be used in most cases.*

   This example uses the default option values in the script as downloaded, plus the `-o` *and* `-p` *options.*

2. Run the NUCmer script. For this example, the user would type:

```
nucmer -o -p nuc 26695parts.seq j99parts.seq
```

   The `-o` *option will generate the* `.coords` *output file and the* `-p` *option specifies the prefix tag to use for the output files. The last two parameters are the reference and query sequence files. These files may be masked to indicate regions within them that should not be matched. For example, the user might employ a utility such as* `dust` *to identify and mask low-complexity sequence. The format of the masking is to replace the bases to be ignored with X's.*

   The result of the script will be three files: `nuc.cluster`*,* `nuc.coords`*, and* `nuc.delta`*. The file* `nuc.cluster` *contains clusters of MUMs found between any of the query sequences and any of the reference sequences. Each cluster has a FASTA-style header line indicating which reference and which query sequence are involved and the orientations of those sequences. The file* `nuc.coords` *contains summary information about all matches found by computing alignments in cluster gaps and off the ends of clusters. The contents of this file are shown in Figure 10.3.5. The file* `nuc.delta` *contains information about alignments in the matches and is not intended for direct viewing, but is input to the* `show-coords` *and* `show-aligns` *programs.*

```
26695parts.seq j99parts.seq
NUCMER

    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  | [TAGS]
================================================================================
       1     2000  |        1     2000  |     2000     2000  |    95.95  | 26695.seq1    j99.seq3
       1     2000  |     2000        1  |     2000     2000  |    93.67  | 26695.seq2    j99.seq2
     178      881  |      187      894  |      704      708  |    92.26  | 26695.seq3    j99.seq5
    1218     1905  |     1313     2000  |      688      688  |    96.08  | 26695.seq3    j99.seq5
       1     1997  |     2000        1  |     1997     2000  |    91.42  | 26695.seq4    j99.seq4
       1     1965  |        1     1981  |     1965     1981  |    91.59  | 26695.seq5    j99.seq1
```

Figure 10.3.5 Contents of file `nuc.coords`. This file summarizes every match found between a query sequence and a reference sequence. The first two columns indicate the position of the match in the reference sequence. The next two columns are the position of the match in the query sequence. Note that if the start position [`S2`] is greater than the end position [`E2`] then the match is to the reverse-complement strand of the query sequence. Columns 5 and 6 are the lengths of the matches in the respective sequences. Column 7 is the percentage of bases that match between the two sequences. Columns 8 and 9 are the tags of the sequences—these are the first fields on the FASTA header lines of the sequence in the input files. Optionally, in column 10 there may be a note about the match (not shown). A note of `[DUPLICATE]` means this match is identical to the one preceding it. `[CONTAINS]` indicates this match contains the preceding match within it. `[SHADOWED]` indicates this match is contained within the preceding match. `[OVERLAPS]` indicates this match shares some positions with the preceding match. These conditions occur because overlapping MUMs may fail to combine with one another.

```
26695parts.seq j99parts.seq

============================================================
-- Alignments between 26695.seq1 and j99.seq3

-- BEGIN alignment [ +1 1 - 2000 | +1 1 - 2000 ]


1          agtgattagtgattatagcatcatttttaaatttaggtataaaacacc
1          agtgattagtgattacagcatcatttttaaatttaggcataaaacgcc
                        ^                         ^      ^


50         ctcaattcaagggtttttgagtgagctttttgctcaaagaatccaagat
50         cttaaatcaagggtttttgagcgagctttttgctcaaagaatccaagat
             ^  ^               ^


99         agcgtttaaaaatttaggggtgttaggctcagcgtagagtttgccaagc
99         agcgtttaaaaatttaggggtgttaggctcagcgtagagtttgccaagc


148        tctatgcattcattgatgatgatagggttttgcgtgggcgtgaagccaa
148        tctatgcattcattgatgatgatagggttttgtgtgggcgtgaagccaa
                                        ^


197        tttcatacgctcctaagcgtaaaatcgccttttccatgctccctaatcg
197        tttcatacgctcctaagcgtaaaatcgccttttccatgctccctaatcg


246        cttgaaatcccagtcttttaaatgcggctcgatgagggcgtcaatttca
246        cttgaaatcccagtcttttaaatgcggttcgatgagagcgtcaatctca
                                     ^        ^        ^
     ⋮
```

Figure 10.3.6 Beginning of an alignment created by the `show-aligns` program for NUCmer output.

View alignments

3. Run the `show-aligns` program to view alignments between a query sequence and a reference sequence. For example, to view the alignment for the first match listed in the `nuc.coords` file, the user would enter:

```
show-aligns nuc.delta 26695.seq1 j99.seq3
```

where `nuc.delta` is the file produced in the previous step and `26695.seq1` and `j99.seq3` are the sequence tags of the reference and query sequences in the match. The beginning of the resulting alignment is shown in Figure 10.3.6.

The sequence tags are listed on the right of each entry in the `nuc.coords` *file. Since this output is 175 lines long, the user may wish to direct it to an output file by appending* `> show-aligns.out` *to the end of the above command line.*

ALTERNATE PROTOCOL 2

PROmer: COMPARING SEQUENCES USING PROTEIN TRANSLATIONS

The PROmer package is another wrapper around the basic MUMmer programs, which allows users to detect similarity between species whose DNA sequences have diverged too much, erasing most or all significant nucleotide matches. It accomplishes this by first translating the sequences into their protein equivalents in all six reading frames, then running a NUCmer-like script on these sequences to detect matches, and finally converting the results back to appropriate coordinates in the original sequences. Proteins produced by orthologous genes are usually conserved long after the nucleotide sequences of the respective genes have diverged substantially. By searching for matches in amino acid sequences, PROmer is often able to detect matches that NUCmer would not.

Necessary Resources

Hardware

Unix or Linux workstation. The largest program in the suite requires main memory of approximately 20 bytes per base of reference sequence plus 1 byte per base of query sequence. Thus, to compare 2 million bases of query sequence to 3 million bases of reference sequence, the computer should have at least (20 × 3 Mb) + (1 × 2 Mb) = 62 Mb of main memory.

Software

PROmer is included in the MUMmer 2.12 package (see Support Protocol for download and installation)

Files

A multi-FASTA query file and a multi-FASTA reference file (see *APPENDIX 1B* for information on FASTA). The files used in this example are sequences extracted from alignment regions of the *H. pylori* genomes used in the Basic Protocol. File `26695parts.seq` has five 2-kb sequences extracted in order from file `hp26695.seq`, and file `j99parts.seq` has five corresponding 2-kb sequences from file `hpj99.seq` but in permuted order, with 2 sequences reversed. The positions of the sequences in the files from which they were extracted are indicated in the FASTA header lines in the files. These files can be obtained from the *Current Protocols in Bioinformatics* Web site at *http://www3.interscience.wiley.com/c_p/cpbi_sampledatafiles.htm*. The first field after the `>` on the FASTA header line of each sequence will be used to identify each sequence; therefore these field should be unique both within and between the query and reference files.

Edit and run the PROmer script

1. Determine the desired alignment parameters. The fundamental parameters are the same as in the preceding NUCmer protocol (Alternate Protocol 1). Note that all numeric values are now in terms of amino acid residues instead of nucleic acid bases.

   `-l` the minimum MUM size to use (default value, 6)
   `-g` the largest separation in the `mgaps` program (default value, 30)
   `-c` the minimum cluster length in `mgaps` (default value, 20)
   `-d` the diagonal difference fraction in `mgaps` (default value, 0.11)
   `-b` the maximum bad alignment extension (default value, 60)
   `-a` the initial match algorithm (default value, `mummer2`)

   There are two additional parameters that can be set by command-line options. The `-m` *option can be used to specify the number of amino acid residues between stop codons that will be ignored in comparisons. If this value is 6, for example, any string of* `6` *or fewer amino acids between stop codons will be excluded from the search for matches. The default value is 8. The* `-x` *option can be used to specify which BLOSUM matrix (Henikoff et al., 2000; UNIT 3.5) will be used to compute protein alignment. Possible values are* `1`, `2`, *and* `3` *corresponding to BLOSUM 45, 62, and 80 matrices, respectively. The default value is* `2` *for the BLOSUM 62 matrix.*

   This example again uses the default parameter values in the script as downloaded, plus the `-o` *and* `-p` *options.*

2. Run the PROmer script. For this example, the user would type:

```
promer -o -p pro 26695parts.seq j99parts.seq
```

   The `-o` *option will generate the* `.coords` *output file and the* `-p` *option specifies the prefix tag to use for the output files. The last two parameters are the reference and query sequence files. These files may be masked to indicate regions within them that should not be matched. For example, the user might use a utility such as dust to identify and mask low complexity sequence. The format of the masking is to replace the bases to be ignored with X's.*

   The result of the script will be three files: `pro.cluster`, `pro.coords` *and* `pro.delta`. *File* `pro.cluster` *contains clusters of MUMs found between any of the query sequences and any of the reference sequences. Each cluster has a FASTA-style header line indicating which reference and which query sequence are involved. In addition, the header line indicates the respective orientation (positive or negative) of the two sequences and the frames of the matches. File* `pro.coords` *contains summary information about all matches found by computing alignments in cluster gaps and off the ends of clusters. The contents of this file are shown in Figure 10.3.7. File* `pro.delta` *contains information about alignments in the matches and is not intended for direct viewing, but is input to the* `show-coords` *and* `show-aligns` *programs.*

View alignments

3. Run the `show-aligns` program to view alignments between a query sequence and a reference sequence. For example, to view the alignment for the first match listed in the `pro.coords` file, the user would enter:

```
show-aligns pro.delta 26695.seq1 j99.seq3
```

   where `pro.delta` is the file produced in step 2 and `26695.seq1` and `j99.seq3` are the sequence tags of the reference and query sequences in the match. The beginning of the resulting alignment is shown in Figure 10.3.8.

   The sequence tags are listed in columns 12 and 13 in the `pro.coords` *file. This output is often long and the user may wish to direct it to an output file by appending* `>show-aligns.out` *to the end of the above command line.*

```
95parts.seq j99parts.seq
MER

 [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  [% SIM]  [% STP]  | [FRM]  [TAGS]
==========================================================================================================
    1     1998  |        1     1998  |     1998     1998  |    90.99    93.09     6.16  |  1  1  26695.seq1    j99.seq3
    2     1999  |        2     1999  |     1998     1998  |    93.99    95.35     4.88  |  2  2  26695.seq1    j99.seq3
    3     2000  |        3     2000  |     1998     1998  |    90.09    92.49     6.83  |  3  3  26695.seq1    j99.seq3
 1998        1  |     1998        1  |     1998     1998  |    94.44    96.25     3.75  | -3 -3  26695.seq1    j99.seq3
 1999        2  |     1999        2  |     1998     1998  |    90.69    93.54     7.06  | -2 -2  26695.seq1    j99.seq3
 2000        3  |     2000        3  |     1998     1998  |    90.69    92.94     7.13  | -1 -1  26695.seq1    j99.seq3
    1     1998  |     2000        3  |     1998     1998  |    92.50    93.85     6.67  |  1 -1  26695.seq2    j99.seq2
    2     1999  |     1999        2  |     1998     1998  |    85.16    87.56     9.00  |  2 -2  26695.seq2    j99.seq2
    3     2000  |     1998        1  |     1998     1998  |    86.21    90.85     4.12  |  3 -3  26695.seq2    j99.seq2
 1998        1  |        3     2000  |     1998     1998  |    85.76    89.81     9.00  | -3  3  26695.seq2    j99.seq2
 1999        2  |        2     1999  |     1998     1998  |    85.76    88.46     6.90  | -2  2  26695.seq2    j99.seq2
 2000        3  |        1     1998  |     1998     1998  |    94.15    96.25     0.15  | -1  1  26695.seq2    j99.seq2

   .
   .
   .
```

Figure 10.3.7 Beginning of file `pro.coords`. This file summarizes every match found between a query sequence and a reference sequence. The first 7 and last 3 columns are as in Figure 10.3.5. Column 8 [`%SIM`] is the percent similarity according to the specified BLOSUM matrix. Column 9 [`%STP`] is the percentage of codons in the match that are stop codons. Columns 10 and 11 [`FRM`] indicate the reading frame of the match in the reference and query sequence, respectively, positive for forward strand, negative for reverse strand.

```
26695parts.seq j99parts.seq

============================================================
-- Alignments between 26695.seq1 and j99.seq3

-- BEGIN alignment [ +2 2 - 1999 | +2 2 - 1999 ]


2           VISDYSIIF*I*V*NTLNSRVFE*AFCSKNPR*RLKI*GC*AQRRVCQA
            VISDYSIIF*I* *N L SRVFE AFCSKNPR*RLKI*GC*AQRRVCQA
2           VISDYSIIF*I*A*NALKSRVFERAFCSKNPR*RLKI*GC*AQRRVCQA

149         LCIH****GFAWA*SQFHTLLSVKSPFPCSLIA*NPSLLNAAR*GRQFH
            LCIH****GF WA*SQFHTLLSVKSPFPCSLIA*NPSLLNA R* RQ H
149         LCIH****GFVWA*SQFHTLLSVKSPFPCSLIA*NPSLLNAVR*ERQSH

296         *FFLTRH*KGLKRKRVGCF*SFFLLTC*KRFF*FLHYRSQTHTTIQPQP
            *FFLTRH*KGLKRKR GCF*SFFLLTC KRFF*FLHYRSQTHTTIQPQP
296         *FFLTRH*KGLKRKRAGCF*SFFLLTCWKRFF*FLHYRSQTHTTIQPQP

443         PWLEFVSPF*PLRVWHKLNNSMRVLIASKPLLPALLPALSIACSILSVV
            PWLEFVSPF* LRVWHKLNNSMRVLIASKPLLPALLPALSIACSILSVV
443         PWLEFVSPF*SLRVWHKLNNSMRVLIASKPLLPALLPALSIACSILSVV

      ⋮
```

Figure 10.3.8 Beginning of an alignment created by the `show-aligns` program for PROmer output.

MUMmer1: ALIGNING TWO SINGLE SEQUENCES

The MUMmer1 script is provided for backward compatibility with release 1.0 of the MUMmer package. This script is designed to find the single longest chain of matches between two single sequences.

Necessary Resources

Hardware

Same as the MUMmer2 protocol except that more memory is required: ~25 bytes per base of both the query and reference sequences. Thus, to compare two 2-megabase genomes will require ~100 Mb of main memory.

Software

The MUMmer1 script is included in the MUMmer 2.12 package (see Support Protocol for download and installation)

Files

A multi-FASTA query file and a single-FASTA reference file (see *APPENDIX 1B* for information on FASTA). The files used in this example are complete genomic sequences from two strains of *Helicobacter pylori*—known as 26695 and J99. These sequences can be downloaded from TIGR's Comprehensive Microbial Resource at *http://www.tigr.org/CMR*, from the NCBI at *http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/micr.html*, or from the *Current Protocols in Bioinformatics* Web site at *http://www3.interscience.wiley.com/c_p/cpbi_sampledatafiles.htm*. These are the same files as in the example used for the Basic Protocol.

Run the MUMmer1 script

1. For this example, the user would type:

```
run-mummer1.csh hp26695.seq hpj99.seq mum1
```

The first two parameters are the reference and query sequence files. The third parameter is a tag name used to identify the output files. The user optionally may specify a fourth parameter `-r` *that will cause the reference sequence to be reverse complemented before any comparisons are made.*

View outputs and alignments

2. The script will produce four output files analogous to those created by the MUMmer2 script: `mum1.out`, `mum1.gaps`, `mum1.errorsgaps`, and `mum1.align`.

File `mum1.out` *contains the MUMs found by the* `mummer1` *program, one MUM per line. The values on each line are: reference start position, query start position, and length. The* `mummer1` *program differs from the* `mummer2` *program in that the matches it finds are unique in both the query and reference sequence. The minimum-length match output is determined by the* `MIN_UNIQUE_MATCH_LEN` *constant at the top of the file* `mummer1`*. The default value is 20. If the user changes this parameter, the program will need to be recompiled by typing* `make mummer1`.

File `mum1.gaps` *contains the MUMs from file* `mum1.out` *partitioned into two groups. The first group, with a header line saying* `Consistent matches`*, is the longest possible set of MUMs that occur in the same order in both sequences, known as a longest ascending subsequence. Length here means the number of bases covered in the reference sequence, not the number of separate MUMs. The second group of MUMs, indicated by a header line saying* `Other matches`*, is all MUMs not in the first group, sorted by position in the query sequence. The columns in this file have the same meaning as the columns in the* `.gaps` *file of the MUMmer2 script (see Fig. 10.3.3).*

Files `mum1.align` *and* `mum1.errorsgaps` *are analogous to the corresponding MUMmer2 files. The first shows alignments of the regions between consecutive MUMs from the* `mum1.gaps` *file (see Fig. 10.3.4); however, no alignment is computed if the region is too long. File* `mum1.errorsgaps` *recapitulates the information in* `mum1.gaps`*, but adds an additional column indicating the number of errors in each gap between consecutive MUMs.*

SUPPORT PROTOCOL

OBTAINING AND INSTALLING THE MUMmer PACKAGE

The MUMmer 2.12 package is available from the TIGR Web site at *http://www.tigr.org/software/mummer*. The package is free of charge to nonprofit researchers using it for noncommercial purposes. After filling out the online license agreement, such users will receive instructions via E-mail on how to download the system. For-profit institutions will find instructions for obtaining a license at the same Web site.

The system is contained in a compressed Unix tar file named `MUMmer2.12.tar.gz`.

Type:

```
gunzip MUMmer2.12.tar.gz
```

at a Unix command-line prompt to uncompress the file, and then type:

```
tar xf MUMmer2.12.tar
```

to create the MUMmer2.12 directory. To install the system type:

```
cd MUMmer2.12
```

to change to the base directory, and then type:

```
make install
```

to compile and build all the programs. The executable files and scripts will be placed in the base directory which can be added to the path.

Additional information is in the files `INSTALL` and `README` and in the `docs` subdirectory.

GUIDELINES FOR UNDERSTANDING RESULTS

It is important for the user of MUMmer to understand that MUMmer alignments are not exhaustive. By definition MUMs must be unique so that matches involving exact repeats in the reference sequence will not be found unless the `max-match` program is used. In most genomic comparisons, however, there are substantial regions of unique sequence that will be matched, after which repeats will be found by aligning the regions between MUMs. Even for repeats, if there is any sequence variation that makes the repeat unique, then MUMmer will find the match.

Repeats can also cause artifactual MUMs. Suppose, for example, that a query sequence contains a region of 30 consecutive A's and that a reference sequence contains multiple regions of more than 30 consecutive A's. It is likely that a few nucleotides immediately following the query poly(A) region will uniquely match the bases after one of the reference poly(A) regions and form a MUM. Thus, matches to exact repeats can become MUMs because of a few randomly matched bases around the repeat.

COMMENTARY

Background Information

Genome sequence alignment research has developed highly efficient algorithms for alignment of sequences, including BLAST (*UNITS 3.3 & 3.4*; Altschul et al., 1990) and FASTA (Pearson, 2000) systems. In 1999, as complete genome sequences increasingly became available, the authors of this unit introduced a method for efficient alignment of large-scale DNA sequences, on the order of millions of nucleotides (Delcher et al., 1999). This alignment system, which the authors called MUMmer, was capable of aligning complete bacterial genomes in <1 min on a standard desktop computer. In 2002 the authors released a new version of the system, MUMmer2, with greater functionality and more efficient implementations of core algorithms (Delcher et al., 2002).

The algorithmic design strategy of MUMmer is a three-step process. First a series of anchor matches are found. The anchor matches are then grouped together into clusters or runs that appear together in the genome. Finally, alignments are computed between matches in the runs and off the ends of the first and last match.

The core match-finding algorithms of MUMmer all take two input sequences, either DNA or proteins, and find *all* subsequences longer than a specified minimum length *k* that are identical between the two inputs. These matches are guaranteed to be maximal, in that they cannot be extended on either end without incurring a mismatch—or, stated differently, no match is completely contained in a larger match. Depending on the specific algorithm used, the matches have varying uniqueness properties. In the original MUMmer algorithm, matches are unique in both input sequences, i.e., the matching sequence occurs exactly once in each of the two input sequences. These matches are found by putting both input sequences into a data structure known as a suffix tree, and then identifying matches by traversing that tree. See Gusfield (1997) for a comprehensive explanation of suffix trees and Kurtz (1999) for details on memory-efficient implementations of them. In this case, the remarkable fact about suffix trees is that this entire matching algorithm can be accomplished in linear time and space, i.e., in time proportional to the lengths of the two sequences, the suffix tree can be built and all MUMs found.

For very long sequences, such as eukaryotic genomes hundreds of megabases long, it is impractical to store both sequences to be compared in computer memory as a single suffix tree. To facilitate comparisons between such sequences, the authors implemented a variation of the basic match algorithm that requires only one of the sequences (the reference sequence) to be stored in the suffix tree. The other sequence (the query) can then be read and matched against the suffix tree in a streaming fashion, outputting the matches as they are found (Chang and Lawler, 1994). This allows arbitrarily long and/or multiple query sequences to be matched against a single reference sequence. The only drawback to this method, however, is that matches are guaranteed to be unique only within the reference sequence. Uniqueness in the query cannot be achieved because matches are output as they are found, before the entire query sequence has finished being read, and the very same match could occur again later in the query. This semi-uniqueness can be desirable when comparing a set of queries representing fragments or a partial assembly to a reference genome. Because the fragments may overlap, it is desirable to allow matches that occur in multiple query elements.

Software design philosophy

MUMmer is specifically designed to be a suite of programs, each with a single function and simple input/output format, that can be used in various combinations. The different match-finding algorithms, described above, can be used interchangeably. The programs also lend themselves readily to being incorporated into other scripts. For example, a user can freely edit the output of the match-finding program to add or remove specific matches before further processing by the clustering and alignment programs.

In addition, it is important to note that MUMmer is distributed as source code that the user can freely modify. The system is written in standard C++. Users with programming expertise should readily be able to modify the programs to introduce variations in the algorithms or modify input/output formats.

Examples of using MUMmer

The MUMmer system has been used to make a number of significant discoveries about large-scale genome structure. Alignments of related bacterial species led to the discovery that chromosome-scale inversions are a com-

mon evolutionary phenomenon in these species, and that the inversions are nearly always symmetric about the origin of replication (Eisen et al., 2000). These inversions show up as X-shaped alignments in the dot plot of all the DNA sequences conserved between two species. The X alignments have been observed by running MUMmer to compare numerous pairs of related bacterial species.

MUMmer was used to construct alignments of the five chromosomes of the model plant *Arabidopsis thaliana*, which range in size from 17 to 29 million base pairs (Mbp), against one another. These alignments revealed the striking discovery that the entire genome appears to have duplicated recently, and over 60% of the genome as it exists today is part of large-scale duplications (Arabidopsis Genome Initiative, 2000). An earlier MUMmer-based analysis on the first two complete chromosomes had found a 5-Mbp duplication (Lin et al., 1999).

The protein-level matches of MUMmer were used to align human chromosomes to each other, searching for duplications similar to those that the authors had found in *Arabidopsis*. The authors' initial searches using DNA sequence alignments revealed no large-scale duplications in the human genome. For each human chromosome, all proteins were concatenated in order (regardless of strand) to create 24 mini-proteomes. MUMmer was then used to align each chromosome to the entire genome at the protein level. This was very successful, revealing hundreds of small-scale and many large-scale duplications, all of them apparently quite ancient. For example, one of the most striking findings was that over 70% of human chromosome 14 appears to be an ancient duplication of part of chromosome 2 (Venter et al., 2001).

Protein-level matches also were used to find alignments between two *Plasmodium* species, *falciparum* and *yoelii,* the parasites that cause malaria in humans and rodents, respectively. These matches were used to aid the assembly of *P. yoelii*, providing a map to guide finishing efforts (Delcher et al., 2002). The `max-match` program of MUMmer was used in the analysis of *E. coli* strain O157 (Perna et al., 2001) and MUMmer was used to find regions of conserved synteny between the Celera assembly of the mouse genome and the human genome (Mural et al., 2002).

Critical Parameters

The `-l` option of program `mummer2` is probably the most important parameter. For extremely closely related sequences, the exact value of this parameter is not very critical; for example, if the inputs are two successive assemblies of the same genome using different shotgun sequence data, then the identity will be so high that MUM lengths anywhere from 20 to 100 or more will give very similar results. As the MUM length is reduced, the system will find increasing numbers of "random" matches; these are sequences that match between the two inputs but that are not shared due to common ancestry. Thus a larger value of this parameter will generate less noise. Balancing this is the fact that for more distantly related input sequences, a large value will result in too few MUMs being identified. Because the system runs in less than 30 seconds when comparing typical bacterial genomes, it is advisable to run it several times and then view the results with a simple plotting tool such as GNUPLOT (see Fig. 10.3.9; *http://www.gnuplot.info*).

The clustering parameters used in the `mgaps` program are also important. These values determine how MUMs will be combined into groups where alignments will be computed. For highly similar sequences, large overall clusters can be specified with small gap sizes and diagonal differences. If one wishes every MUM to appear in a cluster, then the minimum cluster size can be set to 1. Since a cluster will be broken if an alignment cannot be found between its adjacent MUMs, the error-rate parameter for alignments must be set low enough to allow MUMs to be connected.

Troubleshooting

The `makefile` for the system automatically attempts to find the software needed to build and run MUMmer. The necessary programs are `/bin/sh`, `perl`, `sed`, `g++`, and `csh`. If any of these components cannot be found, an error will result. One reason a component may not be found is that it may be installed in a directory that is not on your path. Users unfamiliar with Unix should see *APPENDIX 1C* or consult their system administrator about locating the required component and setting the path. If the necessary component is not on one's system, it will need to be installed; consult the system administrator on how to do this. All of the above programs are included with most distributions of Linux, and are also freely available on the internet.

One should also be sure your system has enough memory for the sizes of the input files. If MUMmer works for small input files, but not larger files, this may be the problem. If the

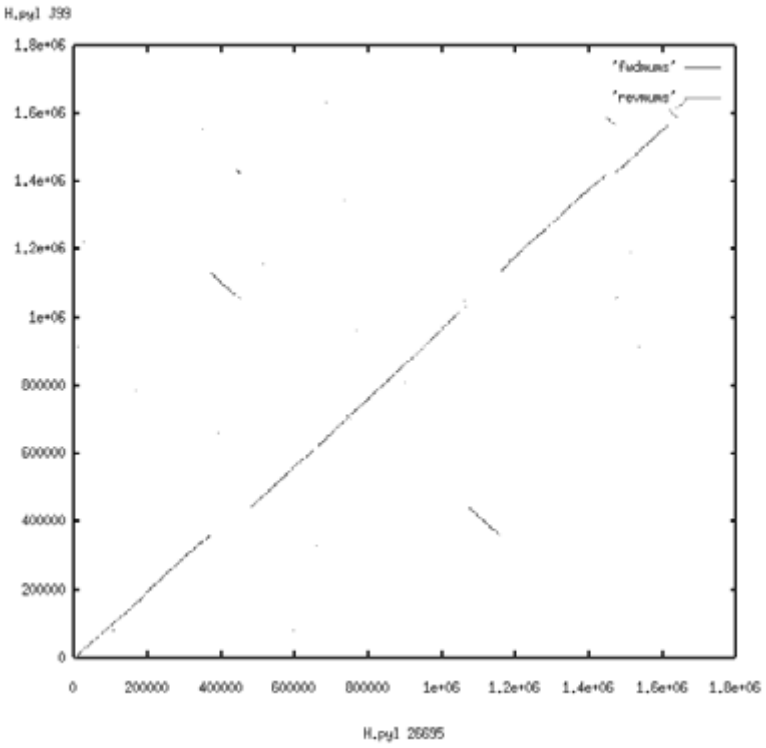

Figure 10.3.9 Dot plot of MUMs between *H. pylori* strains 26695 and J99. MUMs between both forward strands are shown in red and MUMs between the forward and reverse strands are shown in blue. Each line segment connects the start position of the MUM to the end position of the MUM. *This black and white facsimile of the figure is intended only as a placeholder; for full-color version of figure, go to http://www.currentprotocols.com/colorfigures.*

computer system is shared by multiple users, another user may be preventing you from accessing sufficient memory.

Suggestions for Further Analysis

To visualize MUMmer alignments, a simple dot plot can easily be constructed by plotting a line segment for each MUM on a two-dimensional axis. Alignments between the positive strands of each sequence can be plotted in one color with positive slope and positive-negative alignments in a different color with negative slope. An example of such a plot, made using a simple awk script and the free plotting program GNUPLOT, for the MUMs in file `j99vs26695.out` is shown in Figure 10.3.9. (The MUMmer header lines—all lines beginning with >—need to be changed to comment lines or deleted for compatibility with GNUPLOT. The script `mumdotplot.awk` does this in addition to producing the plot.) Aligned regions between two genomes usually show up as a series of colinear dots in these plots, with a slope of +1 or −1 depending on whether there has been an inversion in one of the genomes.

MUMmer is best suited for computing alignments between very long sequences with regions of high similarity. To find matches between short sequences and a large database, or to find matches between evolutionarily distant sequences, a tool such as BLAST (*UNITS 3.3 & 3.4*) is more appropriate.

PIPmaker is another system available for comparing lengthy DNA sequences (*UNIT 10.2*; Schwartz et al., 2000). It uses a hashing approach based on the BLAST algorithm, with improvements to handle large input sizes using only linear space (Chao et al., 1995). PIPmaker finds both approximate and exact alignments and also generates a very useful graphical display, showing which portions of the alignment match at different percent identities.

Acknowledgements

This work was supported in part by grants IIS-9902923 to SLS and IIS-9820497 to ALD from the National Science Foundation, and by

grant R01-LM06845 to SLS from the National Institutes of Health.

Literature Cited

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. Basic local alignment search tool. *J. Mol. Biol*. 215:403-410.

Arabidopsis Genome Initiative. 2000. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature* 408:796-815.

Chang, W.I. and Lawler, E.L. 1994. Sublinear expected time approximate string matching and biological applications. *Algorithmica* 12:327-344.

Chao, K.M., Zhang, J., Ostell, J., and Miller, W. 1995. A local alignment tool for very long DNA sequences. *Comput. Appl. Biosci*. 11:147-153.

Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O., and Salzberg, S.L. 1999. Alignment of whole genomes. *Nucleic Acids Res*. 27:2369-2376.

Delcher, A.L., Phillipy, A., Carlton, J., and Salzberg, S.L. 2002. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res*. 30:2478-2483.

Eisen, J.A., Heidelberg, J.F., White, O., and Salzberg, S.L. 2000. Evidence for symmetric chromosomal inversions around the replication origin in bacteria. *Genome Biol*. 1:research11.01-09.

Gusfield, D. 1997. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York.

Henikoff, J.G., Pietrokovski, S., McCallum, C.M., and Henikoff, S. 2000. Blocks-based methods for detecting protein homology. *Electrophoresis* 21:1700-1706.

Kurtz, S. 1999. Reducing the space requirement of suffix trees. *Software Practice and Experience* 29:1149-1171.

Lin, X., Kaul, S., Rounsley, S., Shea, T.P., Benito, M.I., Town, C.D., Fujii, C.Y., Mason, T., Bowman, C.L., Barnstead, M. et al. 1999. Sequence and analysis of chromosome 2 of the plant *Arabidopsis thaliana*. *Nature* 402:761-768.

Mural, R.J., Adams, M.D., Myers, E.W., Smith, H.O., Miklos, G.L.G., Wides, R., Halpern, A., Li, P.W., Sutton, G., Nadeau, J., et al. 2002. A comparison of whole-genome shotgun-derived mouse chromosome 16 and the human genome. *Science* 296:1661-1671.

Pearson, W.R. 2000. Flexible sequence similarity searching with the FASTA3 program package. *Methods Mol. Biol*. 132:185-219.

Perna, N.T., Plunkett, G., 3rd, Burland, V., Mau, B., Glasner, J.D., Rose, D.J., Mayhew, G.F., Evans, P.S., Gregor, J., Kirkpatrick, H.A. et al. 2001. Genome sequence of enterohaemorrhagic *Escherichia coli* O157:H7. *Nature* 409:529-533.

Schwartz, S., Zhang, Z., Frazer, K.A., Smit, A., Riemer, C., Bouck, J., Gibbs, R., Hardison, R., and Miller, W. 2000. PipMaker—a web server for aligning two genomic DNA sequences. *Genome Res*. 10:577-586.

Venter, J.C., Adams, M.D., Myers, E.W., Li, P.W., Mural, R.J., Sutton, G.G., Smith, H.O., Yandell, M., Evans, C.A., Holt, R.A. et al. 2001. The sequence of the human genome. *Science* 291:1304-1351.

Key References

Delcher et al., 1999. See above.

This describes the original MUMmer1 algorithm.

Delcher et al., 2002. See above.

This describes the enhancements in version 2 of MUMmer, including improved efficiency, more flexible clustering and alignment options, and the ability to handle files with multiple sequences.

Gusfield, 1997. See above.

This is a comprehensive treatment of suffix trees and sequence alignment algorithms for those interested in computer science details.

Internet Resources

http://www.tigr.org/software/mummer

The MUMmer homepage.

Contributed by Arthur L. Delcher
The Institute for Genomic Research
Rockville, Maryland
and Computer Science Department
Loyola College in Maryland
Baltimore, Maryland

Steven L. Salzberg and Adam M. Phillippy
The Institute for Genomic Research
Rockville, Maryland