Inverse Document Frequency Weighted Genomic Sequence Retrieval

Kevin C. O'Kane, Ph.D.
Professor Emeritus
Computer Science Department
University of Northern Iowa
Cedar Falls, IA 50614

kc.okane@gmail.com
https://threadsafebooks.com

Sept 27, 2018

Table of Contents

1 Installation and Use

The programs presented here are used to index and then search a database of nucleotides.

You will need to install the *gcc* compiler and the author's open-source version of the mumps language available at:

https://www.cs.uni.edu/~okane/

1.1 Sequence Databases

You will need to obtain a sequence database from one of the online sources. To use that database, it must be in FASTA format. The system is designed to be used on specific databases such as *gbpri*, *gbrod*, and so forth. The system can be used on the NT database but this would require considerable disk space to do the indexing.

The input to the indexing procedures, as well as queries, must be in Fasta format an example of which is:

```
>gi | AI504348.1  GI:4402199 | AI504348 | vl07c10.x1 Soares_mammary_gland_NbMMG Mus...
TTTTCACTTAAGTGCAAATGTCAGTGGTTTTATTGAAACCTCACCAACGGCCCTCAGGAA
GAAATGACCTAGAAAAGGCTATCTAGTACTCATTTAGTTGGACCATGGTGTGTGTGGGGG
ATGGGGGGGAGAGGGAGG
```

That is, a single title line (truncated in the above) beginning with a ">" sign containing a description of the sequence.

Following the title line are one or more lines containing the sequence. The sequence must be in upper case.

If you want to download an NCBI data set, such as the EST sequences, you need to convert these to Fasta format. The NCBI sequences are at:

ftp://ftp.ncbi.nlm.nih.gov/genbank/

They are in compressed files with names such as:

```
gbest1.seq.gz     42.1 MB     6/20/15, 4:54:00 PM
gbest10.seq.gz    44.3 MB     6/20/15, 4:54:00 PM
gbest100.seq.gz   67.7 MB     6/20/15, 4:54:00 PM
gbest101.seq.gz   60.9 MB     6/20/15, 4:54:00 PM
gbest102.seq.gz   52.0 MB     6/20/15, 4:54:00 PM
gbest103.seq.gz   53.0 MB     6/20/15, 4:54:00 PM
gbest104.seq.gz   37.3 MB     6/20/15, 4:54:00 PM
gbest105.seq.gz   62.9 MB     6/20/15, 4:54:00 PM
gbest106.seq.gz   53.5 MB     6/20/15, 4:54:00 PM
gbest107.seq.gz   45.2 MB     6/20/15, 4:54:00 PM
gbest108.seq.gz   17.8 MB     6/20/15, 4:54:00 PM
gbest109.seq.gz   15.3 MB     6/20/15, 4:54:00 PM
gbest11.seq.gz    44.1 MB     6/20/15, 4:54:00 PM
gbest110.seq.gz   15.1 MB     6/20/15, 4:54:00 PM
gbest111.seq.gz   16.9 MB     6/20/15, 4:54:00 PM
```

```
gbest112.seq.gz   15.1 MB     6/20/15, 4:54:00 PM
```

At this time, there are about 485 files in the EST collection.

Note that the order in which the file names are listed is alphabetic, not numeric.

Each file is compressed (the *.gz* ending). After you download the files you need to decompress them. Most Linux systems have the software to do this already installed.

After the files have been decompressed, they will have the same names but without the *.gz* endings.

The contents of a typical file look something like this:

```
LOCUS       AA000005                344 bp    mRNA    linear   EST 18-JUL-1996
DEFINITION  mg27g05.r1 Soares mouse embryo NbME13.5 14.5 Mus musculus cDNA
            clone IMAGE:425048 5' similar to gb:D14531 60S RIBOSOMAL PROTEIN L9
            (HUMAN), mRNA sequence.
ACCESSION   AA000005
VERSION     AA000005.1  GI:1435870
DBLINK      BioSample: LIBEST_000437
KEYWORDS    EST.
SOURCE      Mus musculus (house mouse)
  ORGANISM  Mus musculus
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
            Sciurognathi; Muroidea; Muridae; Murinae; Mus; Mus.
REFERENCE   1  (bases 1 to 344)
  AUTHORS   Marra,M., Hillier,L., Allen,M., Bowles,M., Dietrich,N., Dubuque,T.,
            Geisel,S., Kucaba,T., Lacy,M., Le,M., Martin,J., Morris,M.,
            Schellenberg,K., Steptoe,M., Tan,F., Underwood,K., Moore,B.,
            Theising,B., Wylie,T., Lennon,G., Soares,B., Wilson,R. and
            Waterston,R.
  TITLE     The WashU-HHMI Mouse EST Project
  JOURNAL   Unpublished (1996)
COMMENT     Contact: Marra M/Mouse EST Project
            WashU-HHMI Mouse EST Project
            Washington University School of MedicineP
            4444 Forest Park Parkway, Box 8501, St. Louis, MO 63108
            Tel: 314 286 1800
            Fax: 314 286 1810
            Email: mouseest@watson.wustl.edu
            This clone is available royalty-free through LLNL ; contact the
            IMAGE Consortium (info@image.llnl.gov) for further information.
            MGI:259600
            Trace considered overall poor quality
            Seq primer: ETPrimer
            Sequence considered overall poor quality.
FEATURES             Location/Qualifiers
     source          1..344
                     /organism="Mus musculus"
                     /mol_type="mRNA"
                     /strain="C57BL/6J"
                     /db_xref="taxon:10090"
                     /clone="IMAGE:425048"
                     /sex="unknown"
                     /tissue_type="embryo"
```

```
                     /dev_stage="13.5-14.5dpc total fetus"
                     /lab_host="DH10B"
                     /clone_lib="LIBEST_000437 Soares mouse embryo NbME13.5
                     14.5"
                     /note="Vector: pT7T3D-PacI; Site_1: Not I; Site_2: Eco RI;
                     1st strand cDNA was primed with a Not I - oligo(dT) primer
                     [5'
                     TGTTACCAATCTGAAGTGGGAGCGGCCGCGGAAATTTTTTTTTTTTTTTTTTTTTT
                     T 3'], on equal amounts of mRNA from 2 13.5dpc and 2
                     14.5dpc embryos [total RNA provided by  Minoru Ko, Wayne
                     State Univ., from 2 ]; double-stranded cDNA was ligated to
                     Eco RI adaptors (Pharmacia), digested with Not I and
                     cloned into the Not I and Eco RI sites of the modified
                     pT7T3 vector. Library went through one round of
                     normalization, and was constructed by Bento Soares and
                     M.Fatima Bonaldo."
ORIGIN
        1 gaacatgatc aagggtgtca cgctgggctt ccgatacaag atgcggtctg tgtacgctca
       61 cttccccatc aacgtcgtca tccaggagaa tggctctttg gttgaaatcc gaaatttctt
      121 gggtgaagag tacatccgca gagttcggat gaggacaggt gtggcttgtt ctgtctctca
      181 agcccagaag gatgagttaa tccttgaagg aaatgacgtt gaactggttt cagattcagc
      241 tgccctgatt cagcaagcca caacagttat aatcaaggat atcaggaagt gtttgaacgg
      301 catctatgtg tctgagaagg taactgtgca gcaggctgac gaat
//
```

The sequence itself begins after the keyword *ORIGIN*.

In the distribution is a *Bash* script named *Make-Fasta.script*. This script will extract the sequence information from all NCBI sequence files with the *.seq* ending (in the current directory from which the Bash script file is executed) and write the results into a file named *fasta.sequences* also in the current directory. It will also convert the sequence letters to upper case which is required for the indexing process. This script will attempt to use all CPUs available in order to work faster. You can change the number of CPUs to be used by editing the script.

1.2 Indexing the Data Set

After you have created or downloaded a Fasta format database of sequences, rename the file (or create a symbolic link) to *nt*. Move this file into the sub-directory *indexSequences* in the distribution.

Next, index the database using the bash script *Generate-Index.script.* This script will run for a considerable amount of time and require a large amount of disk space and memory. It will run faster if you can run this on an SSD drive or have the input file (indexSequences/*nt*) on a separate disk drive than other files in the working set.

The procedure will run faster if you have more memory available. The more the better. There are settings, discussed below, in the script file *Generate-Index.script* for memory and CPU usage. You will need to set these to values that are appropriate for your system. The default values assume a machine with at least 4 CPU cores and 8 gigabytes of memory.

1.3 Retrieval

The program for sequence retrieval is *search.mps* which reads the query sequence from *stdin*.

```
search.mps < input_query.txt
```

where *input_query.txt* is an ASCII file containing the query sequence in Fasta format.

The results will contain the IDF selected matching sequence(s) and, optionally, Fasta results (see below) for the same query.

There is also a script file named *test.script* which will generate a random query, randomly modify it, and submit it to the search procedure. The test query title line will be the title from the randomly selected sequence for reference purposes.

1.4 Examples

The following examples were done on a database consisting of the first 50 files from the NCBI *gbpri* database (about 943,366 sequences in an 8.0 GB file in Fasta format). The test query was also submitted to Fasta which scanned the original input database. The IDF Score is the accumulated inverse document frequency score of the listed sequence when compared to the the query. The Fasta output is described in the Fasta documentation.

While the Fasta program is not part of the IDF system code, Fasta code is included for comparison purposes. The original Fasta paper is at:

W.R. Pearson & D.J. Lipman PNAS (1988) 85:2444-2448

The following example was run on an AMD FX-6100 Six-Core Processor with 8 GB of memory. Results (long titles truncated):

```
test.script

Sequence Searcher Sat Sep 22 12:09:08 2018
Database files:
/home/okane/Desktop/genomics/IDF-New/inputSequences/nt,old
/media/okane/WM1/work/words.merged,old

Query:
>gi | AC144445.3 | AC144445 | Homo sapiens fosmid clone XXFOS-83307H10 1019941581
CTACTGTCTAGTTGTTATACGAAGATTCCCTATTTTCTACCTTTGGTCTCAAAGCGATTGAAATCTCCACATGGAAACTCCA
CAAAAAGAGTGTTTCAAATCTGCTCTTTCTGAAGGAAGGTTCATCTCTGTGAGTTGAATACACACACCACAAATAAGTTACT
GAGAATTCTTCTGTGTAACATTATATGAGGAAATCCCGTTTCCAACGAAGGCCTCAAAGAGGTCCAAATATCCACTTGCAGG
ACTTTACAAAGACAGTGTCTCCAAACTCCTCCATCAAAAGAAAGGTTATACTCTGTGAATTGAACGCACACATCACAAAGTA
GTTTCTGAGAATGATTCTTGTCTAGTTTTTATACGAAGATATTTCCTTTTCTACATTTGGCCTAAAAGCGCTTGAAATCTCC
ACCTGCAAATATCACAAAAAGAGGGTTTCACATCTGCTCTGTCTAAAGGACAGTTCACCTCTGTGAGTATGAATAGAGGCAA
CACAAAGAACTTACTCAGTATTCTTCTTTCTAGCGTTCTATGAAGAAATCCCGTTTCCAACGAAGGCCCCACAGAGGTCCAA
ATATCCTGCTGTGCAGACTTTACAGACAGCAGTGTTTCCAAACTGCTCCATCAAAAGAAAGGTTAAACTCCTTGAGTTGAAC
ACACACATCACAAAGTAGTTTCTGTGAATGATTCTGTCTAGTTGTTATACGAAGATGTTTCCTTTTCTACCTTCTGGTCTCA
AAGCGATTGAGAATCTCCACATGGAAACTCCACAAAAAGAGTGTTTCAAATCTGCTCTTTCTGAAGGAAGGTTCATCTCTGT
GAGTTGAATACACACACCACAAATAAGTTAGTGAGAATTCTTGTGTGTAACATTATATGAGGAAATCCCGTTTCCAACGAAG
GCCTCAAAGAGGTCGCAAATATCCACTTGCAGACTTTACAAAGACAGTGTCTCCAAACTCCTCCATCAAAAGAAAGGTTATA
CTCTGTGAATTGAACGCACACATCACAAAGTAGTTTCGTGAGAATGATTCTGTCTAGTTTTTATACGAAGATATTTCCTTTT
CTACATTTGGCCTAAAAGCGCTTGAAATCTCCACCTGCAAATATCACAAAAAGAGGGTTTCACATCTGCTCTGTCTAAAGGA
CAGTTCACCTCTGTGAGTTGAATAGAGGCAACACAAAGAACTTACTCAGTATTCTTCTTTCT
```

```
IDF Results:

IDF: 10951 >gi | AC144445.3 | AC144445 | Homo sapiens fosmid clone XXFOS-83307H10
IDF:  7998 >gi | AB761759.1 | AB761759 | Homo sapiens DNA, flanking sequence of p
IDF:  7124 >gi | AC104806.3 | AC104806 | Homo sapiens BAC clone RP11-584P21 from
IDF:  7102 >gi | AC245148.2 | AC245148 | Homo sapiens BAC clone CH17-77N3 from ch
IDF:  7036 >gi | AC026131.4 | AC026131 | Homo sapiens chromosome 10, clone XX-XX,
IDF:  6784 >gi | AC019063.4 | AC019063 | Homo sapiens BAC clone RP11-144H20 from
IDF:  5955 >gi | BX322613.6 | BX322613 | Human DNA sequence from clone RP11-745D9
IDF:  5821 >gi | AC146133.3 | AC146133 | Pan troglodytes BAC clone RP43-38C10 fro
IDF:  5671 >gi | AC144535.4 | AC144535 | Homo sapiens 12 BAC RP11-191K23 (Roswell
IDF:  5667 >gi | BX537339.3 | BX537339 | Human DNA sequence from clone RP13-511L2
IDF:  5639 >gi | M93286.1 | M93286 | Human alpha-satellite repetitive DNA.
IDF:  5499 >gi | FP325312.10 | FP325312 | Human DNA sequence from clone CH17-297E
IDF:  5489 >gi | AB761968.1 | AB761968 | Homo sapiens DNA, flanking sequence of p
IDF:  5407 >gi | AC192988.3 | AC192988 | Pan troglodytes BAC clone CH251-386P18 f
IDF:  5351 >gi | AC244258.2 | AC244258 | Homo sapiens BAC clone CH17-410A11 from
IDF:  5329 >gi | M93288.1 | M93288 | Human alpha-satellite repetitive DNA.
IDF:  5315 >gi | AC118650.5 | AC118650 | Homo sapiens chromosome 8, clone CTD-232
IDF:  5027 >gi | BX001034.6 | BX001034 | Human DNA sequence from clone RP11-21H4
IDF:  5017 >gi | BX000449.4 | BX000449 | Human DNA sequence from clone RP11-287N1
IDF:  4865 >gi | AC239904.1 | AC239904 | Homo sapiens FOSMID clone WI2-2971I15 fr

IDF search time: 4

Fasta results:
Fasta search time 128

The best scores are:                                      opt bits E(969128)
gi | AC142529.3 | AC142529 | Homo sapiens fosm (33836) [f] 4276 783.6       0
gi | AC270241.1 | AC270241 | Homo sapiens BAC  (199005) [r] 4265 781.6       0
gi | AC123578.4 | AC123578 | Homo sapiens BAC  (3529) [r] 4123 756.2 2.3e-215
gi | AC019063.4 | AC019063 | Homo sapiens BAC  (187282) [f] 4123 756.0 2.6e-215
gi | AC144445.3 | AC144445 | Homo sapiens fosm (38114) [f] 3607 662.9 2.8e-187
gi | AC133920.2 | AC133920 | Homo sapiens chro (197357) [f] 3539 650.5 1.5e-183
gi | AL590544.7 | AL590544 | Human DNA sequenc (86052) [f] 3381 622.1 5.5e-175
gi | Z12006.1 | Z12006 | Homo sapiens alpha sa (2889) [r] 3353 617.3 1.5e-173
gi | AC104806.3 | AC104806 | Homo sapiens BAC  (147777) [f] 3329 612.6 3.8e-172
gi | AC118650.5 | AC118650 | Homo sapiens chro (105410) [f] 3315 610.1 2.1e-171
```

2 Algorithm and Files

The *Bash* script file *Generate-Index.script* contains the code to index an ASCII database of nucleotide sequences in Fasta format (discussed above). The following describes how the procedure works and the intermediate files used.

2.1 Compilation

At the beginning of *Generate-Index.script* the programs to be used are compiled. These are all written in C and use standard C libraries. The search procedure is written in an open-source version of the Mumps language available at:

https://www.cs.uni.edu/~okane/

The installation file contains a detailed explanation of how to install the Mumps interpreter. The interpreter itself is written in C and C++.

2.2 Disk Designations

It is desirable, but not required, that there be at least two disks available. The designation of the disks and the location s on them for the files to be stored or read from are given in *Generate-Index.script* in the lines:

```
DISK=/home/okane/Desktop/genomics/IDF-New
```

The values shown are for the author's machine.

DISK should point to the location of the IDF code distribution which was created when you decompressed the distribution file.

On that drive there is a directory named *work*. Files will be created in the sub-directory *work*. When the procedure is finished, there will be a files in *work.* The one named *words.merged* which is part of the results that will be used during searching. The location of this file will be encoded into the search routine and, therefore, should not be moved.

The input sequence database in Fasta format should by in the directory *inputSequences*. The sequence file should be named *nt*.

2.3 File Locations

The next section designates the locations and names of files that will be used in the indexing and search functions. These appear as:

```
# files

INPUT_FILE=$DISK/inputSequences/nt
MERGED_WORDS=$DISK/work/words.merged
WORDS_COUNT=$DISK/work/words.count
WORDS_PRUNED=$DISK/work/words.pruned
WORDS_PRUNED_SORTED=$DISK/work/words.pruned.sorted
WORDS_OFFSETS=$DISK/work/words.offsets
WORDS_IDF=$DISK/work/words.idf
WORK=$DISK/work
```

The files will be discussed in the following sections.

2.4 Parameters

There are some parameters that need to be set based on the size of your memory and the number of CPU cores available. These are:

```
# parameters
```

```
# For MAX_SEQUENCE_FILES: 1 means 2 (file 0 and file 1)
# MAX_SEQUENCE_FILES is the number of files that will be
# produced by cutnucs.

# LINES_PER_FILE is the number of lines in each file.
# A larger value for LINES_PER_FILE will require more memory
# for each instance of WordBuild1. You should adjust this number
# along with SETS to best use the memory your machine has
# available.

MAX_SEQUENCE_FILES=99
LINES_PER_FILE=10000000
SETS=4
IDF=65
```

The above work on the authors machine which is a 6 core AMD FX-6300 with 8 GB of memory.

MAX_SEQUENCE_FILES tells the system how many intermediate files to create from the input database. Counting starts at zero so 99 means a maximum of 100 files.

Initially the system breaks down the database into a number of smaller files which are processed in parallel. Ideally, processing will be quickest with the least number of files. This is discussed below.

However, you may want to test the system with a sub-set of the input database. In that case, setting the maximum number of files to 3 or 7 or 11 (meaning 4, 8 and 12) might be desireable. The testing will proceed much quicker with fewer files (although the results won't be as accurate).

SETS determines how many parallel tasks will initially process the input files. At present, this value may be either two or four. If you specify 2, only two processes will be initiated. If you specify four, four processes will be initialized. These are the only values available at present.

LINES_PER_FILE determines the maximum number of lines in each intermediate file. However, no sequence will be truncated as a result of this setting. Each input sequence will appear in its entirety even if this means a larger number of lines.

Next is the issue of available memory. The larger the files, the more memory will be required. Each process performs it's indexing in memory. The amount of memory a process uses is determined by how large the input file is and how many words are in the file *stop.list*.

The file *stop.list* contains words whose value is minimal to indexing such as *AAAAAAAAAAA* (each word is 11 characters long). The *stop.list* file is contructed by calculating the IDF values of all words then placing those whose IDF value is below a threshold into the *stop.list*. Provided with the distribution are some *stop.list* files for several databases.

If you have a comprehensive stop list for a given input database, the indexing process will ignore words in the input that are in the stop list. This will result in much less memory usage.

For example, the *stop.list* for the *gbpri* database with an IDF threshold of 75 contains 2,252,453 words.

Bottom line: you will need to adjust the number of processes and the number of lines per file to your available memory and stop list. You can do this by setting a low number of files to be created, for example 4 (3 in the code) and picking a maximum number of lines. The run the system and watch memory utilization by using the builtin Linux *System Monitor* graphical application. If all available memory is used, readjust the maximum number of lines per file or set the number of processes to two (SETS).

The *IDF* setting is the minimum threshold for the IDF value of a word to be considered for indexing (calculated near the end of the *Generate-Index.script* file). Words with lower values will not be used for indexing although they will be present in the file system unless they are in the stop list as *stop.list* words are removed from indexing early in the process.

2.5 Dividing the Input Database into Segments

Next the system will divide the input database into smaller segments that will be processed in parallel:

```
cutnucs $LINES_PER_FILE $CUT_OUTPUT $MAX_SEQUENCE_FILES $SETS < $INPUT_FILE

echo "End time `date`"

if [ $? -ne 0 ]; then
        echo "cutnucs did not return 0 - probable error"
        exit
        fi

echo
echo "Total characters extracted: `wc -c $CUT_OUTPUT/*.input`"

wc -c $CUT_OUTPUT/*.input | tail -1 > $WORK/TotChars
```

The program *cutnucs* divides the input database (*$INPUT_FILE*) into not more than *$MAX_SEQUENCE_FILES* files of approximately *$LINES_PER_FILE* and places the result in the directory *$CUT_OUTPUT*.

The file names will contain the letters *A* and *B* if *SETS* is 2, and the letters *A, B, C,* and *D* if *SETS* is 4. These letters are used to select files for the different processing silos. The file names will be of the format: *code.number.input* where *code* will be A,B,C or D, *number* will be a sequential numeric value followed by the word *input.*

After the files have been created, the code beginning with *wc* will calculate the total number of characters in all the input files and store this number in the file *TotChars* located in the directory *$WORK.*

In the 13 characters on the first line of each sequence (title line) of each DNA sequence in the output files from *cutnucs*, is a number which gives the offset of where the sequence originally appeared in the input data base.

The program *cutnucs* writes for each file a corresponding *DocCount.** file containing the number of sequences in the correspondingly numbered output file. These are summed and the result stored in *$WORK/DocCount* by the *Bash* code:

```
count=0

for i in $WORK/DocCount.*; do
        x=`cat $i`
        let "count = count + x"
        done

echo "$count" > $WORK/DocCount
echo "There were $count sequences processed"
```

2.6 Initial Word Extraction

Depending on the value of *SETS* above, the system will now initiate two or four independent shells to process the intermediate files created in the previous step.

```
fset=$CUT_OUTPUT/A
echo $fset
index.script $fset &

fset=$CUT_OUTPUT/B
echo $fset
nice -n 2 index.script $fset &

if [ $SETS -eq 4 ]; then
        fset=$CUT_OUTPUT/C
        echo $fset
        nice -n 4 index.script $fset &

        fset=$CUT_OUTPUT/D
        echo $fset
        nice -n 6 index.script $fset &
        fi

wait
```

The *Bash* variable *fset* successively contains the prefix of the file location information of one of the two or four sets of output files from the previous step. It is passed at a parameter to instances of *index.script* which are run as independent processes (the ampersand at the end of the line). These are initiated at slightly different scheduling priorities (*nice*) as experiments indicates this can make a small improvement in performance. The following is the file *index.script*:

```
#!/bin/bash

for file in $1.*; do
        echo "**** index -> $file"
        WordBuild1 $file &
        wait
```

```
rm $file
done
```

The file consists of a *bash* loop that will initiate and instance of *WordBuild1* as an independent process with a file name as the parameter. The file names will be of the form:

code.number.input

where *code* will be one of the letters A,B,C, or D and *number* will be an integer. There will be either two or four instances of *index.script* running at the same time. One of these will process all the files with code A, one all the files with code B and so forth. These will run concurrently.

The *Generate-Index.script* waits until all processes have finished before proceeding.

2.7 WordBuild1

The program *WordBuild1* reads each sequence and breaks the DNA codes into 11 character overlapping n-grams. From each n-gram, a number is calculated which is the index into an array. The element of the array addressed by an n-gram contains a pointer to a list of structures which contain the offset of where the the sequence containing this word occurred in the original input data base (see *cutnucs* above - it encodes this information in the first line of each sequence).

Consequently, as more sequences are read, the size of the list structures containing offset information grows dynamically.

When *WordBuild1* is finished with an input file, it writes out the information in the data structure to a file. The files are named *pid.words* where *pid* is the process id of the current instance of *WordBuild1*. The files are written to the sub-directory *work* in the directory from which *WordBuild1* was invoked (the distribution directory which contains an empty *work* sub-directory).

The format of each file is a set of newline terminated ASCII text where each line is of the format:

```
word offset
```

where *word* is the 11 character DNA n-gram word and *offset* is the offset in the original input database where the n-gram appeared. There will normally be many lines with duplicate *word* vales but each will have a unique *offset*. The output files will be sorted by n-gram in ascending order.

The number of files created will equal the number of input files.

2.8 Merge the Results

The next step involves merging the files from the previous step:

```
ls $WORK/*.words > $WORK/list
merge < $WORK/list > $MERGED_WORDS
```

First, a list of the files is created in *$WORK* by the name of *list*. Then the program *merge* is invoked and the contents of the list are passed to it as *stdin* input. The output of the merge will be in *$MERGED_WORDS*.

The merged file will have the same format as the input files (word offset). The file will be sorted in ascending n-gram order.

Each line in *$MERGED_WORDS* contains an n-gram and the offset into the original input file of the first line of a sequence that contains the n-gram. The file is ordered by n-grams. Thus, if a given n-gram appears in 100 sequences, there will be 100 entries in *$MERGED_WORDS* and they will be adjacent to one another in *$MERGED_WORDS.*

The words (or n-grams) in *$MERGED_WORDS* will be a subset of the total number of n-grams possible due to the *stop.list*. It the *stop.list* has been prepared such that it contains those words whose IDF score would be below the select threshold, then the words in *$MERGED_WORDS* will only consist of words with IDF scores greater than the threshold.

To build a stop list requires a separate execution of the system up to and including the IDF step below. The file *$WORD/words.idfRejects* will contain those words that fell below the threshold and this can be, after removing the first column (the IDF score), copied to *stop.list.*

Different databases have different word frequency characteristics so one stop list does not fit all databases as well as a tailored stop lists.

Using a minimal or no stop list results in a very large *$MERGED_WORDS* file and is very time consuming.

If a stop list not specific to the specific IDF threshold or the database is used, an additional *join* stem could be employed to prune the *$MERGED_WORDS* file once the IDF values have been computed. This is time consuming. A specific stop list (several are in the distro) is better.

However, the *$MERGED_WORDS* file can be pruned of words whose IDF score did not exceed the Threshold with a command similar to the following (note the line continuation backslash character):

```
join -1 1 -2 2 ../../work/words.merged words.idf | \
cut -d ' ' -f 1,2 > new.words.merged
```

This command will take a very long time to complete. Go shopping or cut the grass.

2.9 Count the Number of Instances of each N-gram

Next a count will be calculated for each n-gram:

```
cutWords < $MERGED_WORDS | uniq -c > $WORDS_COUNT
```

The program *cutWords* reads the merged words file from *stdin* and writes to *stdout* each n-gram. In most cases, a given n-gram will appear several times. All occurrences, because the input file was sorted by n-grams, will be immediately successive to one another.

The output will be passed to the standard Linux program *uniq* which will write, for each n-gram, a line consisting of the number of times the n-gram occurred and the n-gram itself. The output is placed in the file *$WORDS_COUNT*. The format of each line of *$WORDS_COUNT* will be *count word*.

2.10 Calculate IDF Values

Next, the IDF values for each n-gram are calculated:

```
idf $IDF $WORK < $WORDS_COUNT > $WORDS_IDF
```

The program *idf* is passed the address of the working directory (*$WORK*) where it will write intermediate output. It is also passed *$IDF* - the minimum IDF threshold It reads as input the file *$WORDS_COUNT* containing the number of times each n-gram occurred and it writes as output the file *$WORDS_IDF*. The program *idf* reads *$WORK/DocCount* (see above) which contains the total number of sequences being processes.

The output is an ASCII text file each of whose lines is of the form:

```
idf word
```

where *idf* is the idf value for the corresponding *word* (n-gram). Only n-grams whose IDF value is greater than or equal to the threshold are written to this file. N-grams whose IDF values are below this threshold are written to a file with the name format in *$WORD/words.idfRejects*.

If you are building a new stop list, you should place an *exit* in *Generate-Index.script* at this point. The file *$WORK/words.idfRejects* will contain your new stop list but you will need to remove the IDF score at the beginning of each line. The format for *stop.list* is one n-gram per line only.

2.11 Create a Table of Offsets Linking Words to Sequences

The next step is to create a table, *$WORD_OFFSETS*, that links each remaining n-gram (after IDF calculation) to the place in *$MERGED_WORDS* where the entries for an n-gram begin. The format of a line in *$WORDS_OFFSETS* is:

```
IDF WORD OFFSET
```

Where *IDF* is the *IDF* score of *WORD* and *OFFSET* is the address in *$MERGED_WORDS* where the first entry for this *WORD* appears. A line in *$MERGED_WORDS* contains a word followed by an offset. The offset points to the first line of a sequence in the original input file that contained the word. Subsequent lines in *$WORDS_MERGED* with the same word point additional sequences that contain the word.

Calculation of *$WORDS_OFFSETS* is done in parallel. The file words in *$WORDS_IDF* is split into four smaller files of equal size:

```
split -n l/4 $WORDS_IDF
```

producing files *xaa, xab, xac,* and *xad.*

The first entry of each of these files is passed to *offsetJump* which calculates *by means of a binary search* the offset in *$MERGED_WORDS* where the word in the entry begins. This value is passed to the next step as a starting point rather than have the program scan through the *$MERGED_WORD* for the first instance of the first entry which it is processing.

The program *offsets* takes as input the file *$MERGED_WORDS*, one of the files *xaa, xab, xac* or *xad* and the offset calculated by *offJump* where the first entry in *xaa, xab, xac*, or *xad* begins. The program *offsets* writes out a file named *$WORDS_OFFSETS.1, $WORDS_OFFSETS.2, $WORDS_OFFSETS.3,* or *$WORDS_OFFSETS.4.* The format of each of these is:

```
IDF WORD OFFSET
```

When these files have been built they are concatenated together and sorted by n-gram into *$WORDS_OFFSETS*.

2.12 Loading the Mumps Global Arrays

Searching is done by a program written in a Mumps language program (*search.mps*) based on contents of the Mumps global array database. Global arrays are disk-based, persistent data structures that resemble arrays or trees. They are accessed in the language in a manner similar to ordinary arrays. A quick tutorial is included in the Mumps distro referenced above. Global arrays are easily spotted in a program as they all begin with the circumflex character(^). The Mumps global arrays are stored in files named *key.dat* and *data.dat.*

The Mumps program (*load.mps*) receives as input the names of the initial input file (*$INPUT_FILE*) and the name of the *$MERGED_WORDS* file. These it stores in two global array references to be used by the retrieval program.

It then reads as input (*stdin*) *$WORDS_OFFSETS* as described above.

For each line it creates two global array references.

The first is an array named *idf* which has one index value, the value of the n-gram from from the line. At this element is stored the IDF value for the n-gram.

The second array element is named *ref* which likewise is indexed by the n-gram from the line. At this array element is stored the value of the offset from the line. The offset points to the first line of a sequence in the original input file.

2.13 Searching the Database

Searching is conducted by the Mumps program *search.mps.*

Search.mps extracts the values of the file names for the merged words file and the original input file and opens these files for input.

Next it reads and echoes the query. A copy of the query is written to an external flat ASCII file named *seq* for possible use by the program *fasta36*. The query should be in Fasta format.

The query sequence is broken down into 11 character overlapping n-grams. Each n-gram is tested to see if it exists in the ^*idf* global array. If it does, all those instances of the n-gram in the merged words file are read and a temporary global array, indexed by the offset in the original input file of the sequence containing the n-gram is created if it does not already exist. The value of the IDF score for the n-gram is added to each temporary array element.

The reader will note that the offset which is the index of the temporary global array element is unique to each original sequence. This, if subsequent n-grams are associated with this sequence, the sequence's element in the temporary array will be correctly updated (i.e., the IDF values will be added).

When all the 11 character n-grams from the query are processed, the contents of the temporary array are written out to a temporary flat ASCII file. Each line will contain the sum of the IDF scores followed by the sequence offset. This file is then sorted in reverse order (highest first) and read back in.

For each line read in, the original sequence title is fetched from the original input database and printed along with the total IDF score. A copy of each sequence is appended to a file named *tmp* (the previous file named *tmp* is no longer in use). This file can be submitted to Fasta36 for evaluation.

At the end of the program there is the option to have Fasta36 evaluate the sequences retrieved as a result of the IDF procedure or to process the query against the full database. Options on the Fasta36 control line can be used to enable a Smith-Watermann display of the matching sections.

3 Distribution Files

Included with the distribution are the following stop list files:

1. *gbpri.stop.list.75* A stop list suitable for an IDF threshold of 75 for the *gbpri* database.