

The ‘wcd ’ Manual

Version \$Id: wcd.texi,v 0.4.5.1 2008/03/20 13:22:58 scott Exp scott \$

Scott Hazelhurst

Copyright © 2007, University of the Witwatersrand, Johannesburg

Table of Contents

1	Description	1
1.1	Clustering	1
1.2	Distance functions	2
1.2.1	The D2 distance function	2
1.2.1.1	General formulation	2
1.2.1.2	Windows	3
1.2.1.3	References	3
1.2.2	Edit distance	3
2	Installing wcd	5
2.1	Getting new versions of <code>wcd</code>	5
2.2	Changes from versions 0.3.x	5
2.3	Installing the distribution	5
2.4	Compiling	5
2.4.1	Method 1: using <code>configure</code>	5
2.4.1.1	MPI Version	6
2.4.1.2	Pthreads	6
2.4.2	Method 2: Manually using <code>make</code>	6
2.4.2.1	Trouble shooting	6
2.4.3	EMBOSS	8
2.4.4	Auxiliary programs	8
2.5	Documentation	8
3	Running wcd	9
3.1	Summary	9
3.2	Examples — A Quick Introduction to <code>wcd</code>	10
3.2.1	Basic Clustering	10
3.2.2	More advanced clustering	11
3.2.2.1	Using a clustering file	11
3.2.2.2	Using a constraint file	11
3.2.2.3	An example of reclustering and constraint files.	12
3.2.3	Merging and adding	13
3.3	Help	14
3.4	Identifying a sequence	14
3.5	Comparing two sequences	14
3.6	Doing clustering	15
3.6.1	Arguments	15
3.6.2	Options	16
3.6.3	Clustering based upon suffix arrays	18
3.7	Clustering a range	18
3.8	Refining clusters	19
3.8.1	Merging clusters	19

3.8.2	Adding sequences	19
3.8.3	Reclustering	19
3.9	Format of input files	20
3.10	Output	21
3.11	Auxiliary programs	22
3.12	Running wcd in parallel	23
3.12.1	Shared Memory Parallelisation	24
3.12.2	MPI Parallelisation	24
4	wcd inside stackPACK	25
5	Technical manual	26
5.1	Program structure	26
5.2	Data structures	26
5.3	The algorithms	28
5.3.1	Heuristic	28
5.3.2	The parameters of do_cluster, and implementing merging and add	29
5.4	How to add distance functions	30
5.5	Parallelisation	30
5.6	Work allocation in MPI	30
5.7	Work allocation using Pthreads	30
5.7.1	Distributed Parallelisation	31
6	Testing	32
Comment by SH	33
7	Acknowledgements and copyright	34
7.1	Acknowledgements	34
7.2	Copyright	34
	Concept Index	35

1 Description

This manual describes `wcd` (pronounced ‘wicked’), a program that performs clustering of sequences. Essentially, it takes a set of DNA sequences (typically Expressed Sequence Tags, ESTs), and clusters the sequences into the related groups.

It reads in a set of DNA sequences (typically ESTs or mRNAs) in FASTA format and organises them into clusters of similar sequences, producing a list of clusters and their members on standard output.

The clustering performed is single-linkage clustering. Sequences are put in the same cluster if they have regions of close similarity. Similarity is determined using one of the standard distance functions. In versions > 0.2.0 of `wcd`, three distance functions are supported: the d2 function, edit distance and a common word heuristic. This may be expanded in later versions of `wcd`.

When using the d2 distance function, `wcd`’s results are similar to those generated by the `d2_cluster` program, which also implements clustering based on the d2 distance function. For a comparison of the two programs, See [Chapter 6 \[Testing\]](#), page 32.

The use of the d2 distance function in DNA sequence comparison is described in Hide et al. (1994) and Burke et al. (1999), and has been shown to be sensitive to sequence similarity even in the absence of direct sequence alignments. This corresponds with the nature of EST data which typically contains single-base read errors and may contain sequence from different splice forms of the same transcript (leading to local dissimilarity).

Thanks to the properties of the d2 distance function, sequence clusters created by `wcd` correspond to closely gene classes i.e. all the ESTs originating from a specific gene will be clustered into a single class.

`wcd` is written in C and has been successfully installed and tested on Linux, FreeBSD, SGI Irix, Sun Solaris and Windows 2000 (under Cygwin).

Users of `wcd` are encouraged to subscribe to the `wcd`-users mailing list using the form at: <http://swing.sanbi.ac.za/mailman/listinfo/wcd-users>.

The remainder of this manual assumes that a user understand the basic principles of what EST clustering is about, and so the basic concepts are only briefly reviewed in this chapter. Subsequent chapterd discuss installing and running `wcd`, as well as giving some technical background so someone who wants to use some of the code.

1.1 Clustering

The basic idea is that we take a set of sequences and cluster them so that sequences that are close to each other in the same cluster. There are many clustering algorithms, and one review can be found in

```
@article{cluster2,
  author =      "A. K. Jain and M. N. Murty and P. J. Flynn",
  title =      "Data clustering: a review.",
  journal =     "ACM Comp. Surveys",
  volume =     "31",
  number =     "3",
  pages =      "264--323",
```

```

    month =      sep,
    year  =      "1999"
}

```

The type of clustering we do is single linkage clustering. We create the largest number of clusters (and hence smallest clusters) such that if two sequences are very close they are put in the same clusters. Note the definition is transitive: if A and B are close, and B and C are close then A and C will be in the same cluster even if A and C are far apart.

The basic clustering algorithm is shown below (obviously, there are performance enhancements).

```

foreach sequence i
  put i in its own cluster
foreach sequence i
  foreach sequence j
    if i and j are close to each other
      merge the clusters to which i and j belong

```

1.2 Distance functions

A distance function takes two sequences and returns a number that says how far apart the sequences are mathematically. Many distance functions have been proposed with different biological and computational motivations: hopefully a well-chosen distance function will return a mathematical value that has some biological distance.

One of the purposes of `wcd` is to allow different distance functions to be used. The distance functions which `wcd` supports

- `d2`, edit distance

1.2.1 The D2 distance function

The `d2` distance function is based on the word count.

- Let x be a sequence and w a word

We use the notation $x' \in x$ to mean that x' is a *window* or substring of x of some specified length.

- $c_x(w)$ is the number of times that w appears in x .

1.2.1.1 General formulation

To measure the distance between sequences we compute the word frequencies and then take the sum of the square of the differences

For a fixed k , we examine all the words w of length k , and compute the frequencies which they occur in the strings x and y , and then sum the square of the differences. Formally,

$$d_k^2(x, y) = \sum_{|w|=k} (c_x(w) - c_y(w))^2$$

Then in general

$$d^2(x, y) = \sum_{k=l}^L d_k^2(x, y)$$

for constants l and L . In practice (and this is what `wcd` does), we compute the `d2` score for a fixed k . The default value of k is 6, but this is a parameter of the program.

1.2.1.2 Windows

The goal of EST clustering is to see whether there is an overlap between two sequences: thus we are not interested in the d2 score between the whole of one sequence and the whole of another, but whether there are two regions (called windows) that have a low d2 score.

Thus the d2 score between two sequences is the minimum of the d2 score between all pairs of windows of those two sequences. $d_k^2(x, y) = \min_{x' \in x, y' \in y} \sum_{|w|=k} (c_{x'}(w) - c_{y'}(w))$

The consequence of this is that d2 applied like this does not obey the triangle inequality. It is very common that

$$d^2(x, z) > d^2(x, y) + d^2(y, z)$$

since there is a window that x shares with y , another that y shares with z but no window common to both x and z .

As an aside, d2 even when applied to sequences is not a metric since $d^2(x, y) = 0$ does not necessarily imply that $x = y$.

We don't compare two sequences directly. Rather we compare all windows of fixed length in one sequence with all the windows in the other. The default window length is 100 (again it's parameterisable).

1.2.1.3 References

The paper by Burke et al describes the basic d2 method and justifies its use. The paper by Hazelhurst describes the algorithm that is implemented in wcd. This paper should be available from the ACM Digital library <http://www.acm.org/dl>.

```
@article{burke99,
  author = "J. Burke and D. Davison and W. Hide",
  title = "{D2\_cluster: A Validated Method for Clustering EST and Full-length cDNA Sequences}",
  journal = "{Genome Research}",
  year = 1999,
  volume = 9,
  number = 11,
  pages = "1135--1142"
}

@InProceedings{hazel2004c,
  author = "S. Hazelhurst",
  title = "An Efficient Implementation of the  $d^2$  Distance Function for {EST} Clustering",
  booktitle = "{Proceedings of {SAICSIT 2004}: Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists",
  pages = "229-223",
  year = 2004,
  editor = "G. Marsden and P. Kotz and A. Adesina-Ojo",
  series = "{ACM} International Conference Proceedings Series",
  month = oct
}
```

1.2.2 Edit distance

Edit distance is well known; the reference below gives a very thorough introduction to the subject.

The version of edit distance implemented in `wcd` does local alignment (Smith-Waterman). Penalties (positive numbers) are given for mismatches, deletions or gaps, and a reward (a negative number) is given for a match. The default values are:

- -1 for a match
- +3 for mismatch
- +5 for opening a gap (indel)
- +2 for extending a gap

The user can change these values.

```
@Book{      gusfield97,
  author    = "D. Gusfield",
  title     = "Algorithms on strings, trees, and sequences",
  address   = "Cambridge, United Kingdom",
  publisher = "Cambridge University Press",
  year      = 1997
}
```

2 Installing wcd

2.1 Getting new versions of wcd

The latest version of wcd can be found at:

<http://code.google.com/p/wcdest>

You should also be able to find it at: <ftp://ftp.cs.wits.ac.za/pub/research/software/>

Please let me know if you would like to be told about new versions of wcd.

2.2 Changes from versions 0.3.x

There are a number of changes from 0.3 to 0.4 in memory organisation and algorithm. The first two are important to note if you used the options as the appropriate default values have changed.

- The sample word length is no longer a parameter. It's fixed as a constant at 8. This was done after extensive empirical testing. Fixing so that a sample word fits in a machine word has numerous computational advantages. The -B option therefore no longer exists. The default value for the -K is now 6.
- Our second heuristic, the so-called t/v heuristic has been replaced in favour of a common word heuristic. The common word heuristic does seem better it is more expensive to implement. The -H or -common_word heuristic's default value is now 65.
- The SMP algorithm has been simplified. It is now possible to use the Pthreads and MPI parallelisation together. This may be useful when you have a cluster of multi-processor machines, but it may be easier just to run multiple MPI jobs.

2.3 Installing the distribution

The distribution comes as a gzipped, tarred file. The following commands should create a directory called 'wcd_distrib' with all the necessary files.

```
gunzip wcd_distrib.tar.gz
tar -xf wcd_distrib.tar
```

2.4 Compiling

There are two possible ways to compile and install wcd

2.4.1 Method 1: using configure

The INSTALL file gives a more detailed explanation and description. The following should work if you don't want anything fancy. If you do, or if you want to ensure that some compile flags are different, then read the INSTALL file.

```
./configure
make
make install
```

Note that the last instruction will attempt to install the executable and documentation in the appropriate places on your system and will probably require you to have root privilege or write permission for those standard directorites. If you don't have these, you can find

the executable `wcd` in the `src` directory and the texinfo documentation `wcd.info` in the `doc` directory. Manually copy these to the correct place.

If you are using FreeBSD, you may have to make a small change to the `common.h` file.

If you want a manual you can print out, say `make pdf`. Otherwise you can read the on-line version using the `info` system. If the documentation is installed in the right place on your system you can just say `info wcd`, otherwise you will have to say `info -f wcd.info`.

`wcd` should work on 64-bit, little-endian and big-endian machines.

2.4.1.1 MPI Version

From version 0.3.0, `wcd` has an MPI version that allows parallelisation of the clustering on a cluster of workstations.

```
./configure --enable-MPI
make
make install
```

Initial experimentation has shown that using the MPI option when not necessary leads to a 2\% degradation in performance, so the main benefit of not using the `enable-MPI` option is that `wcd` will compile without the MPI libraries.

2.4.1.2 Pthreads

From version 0.3.0, `wcd` has an MPI version that allows parallelisation on SMP architectures

```
./configure --enable-PTHREADS
make
make install
```

In version 0.4, the two parallelisation techniques complement each other. You can run on a cluster of SMP machines. However, an assumption is made that each of the SMP machines has the same number of processors. In version 0.3, binaries supporting both techniques could be used but only one used in a particular run; this no longer holds in 0.4.

2.4.2 Method 2: Manually using make

The `wcd` program is written in reasonably standard C. If you don't want to or can't use the autoconf method described above, try the following. Change directory to the `src` directory.

```
make -f WcdMake wcd
```

This was tested on different versions of RedHat and Suse and worked without problem. `wcd` is written in reasonably standard C, and so should work on most systems, but unfortunately the vagaries of different libraries and compilers might make the standard compilation fail. See the troubleshooting section below for some suggestions.

To get documentation, running `make -f WcdMake pdf` in the `src` directory will produce a pdf version of the manual (placing output in the `doc` directory). `make -f WcdMake info` will produce texinfo version.

2.4.2.1 Trouble shooting

If you know something about your system and are happy mucking about with make files, please read the technical note below and make the necessary changes. If not, try the following suggestions. There are three likely causes of problems: (1) `wcd` supports long

options (e.g. you can say `--cluster_compare` rather than `-D`); (2) your compiler is `cc` rather than `gcc`; and (3) your compiler does not support procedure inlining (inlining may make some performance benefit, but does not change functionality or use). The makefile `WcdMake` is the one to look at.

1. Try `make -f WcdMake simple` to see whether the problem is that you don't have the libraries that support long options. If this works, `wcd` will run as is, except that you cannot use the long options form for `wcd`, only the short-form options.
2. If your compiler is named `cc` rather than `gcc`, try saying `make -f WcdMake ccsimple`.
3. If that doesn't work, try switching off inlining as well by doing `make reallysimple`. In this case neither long options or inlining is used. If your compiler is `cc` then try `make -f WcdMake ccreallysimple`.

`wcd` was tested on a number of different systems. The table below shows what worked on those systems. With a little tweaking of the makefile to specify where `gcc` libraries on your machine are, you might get the standard `make wcd` to work too.

Linux RedHat and Suse

`make wcd`

Sun OS 5.10, Darwin, Aix, FreeBSD

`make simple`

Irix64, Alpha/OSF1 Tru64, Cray

`make ccreallysimple`

With the FreeBSD machine I managed to get the full version to work by finding the appropriate libraries and includes. The same thing probably applies to the others.

This section only need be read if `make wcd` did not work and you would like to have long options working.

The non-standard options of `wcd` are to use inlining and the long options library. If your system does not support inlining then the compiler must be given the `-DNOINLINE` flag when run. If your system does not provide a library for getting long options then the compiler must be given the `-DNOLONGOPT` flag.

If, however, your system does provide a long options library and you would like to use long options, then you must say where the include file (`'getopt.h'`) and where the library will be. You may also have to explicitly tell the compiler to use the `'libtextlib'` library. Find out where these things are, and change the variables in the `WcdMake` to the appropriate values and then run `make try`. For example, if the include file is in `/usr/share/contrib/gcc` and the library is in `/usr/local/lib` you might have

```
INCLUDES = /usr/share/contrib/gcc
LIBS      = /usr/lib
EXTRAFLAGS = gettextlib
```

You might or might not have the `EXTRAFLAGS` variable defined.

The makefile assumes that the compiler available is `gcc`. If it's not or if you wish to use a different compiler, then edit the file `Makefile` and change the definition of the variable `CC` to the appropriate compiler.

If this doesn't work, then it may be that your libraries are in non-standard places. Look at the `Makefile` and try the following things

- Add ‘-DNOLONGOPT’ to the *CFLAGS* variable. In this case, you can’t use the long form options (e.g. you must say `-g` rather than `--histogram`)
- Try replacing ‘gcc’ by ‘cc’.
- Add to the *CFLAGS* the directories where the include and libraries can be found.

wcd’s MPI and PTHREADS code are optional. In the code there are macros that protect all references to MPI and PTHREADS. Unless these macros are defined the compiler won’t know about them. This enables wcd to be compiled without having either libraries.

If you are making manually rather than through configure, and you want to use MPI or PTHREADS (either or both), then you must make sure that either (or both) the MPI or PTHREADS macros are defined. You could either put the relevant defines in the `wcd.h` file, or pass it to gcc with something like `-D MPI`.

2.4.3 EMBOSS

EMBOSS wrappers are available from the same source as the wcd code itself.

2.4.4 Auxiliary programs

A number of auxiliary programs are included. They may be useful but are not essential. They are either Perl or Python

See [\[Auxiliary Programs\]](#), [page 22](#), for more details. They should run as is, but if your version of perl or python is not in a standard place, edit the files and change their first lines to replace the line that says

```
#!/usr/bin/perl
with
#!/usr/local/bin/perl
or whatever.
```

In addition to these programs, there is a shell script `wcd_wrapper.sh` that allows wcd to be used as a replacement for `d2_cluster` in the `stackPACK` analysis pipeline.

2.5 Documentation

This info file can be read by saying

```
info -f wcd.info
```

To create the documentation say ‘`make pdf`, `make html`’, etc depending on what sort of document you want. This creates the following:

- ‘wcd.info’: the info file which can be read using the texinfo system. This is probably easiest for reference purposes. You can read the on-line version using the `info` system. If the documentation is installed in the right place on your system you can just say `info wcd`, otherwise you will have to say ‘`info -f wcd.info`’, or give the full path.
- ‘html’: a directory that contains all this help information in HTML files. Use your browser to read it.
- ‘wcd.pdf’: A PDF version of this help file, which is probably the best for a hard copy.

You can play around with ‘`makeinfo`’ etc. if you want different versions.

3 Running wcd

wcd can be invoked in different ways. The first just shows the usage; the others are functional. They take an input file in FASTA format and process it in some way. wcd numbers the sequences in the file from 0 upwards, and these indices are used to refer to the sequences.

3.1 Summary

Options or switches to wcd can be given in short form (e.g. -D) or long form (e.g. --cluster_compare). On some systems, only the short form options are supported.

```
Usage: wcd -v | -u | -h
```

```
-v: version
```

```
-u: usage
```

```
-h: usage
```

```
Usage: wcd [opts] <filename>
```

```
-c, --show_clusters: show clusters compactly
```

```
-d fname: use the file as a dump file
```

```
-F fun, --function fun: say which distance function to use -f fname, --init_cluster: u
```

```
-g, --histogram: show histogram
```

```
-H val, --common_word val: set common word heuristic threshold (default 5)
```

```
-j fname, --constraint1 fname: set constraint file 1.
```

```
-J fname, --constraint2 fname: set constraint file 2.
```

```
-K val, --sample_thresh val: give sample heuristic threshold
```

```
-l val, --window_len: set window len (default 100)
```

```
-n, --no_rc: don't do RC check
```

```
-o fname, --output fname: send output to file [stdout] -P fname, --parameter fname :
```

```
-R <filename> <ind1> <ind2>: perform cluster on a range
```

```
-s, --performance: show performance stats
```

```
-S val, --skip val: set skip value
```

```
-t, --show_ext: show extended cluster table
```

```
-T val, --threshold val: set threshold to value
```

```
-w val, --word_len: set d2 word length (default 6)
```

```
-X: clone linking
```

```
Usage: wcd [-i|--show_seq] <index> <filename>
```

```
show the sequence with the given index
```

```
Usage: wcd [-I|--show_rc_seq] <index> <filename>
```

```
show the RC of the sequence with the given index
```

Usage: wcd [-E|-e|-p] <filename> <ind1> <ind2>
 -E, --compare: show seqs, number common words, and d2scores
 -e, --abbrev_compare: show min of d2scores (pos + rc)
 -p, --pairwise: show pairwise d2 scores of all windows

Usage: wcd [--cluster_compare,-D] <seqfile> <indicesfile>
 compare two clusters

Usage: wcd [--merge,-m] <seqf1> <clf1> <seqf2> <clf2>
 merge two clusterings

Usage: wcd [--add,-a] <seqf1> <clf1> <seqf2>
 add a set of sequences to a known clustering

Usage: wcd [--recluster,-r] <clf1> <seqf1>
 recluster from a less stringent clustering

3.2 Examples — A Quick Introduction to wcd

This section shows common ways in which wcd is likely to be invoked.

3.2.1 Basic Clustering

The following examples show straightforward clustering examples.

- > wcd --show_clusters data/5000.seq
 Cluster the sequences found in the file data/5000.seq. Print the clusters on standard output in compact form. Use the d2 function to determine cluster membership.
- > wcd --histogram --show_clusters data/5000.seq
 As above, but also print a histogram that shows the size of the clusters found.
- > wcd --histogram --function ed data/5000.seq
 Cluster as above, but use edit distance as distance function.
- > wcd --output ans/5000.ans --histogram --show_ext data/5000.seq
- > wcd -o ans/5000.ans -g -t data/5000.seq
 As above, but print the clusters in extended, table format. Also save the output in a file.
- > wcd -c -N 5 data/5000.seq
If the wcd has been installed with the PTHREADS option. Run wcd on 5 processors at the same time.
- > mpirun -np 16 wcd -c data/5000.seq
If the wcd has been installed with the MPI option. Run wcd on 16 processors using the MPI libraries.

```
> wcd -X -c data/5000.seq
```

Cluster, but also use clone information. If two ESTs come from the same clone, they'll also be put together. The clone information comes from the FASTA file directly – it's the symbol that follows the word “clone” in the header. This is a convenient option, but for larger files it would be better to put this information in a constraints file.

3.2.2 More advanced clustering

3.2.2.1 Using a clustering file

This can be used to seed the clustering to start with. Instead of starting each sequence in its own cluster, we do some preallocation of clusters. `wcd` will then continue clustering using this as a start: no clusters will be broken up, but some of the clusters will be merged. This might be useful when you know for some reason that some sequences should be clustered together regardless of d2-score (e.g. from an annotation or from biological knowledge).

In this case, create a clustering file. Each cluster should be on a line by itself, terminated by a full stop ‘.’. This line can be as long as you like, but don't break it. For example, suppose we know that sequences 0, 2, 10, and 11 should be clustered together; and so should 6, 17, 107, and 120; and so should 151 and 152. Your cluster file (let's assume it's called *init.cl*) would look like this:

```
0 2 10 11.
6 17 107 120.
151 152.
```

Those sequences that are not mentioned will be put in their own clustering. Clustering is then done by saying:

```
wcd --show_clusters -f init.cl data/5000.seq
```

3.2.2.2 Using a constraint file

A constraint file enables you to specify additional knowledge about the data and so help `wcd` do clustering more efficiently and more correctly. Each line in the cluster file gives a directive. There are three directives.

- **fix.** Suppose you know that two sequences definitely should not be clustered together (you might know this from previous experiments). You can then tell `wcd` never to merge two clusters containing these sequences. For example, suppose sequences 1, 17 and 325 definitely do not belong in the same cluster. Then you would have as a line in the constraint file:

```
fix 1 17 325.
```

Note that fixedness is either all or nothing. There is no way in the current version of saying don't cluster 1 with 325, and don't cluster 45 with 360, but it's OK to cluster 1 with 45 or 360. If there turns out to be a need for it, it might be included in a later version of the program.

- **cluster-only**

The clustering table allows you to provide an initial clustering, which `wcd` can then refine. Sometimes you may only want to refine the clustering of some of the sequences. In which case you can make major performance savings by using the **cluster-only**

directive to tell `wcd` to only refine the clustering of some of the sequences and to leave the clustering of all the others as given initially by the clustering table. For example, if we had the following clusters: [0, 1, 4] [2,3,7] [5] [6,8] [9]; and we were generally happy with the clustering but wanted to see whether [2,3, 7] should be merged with [6,8], the following directive could be used

```
cluster-only 2 3 6 7 8.
```

`wcd` would then check to whether those clusters would be merged.

NB: It only makes sense to have one cluster-only directive. It can be as long as you like.

- **reset** This is similar to **cluster-only** except you want to say that you are happy with the clustering of the other sequences but not happy with the clustering of the specified sequences. Typically, you would be concerned that the clustering of the specified sequences was too lenient (i.e. that some sequences had wrongly been put together). So taking the above example, if you said

```
reset 2 3 6 7 8.
```

You would be saying that you wanted to leave the clustering of 0, 1, 4, 5, and 9, but you wanted to cluster the other sequences de novo, completely ignoring the initial clustering.

The major difference between **cluster-only** and **reset** is that with **reset** you are saying that you want to recluster the specified sequences de novo: you think that some of the sequences specified that have been clustered already should not be and you want to check again (probably using other parameters). With **cluster-only** you are happy with what clustering has been done, but you want to check whether there should be even more clustering.

- **cluster-others, reset-others.** This has the same semantics as the previous two except the specified sequences should be left as is and not processed, and the sequences not specified should be clustered again.

3.2.2.3 An example of reclustering and constraint files.

We had a very large data set to cluster with heterogeneous data. Some of the long sequences had very large overlaps. We did the following.

1. Prepare a suffix array of the data file (this part is explained in more detail later).

```
fasta2sary -x -d 10 bp.fasta -o bp.fasta.nlc
mksary bp.fasta.nlc
```

2. Cluster with a very high degree of stringency

```
wcd -c -F suffix -w 120 bp.fasta > bp120.clt
```

This creates a cluster table which contains one very large super cluster (of about 43k sequences) and lots of other very small clusters. We copy the line that contains the supercluster into a file `bp120.con` and put “cluster-others” in front of it

```
reset-others 6355 29988 2 9282 71821 4 .....
```

3. Cluster less stringently

We cluster again, less stringently. This time we initialise the clustering with the clustering given by `bp120.clt` subject to the constraint file. This says: leave the super-cluster as-is. Recluster all the other sequences from scratch.

```
wcd --init-cluster bp120.clt --constraint1 bp120.con -c -F suffix -w 90 bp.fasta > bp90
```

This shows no new super clusters so we throw away bp90.clt and continue with bp120.clt.

4. Cluster using a small word size

We recluster, leniently, (25-30 is probably a good choice of word size) and create a new cluster table bp30.clt. To recap what this does: leave the supercluster as is, and recluster everything else from scratch on the basis that two sequences should be put in the same cluster if they share a common word of length 30.

```
wcd --init-cluster bp120.clt --constraint1 bp120.con -c -F suffix -w 30 bp.fasta > bp30
```

This clustering is probably too lenient and so the clusters, except for the super-cluster, are probably too big.

5. Refine the clustering bp30.clt

Now we want to cluster again more strictly using the bp30.clt as the starting point. Within each cluster of bp30.clt we recluster afresh using the standard d2-clustering algorithm. We do not compare sequences from different clusters of bp30.clt to see whether they should be put together, but only compare sequences within the clusters of bp30.clt. If we didn't have to worry about the supercluster we would just say:

```
wcd --recluster bp30.clt -c bp.fasta
```

But we do have to worry about the supercluster. So there are two things we need to do: first, tell wcd not to look at elements of the supercluster; second, tell wcd to put all the elements of the super-cluster together. To do the first we use the constraint file. To do the second we extract out of bp120.clt the line with the supercluster and save it in a file, say 'super.clt'. This is exactly the same as 'bp120.con' without the 'cluster-only' directive and initialise the clustering with it. So all in all we say

```
wcd --recluster bp30.clt --init_cluster super.clt --constraint1 bp120.con -c bp.fasta
```

3.2.3 Merging and adding

These features of wcd enable you to combine two clusterings. You could do this de novo, but there are performance benefits of using these wcd features.

Suppose you have two files

- File 1, 'data1.seq': 15 sequences, numbered 0 through 14. The clusters (as produced by wcd and saved in a file 'data1.cl')

```
0 1 2 12 13 14.
```

```
3.
```

```
4 5 6 8.
```

```
7 9 10 11.
```

- File 2 ('data2.seq'): 12 sequences, numbered 0 through 11. The clusters (as produced by wcd and saved in 'data2.cl') are:

```
0 2 4 10.
```

```
1 3 5.
```

```
6 7 8 9 11.
```

You now want to merge the two files. You are happy with the clustering of the two files with respect to themselves, but you now need to see whether the sequences in the one file are related to the sequences in the second file. You would do this by saying:


```
wcd --show_clusters --merge data1.seq data.cl data2.seq data2.cl
```

This merges the two clusterings. All the sequences in the first file will be compared to all the sequences in the second. The new clustering would be output. The sequences in the second file will be renumbered 15 to 26. For example a possible output might be:

```
0 1 2 12 13 14.
3 21 22 23 24 26.
4 5 6 8 16 18 20 7 9 10 11.
15 17 19 25.
```

This could happen if sequence 3 in file 1 is related to sequence 21 (6 in file 2); sequence 4 in file 1 related to sequence 16 (1 in file 2); and sequence 7 in file 1 to 18 (3 in file 2).

Adding

Adding allows you to add unclustered sequences into a cluster.

```
wcd --show_clusters --add data1.seq data1.cl data2.seq
```

would add the sequences in the file `data2.seq` to the ones in `data1.seq`, clustering as appropriate.

3.3 Help

Usage: wcd -u |-h

```
-u: usage
-h: usage
-v: shows version
```

Shows how to invoke wcd

3.4 Identifying a sequence

```
Usage: wcd [--show_seq, -i] <index> <filename>
       wcd [--I show_rc_seq] <index> <filename>
```

- `wcd --show_seq i dataf`
Prints the i-th sequence in the data file dataf
- `wcd --show_rc_seq i dataf`
Prints the reverse complement of the i-th sequence in the data file dataf

3.5 Comparing two sequences

Warning: These options *may* be removed in later versions of d2-cluster, and should be treated with caution.

These options are included to allow exploration and evaluation of data rather than for clustering purposes. The problem is that optimisations made for performance reasons have meant that they do not give completely accurate answers. For example, if we find windows where the d2 score is less than a threshold, we announce success; we don't try to find the pair of windows with the smallest overlap. In subsequent releases there may be a separate program which provides these facilities (though with less efficient code).

All these options can also take as options the options which allow changing of threshold, window and word size.

```
Usage: wcd [--compare|-E] <filename> <ind1> <ind2>
       wcd [--abbrev_compare|-e] <filename> <ind1> <ind2>
       wcd [--pairwise|-e] <filename> <ind1> <ind2>
```

- `wcd --compare dataf i j`

Compares the sequences *i* and *j* from the datafile *dataf* and prints out the following

- The *i*-th sequence, the *j*-th sequence, and the reverse complement of the *j*-th sequence
- A line with the following information
 - *i* and *j*
 - An estimate the number of samples of the *j*-th sequence which appears in the *i*-th.
 - An estimate the number of samples of the *j*-th sequence which appears in reverse complement of the *i*-th.
 - An estimate the number of words of the *j*-th sequence which appears in the *i*-th.
 - An estimate the number of words *s* of the *j*-th sequence which appears in reverse complement of the *i*-th.
 - the d2 score between *i* and *j*
 - the d2 score between *i* and the reverse complement of *j*

- `wcd --abbrev_compare dataf i j`

Prints the minimum of the: (1) the d2 score of *i* an *j*; and (2) the d2 score of *i* and the reverse complement of *j*.

- `wcd --pairwise dataf i j`

First prints a table of the d2 scores of all windows of sequence *i* compared to all windows of sequence *j*. Then does the same with the RC of sequence *j*

```
Usage: wcd [--cluster_compare, -D] dataf clusterf
```

Takes two arguments: a data file with sequences, and a file that gives two clusters. This cluster file should contain exactly two lines. Each line should contain the indices of the sequences belonging to a cluster. The indices should be separated by spaces, and the line terminated by a full stop.

The program will then compare each sequence in one cluster with each sequence in the other and print out the d2 scores (both positive and RC). One each line of output there is the result of one comparison. First the two indices are printed out, and then the two d2 scores.

3.6 Doing clustering

```
wcd [opts] <file>
```

3.6.1 Arguments

When used in this way, `wcd` takes one argument: the name of the input file.

3.6.2 Options

- `[--output | -o] fname`

By default all `wcd` output goes to standard output. Using this option allows you to specify another file

- `[--num_seqs | -C] val`

By default all the sequences in the input file will be processed. If you only want to process part of a file, you can use the `-C` option.

e.g. `--num_seqs 100`

will only process the first 100 sequences.

If you specify a number greater than the number of sequences actually in the file, then the whole file will be processed.

- `--show_clusters, -c`

Prints the results of the clustering in a compact way. Each cluster is printed on a line by itself. The sequences that make up the cluster are separated by commas. See [Section 3.10 \[Output\], page 21](#), for more description.

- `--histogram, -g`

Show a histoGram of the results of the clustering. For each cluster size, it shows how many clusters there are that size up to some maximum size.

- `--show_ext, -t`

Prints the clustering in extended format. See [Section 3.10 \[Output\], page 21](#), for a description of the format.

- `[--output | -o] fname`

By default any output gets sent to standard output. You can send output to a given file.

- `[--function | -F] fun wcd` decides whether two sequences should clustered together on the basis of a distance function. The distance function that can be used are

- `--function d2`: use the d2 function. This is the default. The default threshold is 40.
- `--function ed`: use edit distance (local alignment). The default threshold is -20. See the `--parameter` option below for how to specify other options.
- `--function heuristic`: use the common word heuristic described below. The common word heuristic gives a crude and fast membership criterion.
- `--function suffix`: A pair of sequences are clustered together if they share at least one word. A good value to use is 25: note that the default value of 6 is a very bad value.

- `[--parameter | P] fname`

This specifies a parameter file that parameterises the distance function. In the current version this is only used by the edit distance option. The first four lines specify a 4x4 matrix which give penalties for substitutions (a,c,g,t vs a, a, c, g, t). There are four integers per line which should be separated by a single space. The fifth line gives two integers, separated by a space which give the cost for opening a gap and extending a gap. The file that would be used for the default parameters is shown below

```
-1 3 3 3
3 -1 3 3
3 3 -1 3
3 3 3 -1
5 2
```

- `[--common_word | -H] val`

Set the common word heuristic threshold (the default is 65). Before running a d2 check between 2 sequences, this first checks to see how many distinct 6-words are shared between the sequence (NB, the sequence, not some windows). This can be done in linear rather than quadratic time and so is probably 2 orders of magnitude faster than checking d2. If not enough common words are found, a d2 check will not be done. [NB: this has changed from version 0.3]

- `[--window_len | -l] val`

Set the window length to the given value. The default is 100

- `[--skip_val | -S] val`

Set skip value — how much the window along the second sequence should be updated. The default is 1. Don't be too aggressive. Setting the common word threshold at its default value is probably better than changing the skip value (IMO).

- `[--threshold_val, -T] val`

Set the distance threshold — the default is 40 for d2, -20 for edit distance.

- `[--word_len | -w] val`

Set the d2 word length (default 6)

- `--performance, -s`

Show performance stats

- `--no_rc, -n`

Don't do the reverse complementation check (rc-checking is done by default).

- `[--constraint, -k] filename`

Give the constraint file for the first input data file. This is optional. The constraint file enables you to ensure that certain sequences are not clustered together or to ignore certain sequences while clustering. See [\[Format of Constraint File\]](#), page 20, which gives more details on the required format of the constraint file, and the semantics.

- `[--sample_word_len | -B] val`

Word length used in the *sample* heuristic. See below for use.

- `[--sample_thresh | -K] val`

This is the threshold for the *sample* heuristic. Suppose K and B are the sample word length and threshold parameters. When comparing two sequences i and j , the first sample test described below is done. If it passes, a more rigorous test is done for similarity; if it fails the pair is declared not to have overlap.

The sample test: When comparing two sequences i and j every 8-th word of length B is sampled from j ; at least K must also occur among all the words in i . The defaults are $K = 7$, $B = 8$ which is conservative. [NB: this has changed from version 0.3]

- **-X**: Do clone linking

wcd will use the clone information in the sequence headers to put sequences together. If a sequence header contains the word *Clone* followed by a string, then that sequence is identified as matching a particular clone. All ESTs matching a particular clone will be clustered together. (The current implementation is very simplistic and probably adds about 25\% cost to clustering. It will be improved in future).

3.6.3 Clustering based upon suffix arrays

This provides a coarse clustering very quickly. It puts two sequences in the same cluster if they share at least one word (of the specified word length). You need to create a suffix array of the input data file. wcd expects a certain naming convention to be used.

In order to use this facility you must create some auxiliary data files in the same directory as your main sequence file. I assume you have available the **mksary** suffix array package, though in principle others should do. ‘mksary’ can be found at <http://sary.sourceforge.net/>

Once ‘mksary’ has been installed, do the following

```
./fasta2sary data.fasta -o data.fasta.nlc
mksary data.fasta.nlc
```

This will leave three files **data.fasta**, **data.fasta.nlc** and **data.fasta.nlc.ary**. Again: wcd expects this naming convention to be met.

To cluster

```
wcd -F suffix -w 30 -c data.fasta
```

3.7 Clustering a range

Usage: wcd [--range, -R] data*i* *j*

The range option allows clustering only a range or slice of the input data file in the following way.

- Each sequence in the file with an index from *i* (inclusive) to *j*-1 (inclusive) is compared to each sequence from *i*+1 to the end. More precisely

```
for(k=i,k<j,k++)
  for(m=k+1, m<num_seqs; m++)
    compare sequences k and n and cluster if necessary
```

If you think of the all the comparisons to be done as the upper half of a matrix, the range option restricts the comparisons to be done to a slice of this matrix. The purpose of the option is to allow a simplistic and crude parallelisation of work. We can run the multiple wcd processes with the same data but different slices. Typically we would also use the **--dump** option with this. See the section on parallelisation in the technical manual.

If the dump option is chosen, just before completion wcd creates a file with an **-FIN** suffix. For example if the dump file name is *range10*, then a file called *range10-FIN* is created. This enables monitoring programs which run asynchronously and cannot control the wcd process directly to use the file system to determine whether the job has completed.

3.8 Refining clusters

3.8.1 Merging clusters

This can be used to merge two known clusterings. The input is two FASTA files with the sequences and two files that give the clusterings.

It assumed that the two FASTA files are disjoint.

```
Usage: wcd [--merge,-m] <seqf1> <clf1> <seqf2> <clf2>
        merge two clusterings
```

Here you merge two clusterings that have already been computed. The four arguments are: the first FASTA file, the first clustering file, the second FASTA file, and the second clustering file. These are mandatory.

The `--constraint` option may be of particular use here. This can be used to constrain the first input file and its related clustering. You can use the `--constraint2` option to constrain the second input file.

The files that specify the clustering must be in the same format as produced by the compressed clustering format.. The sequences are referred to by index number (the position of the sequence in the input file), numbered from 0. Each cluster is given on a line by itself terminated by a full stop: the indices of the sequences in the cluster are printed out, separated by spaces.

The output is a a new cluster table in the same format as the input cluster table. The indices shown in the table are:

- The same as the input index if the sequence came from the first file specified.
- n +input index if the sequence comes from the second file, assuming n sequences in the first file.

Another useful option for merging is:

```
[--constraint2, -k] filename
```

Give the constraint file for the second input data file (it. This is optional. The constraint file enables you to ensure that certain sequences are not clustered together or to ignore certain sequences while clustering. See [\[Format of Constraint File\]](#), [page 20](#), which gives more details on the required format of the constraint file, and the semantics.

3.8.2 Adding sequences

This can be used to add a number of new sequences to an existing cluster. It is assumed that the new sequences do not exist in the original file.

The input is two FASTA files and a cluster table for the first file. The remarks above apply here.

3.8.3 Reclustering

```
Usage: wcd [--recluster,-r] <clf1> <seqf1>
        recluster from a more stringent clustering
```

This takes a clustering based on a more lenient (or just different) criterion and reclusters using d^2 -scores as the basis for clustering. The clustering as given by the input cluster table

is given as a scheme. For each cluster of the initial cluster table, `wcd` does a d2-clustering on the sequences in that cluster, ignoring all the other sequences. `wcd` will never compare the sequences in one cluster with the sequences in another. The resulting clustering is therefore a finer partition.

3.9 Format of input files

The format of input files is:

- FASTA format
- will treat Ns randomly.

What is meant by FASTA format? Each sequence **MUST** be preceded by an identification line. Each sequence itself may be on one line, or it may be on several lines. If it is on several lines, each line should terminate with a carriage return and there must be NO spaces on each line.

The identification line starts with a ‘greater-than’ sign (>). This is all that is required. IF there is an alphanumeric sequences (string with no blanks) IMMEDIATELY following the greater than sign then that is treated as a sequence ID that is used by a few of the options for display purposes. The rest of the identification line is completely ignored.

Format of clustering input

The merge and add options require as input files that specify a clustering. These files must use the compressed format described below.

Format of constraint file

Constraint files consist of a sequence of constraints, each on a line by itself. Each line in the constraint file is a directive followed by a list of indices, terminated by a full stop ‘.’. There are three directives and their semantics are described below.

- **fix**

This directive can be used to specify a list of sequences which should be labelled *fixed*. Any cluster than contains a fixed sequence will be labelled as fixed. This is useful when the user has some external knowledge about the clustering and wants to ensure that some sequences aren’t clustered together (e.g. by a poor quality EST).

Normally when a program starts, each sequence is put into a cluster. By default, a sequence is put into a cluster by itself, but if a clustering file is given then the clustering specified by that will be used.

Thereafter, clustering starts. However, if the **fix** directive is used, two sequences that are labelled as fixed will never be merged. If an EST matches more than 1 fixed cluster it will be added to at most 1 of them. Note that sequences that are not fixed can be added to fixed clusters, and a non-fixed cluster can be added to a fixed cluster.

- **cluster-only**

This tells `wcd` to only try to cluster those sequences that in the list (ignore the rest). This is useful if you only want to cluster a part of an input file (e.g. you might know the clustering for rest of the file).

NB: It is an error to put more than one **cluster-only** or **reset** in a constraint file.

- **reset**

This is similar to **cluster-only** but in addition, the clustering of the sequences in this list is reset to the default clustering (i.e. each sequence in the list is put in its own cluster).

This is used where you are given a clustering file as input, but while you are generally happy with the clustering given for some sequences, you would like others to be reclustered. The implication is that those sequences in the reset list

- will be clustered using d2 into one or more clusters;
- wcd will not attempt to cluster them with any of the other sequences

3.10 Output

All output goes to standard output. Look at the arguments section to decide the format of the output. Note if you don't have any format arguments, nothing will get printed, which will be a waste.

Format of the Compressed Cluster Table

A convenient way (for humans and probably for many programs) to show the output of the clustering process is to use the **--show_clusters** option. The format of the compressed cluster table is very simple. Each cluster appears on a line by itself. The cluster is given by listing the indices of the sequences that make up the cluster. The indices are separated by a space, and the last sequence in the cluster is followed by a full stop '.'.

```
0 1 3.
2.
4.
5 7.
6 8 9.
10.
```

Format of Extended Cluster Table

When the code **--ext_show** option is chosen, the clustering is given in table format. The columns of the table are as follows:

- the sequence identifier (note that the sequences are numbered from 0, in the order that they appear in the input file);
- cluster number: in each cluster, one sequence is chosen as the representative of the cluster and its index is used for the cluster. You can identify the roots because their indices are the same as their cluster numbers.
- link: the number of another sequence in the cluster. It is guaranteed that if you start at the root, or representative sequence in the cluster, you can traverse the entire cluster using the link field. It is not guaranteed that two adjacent nodes are within the d2 threshold.
- orient: the orientation of that witness (positive or RC) with respect to the root or representative sequence in the cluster.
- witness: the number of another sequence in the cluster which is within the d2 threshold of that sequence. This may or not be the same as the *link* field. Note that the value of

the *link* field is an artifact, merely a convenient way in which can list all the sequences in one cluster, whereas the *witness* field tells us about two sequences that do overlap.

The *orient* field requires a little more explanation. It gives the orientation of the sequence with respect to the *root* of its cluster. Formally, this is described as follows. Let x and y be two sequences in a cluster. While they may not overlap, we know that there is an ordered list or path of sequences $x = x_0, x_1, \dots, x_{n-1}, x_n = y$ such that for each i either $d^2(x_i, x_{i+1}) \leq \theta$ (positive match) or $d^2(x, rc(x_{i+1})) \leq \theta$ (reverse-complement match), where θ is the threshold. In particular for every sequence there is such a path from the *root* of the cluster to that sequence. In a path from the root to a sequence x , we compute the number of times the match is a positive one, and the number of times it is a reverse-complement one. The **orient** field is 1 if the number of reverse-complement matches is even, and -1 if the number is odd.

In *principle* it is possible for there to be two paths from the root to a sequence which would yield different *orient* values. First, this is unlikely to happen. Second, all the orient field is saying is that such an orientation of the sequence is legitimate. The fact that other orientation is also legitimate does not affect the correctness of the result.

The Dump option

Usage: `wcd -d dump_file seq_file`

When used with this option, `wcd` will open the given dump file for writing and then perform clustering. Whenever it finds two sequences that should be clustered it writes the match to the dump file: the output are the indices of the two sequences, and a 1 (if there is a positive match) or -1 (if there is an RC match).

This was introduced into `wcd` to support our simplistic parallelisation (see the parallelisation section in the technical manual).

3.11 Auxiliary programs

A number of auxiliary programs come with the `wcd` distribution.

- **rindex.py**

This Python program takes two arguments, the names of two files each containing (compact) clusterings. It computes the sensitivity, specificity, Jaccard index, Rand index and correlation coefficient between the two clusterings.

If you use the `-index rand` option only the Rand index is shown. If you use the `-index jaccard` option, only the Jaccard index is shown.

If you use the `-diff n` option, the indices above are not printed but the mismatches between the two clusterings are shown. First the pairs that are clustered by the first cluster but not the second are shown, and then the ones clustered by the second but not the first. If $n=1$, then all such pairs are shown; otherwise only the pairs that belong to clusters with n or fewer sequences are shown. This is helpful to explore differences in clusterings.

- **ext2comp.pl**

This Perl program converts the extended cluster table format to the compressed table format.

- `comp2ext.pl`

This Perl program converts the compressed table format to the extended table format. Since all the information of the extended table is not in the compressed format, you will find 0s in the *orient* column and -1 in the *witness* field.

For both programs, input and output are standard input and output. So you would probably run the programs thus

```
./ext2comp.pl < cluster.ext > cluster.com
./comp2ext.pl < cluster.com > cluster.ext
```

- `fasta2sary.py`

This takes as input a FASTA file and produces the file in a format suitable to produce a suffix array. It can do simple clean up as well.

```
python fasta2sary.py -x -d 11 myfile.fasta -o myfile.fasta.nlc
```

Note the convention that should be used. The output file must be the same as the input file with `.nlc` appended.

- `analysecluster.py`

Takes as input a `clt` file and produces a histogram of the cluster tables. If you use the `-t N` option, instead of the histograms any clusters with more than `N` sequences are output.

- `combine.c`

This takes as input a list of names of dump files, reads in each dump file in turn, and constructs the clustering from that. To make the executable, say `make combine`.

- In addition to these programs, there is a shell script `wcd_wrapper.sh` that allows `wcd` to be used as a replacement for `d2_cluster` in the `stackPACK` analysis pipeline.

3.12 Running wcd in parallel

`wcd` has support for both shared and distributed memory parallelisation. There are, however, *major* restrictions should you use these options. In version 0.3.0, the `wcd` options for merging, reclustering, dealing with constraints etc, are NOT supported when you use the parallel options. It is my intention that future versions will fix these problems. The following options are not supported if you use the parallel options.

- suffix-last based clustering
- `-show_seq`, `-show_rc_seq`
- `-E`, `-compare`: show seqs, number common words, and `d2scores`
- `-e`, `-abbrev_compare`: show min of `d2scores` (`pos + rc`)
- `-p`, `-pairwise`: show pairwise `d2` scores of all windows
- `-cluster_compare,-D]` compare two clusters
- `-merge,-m`: merge two clusterings
- `-add,-a`
- `-recluster,-r`

3.12.1 Shared Memory Parallelisation

If you are running `wcd` on a shared memory processor with multiple threads, the `--num_threads` or `-N` option can be used to specify how many threads should be used. If there's a close match to the number of CPUs that are available and unloaded, you should see a performance improvement though the current version is not very scalable.

3.12.2 MPI Parallelisation

By enabling MPI support when installing, `wcd` can be used in a cluster of workstations. A description of MPI is beyond the scope of this document. Use `mpirun` to run `wcd` (which takes the normal parameters). This code has been tested using LAMMPI (RedHat, Suse, MacOS X), MPICH (Ubuntu) and MVAPICH (Suse).

For example, using LAMMPI the `lamboot` command specifies what processors are available (the list is given in the hosts file – in its simplest form a list of the machines or their IP addressed). The `mpirun` command is then used to run `wcd`. A simple example follows.

```
lamboot hosts
mpirun -np 4 wcd -c sample.fas
```

This will run `wcd` on 4 different processors (these processors may be real or virtual, depending on what's available on the machines specified by the hosts file). When `wcd` runs like this with multiple processors available, one version of `wcd` runs as the master, and the rest as slaves. The sequence input file must be available on the master node, but need not be on the others.

The master process does not do any clustering itself, but merely coordinates the clustering process. In the above example, this means you would be running a master and three slaves and so could expect a 3-fold improvement in performance at best. The computational load on the master is fairly small and so it is safe (memory being available) to schedule both a master and a slave on same processor.

In future versions of `wcd`, the behaviour is likely to change so that the master does do clustering (to make it more memory effective).

NOTE: When you install `wcd` you can enable both Pthreads and MPI so that the executable can do both. BUT: Do NOT try to use the Pthreads and MPI at the same time (this will be something that goes into a later version of `wcd`).

4 wcd inside stackPACK

This chapter written by Peter van Heusden

Electric Genetics' **stackPACK** is a widely-used software suite used for expression variance analysis and transcript reconstruction.

By default, **stackPACK** uses **d2_cluster** as the clustering tool in its analysis pipeline. To incorporate **wcd** into **stackPACK** follow these steps: \

1. Install **wcd** as described in See [Chapter 2 \[Installing wcd\]](#), page 5. If you wish to use **wcd** from the **stackPACK** web user interface, you need to ensure that **wcd**, **wcd_wrapper.sh** and **comp2ext.pl** are runnable by the web server user.
2. Edit the `'/etc/stackpack'` file. Find the section labelled `[d2_cluster]` and changed the line reading `executable=` to refer to the location of the **wcd_wrapper.sh** script.

Please note that at present, you cannot do 'add' clustering in **stackPACK** using **wcd**.

5 Technical manual

Hopefully this will fill out a bit and become a little more coherent.

5.1 Program structure

Originally `wcd` was intended to be one small program and so would fit into one file. However, it grew and we needed to break up the program. The main files are

- `wcd.h`: This contains the critical type definitions of the program.
- `wcd.c`: This is the main program. It contains some utility and auxiliary code, processes all the command-line arguments and then calls the right routines. The key function in this code is the *docluster* code. This does the pair-wise comparison of the sequences and where the comparison falls below the thresh-hold merges the clusters, calling the appropriate find-union data structures.
- `ed.c`: this contains the routiness for computing the edit distance.
- `d2.c`: this contains the routines for computing the d2 score.
- `common.c`: this contains code and declaration for manipulating words and sequences

5.2 Data structures

The key data structure used is a find-union data structure. This discussion assumes that you know how that works — look in a standard algorithms text like Cormen, Leieserson and Rivest if you don't.

All the sequences in one cluster will belong at the end to the same tree in a find-union forest. Each sequence will point to the root of the tree.

To represent sequences we use an array of following structs

```
typedef struct SeqStruct {
    seqType seq;    // the sequence
    int    len;     // length
    string id;      // sequence id (from data file)
    int    cluster; // refers to index of root
    int    rank;    // for find-union
    int    next;    // for find-union: next in linked list of cluster
    int    last;    // for find-union: last ditto
    int    match;   // index of another seq for which a match was made
    short  orient;  // pos or rc for that match
    int    flag;    // flags for fix, reset etc
} SeqStruct;
```

```
typedef SeqStruct * SeqPtr;
```

```
SeqPtr seq;
```

The key variable is `seq` which is declared globally and statically. The main function opens up the data file(s) and computes the number of sequences in the file. Memory is then

allocated for `seq` which is then treated as an array thereafter. The values are actually given to the individual array elements in the function `read_sequences`.

The fields of the struct are:

- `seq`: the sequence itself: we store the sequences compactly – see below.
- `len`: the length of the sequence
- `id`: the sequence ID — this comes from the data file (the string to the right of the '>').
- `cluster`: This is the index of the representative sequence for the cluster to which a sequence belongs. This is initialised so that each sequence is the root of its own cluster.
- `rank`: this is the rank used in the rank-heuristic of the find-union algorithm. Roughly it describes how many sequences are in the tree rooted at this sequence. This is only valid for sequences that are the root of the cluster.
- `next`, `last`. These are not standard f-u fields. However, we want to not only know which sequence belongs to which cluster but to be able to print out the clusters efficiently. Thus, besides keeping the fields needed for the f-u structure, we also keep a linked list of all sequences in the cluster. `next` points to the next sequence in the list. The root will always be the first element in the linked list and for the root, the `last` field will point to the last sequence in the list. The `make_union` function updates these values.
- `match` is an index of an arbitrary other sequence in the cluster for which there is an overlap.
- `orient`: orientation of the match (-1 or 1). Initially this is set to 1, since each sequence starts being the root of its own cluster. The following algorithm is used to update it. Suppose we find a match between a sequences i and j . We consider the orientations of i and j with respect to their current roots:
 - If the match was positive, then if the orientation of i and j to their respective roots is the same, then we know that if we take the union of the two clusters then the orientation of the elements of the two clusters are consistent and no changes need to be made.

However, if the orientations of i and j to their respective roots are different, then we know that the orientation of the clusters is not consistent. Therefore, we must invert the orientations all the sequences in one of the two clusters before merging. The choice of cluster is arbitrary but we choose the one that will become the 'child' cluster.

 - If the match was a reverse-complement match, the situation is the reverse of the above case. If the orientations of i and j are the same, then we know the orientations of the two clusters is inconsistent, and so change the orientations of the sequences in one of the two clusters.

Storage of sequences.

Different versions of this program have taken different approaches to the storage of sequences. Initially, I went for one base per byte. Then, I went for four bases per byte. The reason for this (explained below is to save memory). However, this does make the code more complicated and cause recomputation as we need to extract out information. Then I went to a redundant representation of the data – each word being stored in one machine word. However, this was too memory intensive for large data sets so I went back to the compact representation

The data is stored in compressed form. The sequences are read in in ASCII format, one base per byte. However, we only need two bits to represent the four bases, and so we compress the data storing four bases per byte. There is a little complexity in this — the macros `GETWORD` and so on look like witchcraft and some performance penalty in time, but if we have 1M ESTs then it may make a few 100M of memory difference which may be useful.

A down side of the compressed format is that we can only treat Ns that are in the data as some arbitrary symbol. It would be better to ignore all words that contain N, but there is no way of knowing this in the compressed format.

The key data structures are shown below. Each sequence is an array of integers: the definition of `seqElt` is key. Have a look at the `wcd` code to see the current version (this may change). In 0.1.8 at least we used `int32_t`. If you want to change this to some other type, change the definition of `seqElt` and change the definition of `SEQELTWIDTH` to reflect this change (i.e. the bit width).

```
typedef int32_t    seqElt;
typedef seqElt    * seqType;
#define SEQELTWIDTH 32
#define BASESPERELT (SEQELTWIDTH/2)
```

In `read_sequences` each sequence is read into memory in turn. Once we know the length of the sequence, we divide through by `BASESPERELT` to compute the length of the needed array and allocate memory accordingly.

The maximum EST length is set by a macro constant definition `MAX_SEQUENCE_LENGTH`. It's currently 64K, but can be changed to any reasonable value. The only real limitation is that the `next_seq` variable in `read_sequences` is declared in the function and there might be a limitation on how the memory allocated on the stack can be on some machines. In version 0.1.8 for example this would mean that the the actual limit on sequence length is 64K*8 which is probably sufficient. If you run into this problem you may consider moving the declaration globally.

5.3 The algorithms

5.3.1 Heuristic

Since `d2` is an expensive test, a heuristic test is done before `d2` is called. This is based upon the idea that for a `d2` score to be under the threshold, the two sequences must share a number of non-overlapping common words.

I have not had time to work on this analytically though I think it can be shown. However, I have done some empirical work which showed that if you graph `d2` score (y axis) against common word score (x axis) that all the data points lie above the line $d2 = -12.5cw + 150$. This means that at a `d2` threshold of 40, all matches have over 8 common words. Thus the default set of 5 is very conservative.

The heuristic is a little more complicated than this. If we are comparing sequences `u` and `v`, then

- First we build a table of all words in `u`.

(Note that this is not particularly expensive to do. When we do the clustering we have a double loop

```

    for(i=0; i<n; i++) {
        for(j=i+1; j<n; j++) {
            compare i and j

```

the building of the table is done in the outer loop, outside of the inner loop and so the cost is negligible when amortised over all values.)

- Then if the threshold for the number of common words is at least 5, we sample the words in *v* ; If there are fewer than 2 matches, we report failure.
- If there are more than 2 matches, we then see how many non-overlapping words in *v* appear in *u*. If they are above the threshold we report success

5.3.2 The parameters of `do_cluster`, and implementing merging and add

Normally, to do a clustering we need to compare each sequence to each other. The obvious way of doing this is by using a double loop:

```

for(i=0; i<num_seqs; i++) {
    for(j=i+1; j<num_seqs; j++) {
        blah, blah
    }
}

```

When we do a merge, we have two separate clusterings and there is no need to compare the sequences in the first half with each other, and the sequences in the second half with each other but only those sequences in the first half with those in the second. Assuming there are *n1* sequences in the first part, the code becomes

```

for(i=0; i<n1; i++) {
    for(j=n1; j<num_seqs; j++) {
        blah, blah
    }
}

```

When we do an add, we have a clustering that we add sequences to. The clustered data is in the first part and the new data is in the second. There is no need to compare the sequences in the first half with each other. The code becomes

```

for(i=0; i<num_seqs; i++) {
    for(j=max(n1,i+1); j<num_seqs; j++) {
        blah, blah
    }
}

```

Each of these three cases can be dealt with by passing as parameters of `do_cluster` the bounds of these two loops. The indices are set in the main function based on the program's arguments.

In addition, sometimes we only want to cluster some of the sequences. There are two ways in which this can be done. One is to just set the *ignore* flag for the sequences that should be clustered to an appropriate value. The other is to pass the `do_cluster` function an array which contains the indices of those sequences to be clustered. This *index* array can then be used to extract out the indices of the sequences.

Thus there are four parameters for the `do_cluster` function:

- `i_end`: where the i-loop should finish looping.
- `j_beg`: where the j-loop should start looping
- `j_end`: where the j-loop should end looping
- `index`: an array of the indices of sequences to be clustered.

If this parameter has a `NULL` value, then all the sequences from 0 to `i_end` will be compared to all the sequences from `j_beg` to `j_end`.

If this parameter has a non-`NULL` value, then all the sequences from `index[0]` to `index[i_end]` will be compared to all the sequences from `index[j_beg]` to `index[j_end]`.

5.4 How to add distance functions

If you want to add a new distance function, say *farfun*, you need to implement three functions

- *farfun*: this should actually implement the distance function. It need not return the correct result: however if the distance between the two sequences is less than the threshold then the function should return a number less than the threshold (i.e. once you detect that the value is less than the threshold you can just return an estimate, you don't have to find the actual value)
- *farfunpair*: this should implement the distance function but should return the correct answer.
- *farfuninit*: this is initialisation code that will be called when the program runs.

The code should be placed in a file called `farfun.c` and the prototype should be put in `farfun.h`.

Then in the `wcd.c` add to `chooseFun`.

In `common.c`, call the initialise code in `initialise`.

5.5 Parallelisation

This section first discusses work allocation in MPI and Pthreads and then discusses some more primitive support that `wcd` has for parallelisation.

Different work allocation methods are provided for MPI and Pthreads. This is to allow for experimentation and future releases of `wcd` are likely to change the way in which they do things.

5.6 Work allocation in MPI

In MPI, work allocation is done statically. The input file is read in and estimates are made of the clustering costs based upon sequence length and the number of sequences. The workload is then allocated to the slaves and the slaves are launched. This may well change in future so if you write a script that uses this option, please check in future releases.

5.7 Work allocation using Pthreads

In Pthreads, the work allocation is done dynamically. Conceptually the work is broken up into a `d x d` grid of work and the work is then allocated to slaves. As there are many more blocks than slaves, each slave will have to ask for work many times.

5.7.1 Distributed Parallelisation

`wcd` has a number of options to support a simple parallelisation on a local network. The approach is very simplistic: we had a very large real job (110k mRNAs, 250M of data) that needed to be clustered in a hurry and urgently, so we hacked an inelegant solution together. Now that MPI support is available, these options will become less important and may be removed in future versions of `wcd`. (Let me know if you rely on them.)

In our labs we have about 150 machines on one NFS server. We wrote a Perl script that launched `wcd` on each of the machines, using the `--range` and `--dump` options to specify the workload that machine had to take on and to say where the results should go.

NFS is used for communication. Each process reads in the entire sequence file – obviously hideously expensive. To make what we did socially acceptable and to prevent our NFS server from falling over, the Perl script that launches the `wcd` processes delays for about 60s between launches which means that it takes over 2 hours to start all the `wcd` processes.

The auxiliary program, `combine` can be used to read in all final dump files to produce the overall clustering.

6 Testing

This chapter written by Peter van Heusden

This section was written with respect to an early version of `wcd` 0.3. The code and performance has changed significantly since that time.

`wcd` was compared against `d2_cluster` on a dual processor Pentium III system, with processors running at 1GHz and 1.5Gb RAM.

The test dataset was created out of 23300 ESTs and 315 mRNAs known to be associated with 27 genes on chromosome 22 that are known to be alternately spliced. The average length of the mRNAs was 4923 bases and of ESTs 633 bases.

Input sequences were masked for repeats and contamination using `cross_match`.

Three test runs were done, first with only ESTs, second with only mRNAs and finally with a dataset comprising all ESTs and mRNAs. Only a single CPU was used for processing, and run times were as follows:

ESTs only: `wcd` 1 hour 4 minutes (3870 seconds user cpu, 16 seconds system cpu) `d2_cluster` 1 hour 2 minutes (3704 seconds user cpu, 14 seconds system cpu)

mRNAs only: `wcd` 5 hours 16 minutes (19014 seconds user cpu, 24 seconds system cpu) `d2_cluster` 3 minutes 20 seconds (198 seconds user cpu, 0.7 seconds system cpu)

ESTs and mRNAs combined: `wcd` 5 hours 22 minutes (19353 seconds user cpu, 2.23 seconds system cpu) `d2_cluster` 1 hours 21 minutes (4905 seconds user cpu, 1 second system cpu)

As can be seen from these results, `wcd` currently is significantly slower than `d2_cluster` when dealing with mRNA data. `wcd`'s performance can be improved by increasing the number of words that two sequences must have in common before `wcd` will do a detailed comparison. This parameter is set with the `-H` flag. The price of this increase in performance will be decrease in sensitivity.

In terms of sensitivity, the following results show that `wcd` has comparable sensitivity to `d2_cluster` in finding similarities between EST and mRNA sequences.

For the dataset of EST sequences, the following clusters were found: `wcd` 125 clusters consisting of 12520 sequences. `d2_cluster` 129 clusters consisting of 12690 sequences.

A detailed comparison of the clustering results shows that `wcd` joined together 6 clusters into 3 in its results, whereas `d2_cluster` joined together 8 clusters into 2 in turn. In the results from `wcd`, 179 sequences were singletons that were in clusters in the `d2_cluster` results, whereas in the `d2_cluster` results, 9 sequences were singletons that were in clusters in the `wcd` results.

While these results suggest that `d2_cluster` is marginally more successful than `wcd` in assigning sequences to clusters, the difference between results is not significant (only 0.76% of sequences were singletons in `wcd` results but in a cluster in `d2_cluster`) results.

For the dataset of mRNA sequences, the following clusters were found: `wcd` 26 clusters consisting of 265 sequences. `d2_cluster` 26 clusters consisting of 270 sequences.

As in the EST results, `d2_cluster` assigned more sequences (5 or 1.58% of the dataset) to clusters than `wcd` did. Again, however, the results are not significantly different in terms of sensitivity.

For the dataset of all (EST combined with mRNA) sequences, the following clusters were found: **wcd** 83 clusters consisting of 12852 sequences. **d2_cluster** 82 clusters consisting of 13026 sequences.

As can be seen by comparing the combined dataset to that of ESTs, the addition of mRNAs to the dataset has the result of reducing fragmentation.

A detailed comparison of the clustering results shows that **wcd** joined together 4 clusters into 2 in its results, whereas **d2_cluster** joined together 9 clusters into 2 in turn. In the results from **wcd**, 181 sequences (0.76% of the dataset) were singletons that were in clusters in the **d2_cluster** results, whereas in the **d2_cluster** results, 7 sequences were singletons that were in clusters in the **wcd** results.

Again, the results show that **d2_cluster** is marginally more successful at assigning sequences to clusters than **wcd** is, but that overall the difference in results between the two programs is not significant.

Comment by SH

Testing of the difference in quality between the **d2_cluster** program and **wcd** is a little tricky. In principle, it is highly unlikely there is any real difference. Also note that **wcd** 0.4 is significantly faster than 0.3.

Two methodological points: cluster size is only a very rough measure of correctness; and the only valid comparison is with a known correct answer.

Research we have done using different distance measures has shown that the parameters used can be more important than which distance measures. **wcd** and **d2_cluster** have slightly different default parameters. Changing the parameters will change the results. If you use the right parameters, you will get good answers; if you don't you won't.

Moreover, changing the heuristics slightly can change the performance dramatically. Changing some of the heuristic parameters will speed up clustering by more than a factor of 2 with little impact of quality.

7 Acknowledgements and copyright

7.1 Acknowledgements

wcd was written after a very productive visit to the South African National Bioinformatics Institute (SANBI). SANBI introduced me to the problem and as we needed to have a d2 clustering program for our research, I wrote one. Many people helped but thanks particularly to Winston Hide for alpha and beta testing of the code as well as some very detailed biological analysis. Peter van Heusden was responsible for significant testing as well as integrating into StackPack.

Ramon Nogueria was responsible for the development of the acd files and wrapper for the EMBOSS implementation. Richard Starfield worked on the Pthreads implementation.

Zsuzsanna Lipták, now at the University of Bielefeld introduced the maths and the algorithms to me, and made sure I knew what I was doing. She also helped me refine the algorithmic innovations in this version of d2 cluster and made several useful suggestions.

Anton Bergheim and Pravesh Ranchod at Wits also gave valuable feedback.

Research grants from the National Bioinformatics Network and the National Research Foundation supported this work.

All the above helped with alpha testing.

7.2 Copyright

The program copyright details are:

Copyright © Scott Hazelhurst 2003–2007
 School of Computer Science,
 University of the Witwatersrand,
 Johannesburg
 Private Bag 3, 2050 Wits
 South Africa
 scott@cs.wits.ac.za

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public Licence as published by the Free Software Foundation; either version 2 of the Licence, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public Licence for more details.

Version date: 20 March 2008 \$Id: wcd.texi,v 0.4.5.1 2008/03/20 13:22:58
 scott Exp scott \$

This documentation is distributed under the GNU Free Documentation Licence.

Concept Index

A

auxiliary programs 8, 22

C

clone 11
cluster table 21
constraint 17, 20

D

distance function choice 2, 10, 16
distance function parameters 16
dump 22

E

extended format of cluster table 21

F

fasta2sary 18
format, output 21

H

heuristic 17, 28

I

installing 5

M

merging 13
MPI 8, 10, 24
MPI, installation 6

O

orientation 21, 27
output 16

P

parallelisation 10, 23, 24
Pthreads 24
PTHREADS 8, 10
Pthreads, installation 6

R

range 18
recluster, de novo 12
recluster, from a coarser cluster 19
reverse complement 22

S

shared memory processor 24
SMP 24
stackPack 25
stackPACK 8
suffix 18
suffix, using fasta2sary 23

T

testing 32
threshold 17

W

window length 17
word length 17