Overlap Detector Module Manual

Art. Delcher

8 February, 2000 Version 1.7

1. Overview

The Overlap Detector Module of the Assembly Subsystem has two main functions: maintain a database of fragment information, and detect which pairs of fragments contain matching data. The purpose of the fragment database is to store previously received fragments, so that their matches with newer fragments can be computed.

2. Interface

2.1. Overlapper

The Overlap Detector Module is invoked by the following command:

```
overlap  [options]  <database-dir>  [input-filename]
```

The database directory is the path to a directory that holds the fragment database. Currently this directory will hold the following binary files:

- ♦ `db.dst` The distance records describing distances between fragments
- ♦ `db.frg` The fixed-length info for each fragment
- ♦ `db.seq` The sequence and quality data for the fragments
- ♦ `db.src` The source comments associated with fragments

The input-filename is used to designate both the input and output message streams. Input messages generally come from either a `.inp` file (as generated by the gatekeeper) or from a `.urc` file (as generated by the screener). Output messages are sent to a file with the same name as the input file except that the extension is changed to `.ovl` .

2.1.1 Options

With no options, the Overlap Detector Module will attempt to create a new fragment database in the specified directory. If the directory already exists (whether it's empty or not), the program will print an error message and exit.

The `-a` option directs the Overlap Detector Module to add the new fragment data to the existing fragment database in the specified directory. If the specified directory does not exist, a new one is created. If the directory exists, but it does not contain the database files, an error occurs.

The `-c` option directs the Overlap Detector Module to run in contig mode. In contig mode the overlapper looks for overlaps between fragments and contigs. The fragments are taken from the specified fragment store and the contigs come either from the input file or from a separate contig store. This option is used in the Contig Assembler pipeline.

The `-f` option directs the Overlap Detector Module to create a new fragment database in the specified directory. If a database already exists in that directory, it is first deleted.

The `-h` option directs the Overlap Detector Module to build its hash table from fragments taken from the fragment store (or contig store in contig mode). With this option no input file should be specified, and an output filename must be specified with the `-o` option. The format is:

`-h` <*first-frag-iid*>–<*last-frag-iid*>

The last-frag-iid can be omitted in which case all fragments from the first-frag-iid to the end of the store are used.

The `-m` option directs the Overlap Detector Module to produce multiple overlaps for a single pair of (oriented) fragments if they exist. This is the opposite of the `-u` option. For example, if the front of fragment *A* overlaps the back of fragment *B*, and separately the back of fragment *A* overlaps the front of fragment *B*, the `-m` option will output two separate overlap messages, whereas the `-u` option will output only one (with a range of ahangs to indicate the variation).

The `-n` option directs the Overlap Detector Module not to compute any overlaps. Thus, there are not any overlap messages in the `.ovl` file, just messages passed through from the input. The main purpose is to create or add to the fragment database.

The `-o` option specifies the name of the output file to be generated. This option is required if the `-h` option is specified.

The `-P` option directs the Overlap Detector Module to produce protoIO (*i.e.*, human-readable) format output. Without this flag, the output produced is binary format.

The `-r` option directs the Overlap Detector Module to use only fragments from the specified range in the fragment store. These are the fragments that will be streamed against the fragments in the hash table. In general this option is used in conjunction with the –h option. The format is:

`-r` <*first-frag-iid*>–<*last-frag-iid*>

The last-frag-iid can be omitted in which case all fragments from the first-frag-iid to the end of the store are used.

The `-t` option specifies the number of parallel threads for the Overlap Detector Module to use. Without this flag, the number of threads is the value of the `#define` constant `DEFAULT_NUM_PTHREADS` (currently set to 4). Either `-t 0` or `-t 1` will cause the overlapper to use a single thread. To compile without including the pthread libraries set

```
#define  USE_THREADS  0
```

in `AS_OVL_overlap.h`.

The `-u` option directs the Overlap Detector Module to produce only one overlap for a single pair of (oriented). This is the opposite of the `-m` option (see above). This is the default value currently.

2.1.2 Examples

```
overlap -P -t2 -f test.Store test.urc
```

Create a new fragment store in `test.Store` using the fragments in `test.urc`; and find all overlaps among these fragments. Output will go to file `test.ovl`.

The `-P` option means the output will be in ASCII format and the `-t2` option means that 2 pthreads will be used in computing overlaps.

```
overlap -o out -a test.Store new.urc
```

Append to the existing fragment store in `test.Store` the fragments in `test.urc` ; and find all overlaps between two new fragments and between and old and a new fragment. Output will go to file `out` .

```
overlap -n -f test.Store new.urc
```

Create a new fragment store in `test.Store` containing the fragments in `test.urc` , but generate no overlaps. The output file `test.ovl` will contain only fragment, link and distance messages.

```
overlap -h 2001-3000 -r 1-3000 -o part.ovl test.Store
```

Find all overlaps between fragments numbered 1-3000 and fragments numbered 2001-3000. Output goes to file `part.ovl` and will contain only overlap messages.

2.2.Script Generator

There is a separate program to generate a script of LSF commands, where each command is a separate overlapper job. This program is invoked as follows:

```
    make-ovl-script  [options]  <database-dir>  input-filename
```

The database directory and the input filename are the same as for the `overlap` command. The options are the same as for the `overlap` command except for the following:

- ♦ The `-h` and `-r` options are not used currently. (I anticipate adding `-h` at least.)
- ♦ The `-s` option is used to specify the name of the output script generated by `make-ovl-script` . Without this option the name of the scriptfile generated is the input filename with the extension changed to `.script` .
- ♦ The `-o` option specifies the prefix of the output overlap (`.ovl`) files that will be generated by the LSF jobs. A sequence number is appended to each.

2.2.1 Examples

```
make-ovl-script -P -t2 -f test.Store test.urc
```

Generate a script file named `test.script` with LSF commands to first create a new fragment store in `test.Store` using the fragments in `test.urc` ; and then find all overlaps among these fragments. Output files from the LSF jobs will be `test.ovl.0` , `test.ovl.1` , …. The `-P` and `-t2` options are included in the overlap commands in the LSF jobs.

```
make-ovl-script -o out -s lsf -a test.Store test.urc
```

Generate a script file named `lsf.script` with LSF commands to first append the fragments in `test.urc` to the existing fragment store in `test.Store` ; and then find all overlaps between fragments that were in the store and those just added and between two new fragments. Output overlap files from the LSF jobs will be `out.0`, `out.1`, ...

3. Design

See `cds/AS/doc/Assembler/Designs/OVLModule.rtf` (the design document) for a description of the algorithms used. Also see the documentation for the fragment database in `cds/AS/doc/Assembler/SoftwareModules/FragStore.rtf`.

4. Memory Usage

This module is entirely self-contained—it does not use or supply memory to other modules.

The dominant memory usage is for the hash table that indexes "new" fragments. The size of that table is fixed and determined by the two constants: `HASH_MASK_BITS` and `ENTRIES_PER_BUCKET`. The total size in bytes is:

$$(5 \times \texttt{ENTRIES\_PER\_BUCKET} + 8) \times 2^{\texttt{HASH\_MASK\_BITS}}$$

For a typical batch of 100,000 new fragments of average length around 500, `HASH_MASK_BITS` would be 23 and `ENTRIES_PER_BUCKET` would be 12, giving a total size of about 570 megabytes.

Additional memory is used to store the fragments themselves. In the current version of the program, the fragments themselves are stored twice (once in the memory-resident fragment store and once in the structure attached to the hash table) costing 2 bytes per fragment base. Quality values currently are not used and hence not stored. There is also a 4-byte structure used to track multiple occurrences of the same kmer. For the typical batch of 100,000 fragments of length 500 this totals approximately 300 megabytes.

There is also memory allocated for each fragment (no matter what its length is) to track its length, position in the index, branch points, etc. This totals about 32 bytes per fragment, or only 3.2 megabytes for a typical 100,000-fragment batch.

All other memory is devoted to data structures to store overlap information between a single "old" fragment and the fragments in the hash index. This is variable depending on the actual number of overlaps involved. In the worst case of a truly ubiquitous, periodic repeat region, it might represent thousands of overlaps, but would use still use no more than a few megabytes of memory.

Running `ps` on the typical run shows total memory use of around 1.2 gigabytes (data and instructions).

The maximum number of fragments that will be inserted into the hash table is `MAX_HASH_STRINGS`, *i.e.*, the batch size, typically set at 100,000. The program checks that $\texttt{MAX\_HASH\_STRINGS} \times \texttt{HASH\_EXPANSION\_FACTOR}$ does not exceed the value of $\texttt{ENTRIES\_PER\_BUCKET} \times 2^{\texttt{HASH\_MASK\_BITS}}$ and fails with an error message if this is not the case.

5. Other Performance Factors

The quality and number of overlaps found is affected by several parameters. For an overlap between two fragments to be reported, the number of errors (*i.e.*, differences in bases) must be no greater than `ERROR_RATE` times the length of the overlap region between the fragments. That overlap region must be at least `MIN_OLAP_LEN` bases long. In addition, the overlap region must contain an exact match of at least `WINDOW_SIZE` bases.

Increasing the `ERROR_RATE`, decreasing the `MIN_OLAP_LEN`, decreasing the `WINDOW_SIZE`, or any combination of these will make the overlapper more sensitive, *i.e.*, find more overlaps. This will also make the overlapper run substantially slower. The default settings for these parameters are `ERROR_RATE` = 0.06, `MIN_OLAP_LEN` = 40, and `WINDOW_SIZE` = 20. With these values, building a hash-table for 100,000 *C. elegans* fragments and computing the overlaps among them takes approximately 5 minutes on a single DEC 8400 with 524MHz EV6 processor.

The nature of the input fragments also strongly influences overlapper performance. The more similarity in sequence among the fragments (as in repeat regions, for example) the slower the overlapper runs.

6. Parallelization

The current version of the Overlap Detector Module runs in parallel using pthreads. Building the hash table for "new" fragments is done sequentially, but then computing overlaps between pairs of new fragments and between "old" and new fragments is processed by multiple threads. Each thread simply picks up the next available subrange of fragments and finds overlaps between that subrange and the new fragments in the hash table. The number of fragments in a subrange is specified by the constant `MAX_FRAGS_PER_THREAD`.

Memory for the hash table and for the new fragments is shared among all the threads. Each thread keeps a separate, private copy of the data structures it uses to track matches between its fragments and those in the hash table. Memory for these data structures is dynamically allocated.

7. Limitations

Internal fragment ID's must fit in 32 bits.

8. Status

- ♦ No use is made of quality data. The alignment reported is one with the minimum number of insert/delete/mismatch errors.
- ♦ Any region noted by the UR dectector with the flag `AS_OVL_HEED_RPT` is not added to the hash table index. Specifically, any k-mer that encroaches on such a region by more than `WINDOW_SCREEN_OLAP` bases, is not added to the hash table. To be reported as an overlap, a matching region must contain at least

MIN_OLAP_OUTSIDE_SCREEN bases that are not in a region designated by the UR detector.

- Microsat/polymorphic overlaps are not done.
- No special handling of periodic (or nearly periodic) overlap strings is done.

9. Architecture and Dependencies

The Overlap Detector Module consists of the following files:

AS_OVL_overlap.c	The main program and overlap routines.
AS_OVL_driver.c	The I/O routines and control of the overlap routines.
AS_OVL_overlap.h	The header file for the above 2 files.
AS_OVL_delcher.c	Common includes and functions.
AS_OVL_delcher.h	The header file for the above.
MakeScriptOVL.c	The program to generate a script file of LSF commands to submit separate overlapper jobs.
MakeScriptOVL.h	The header file for the above.

It depends on the following modules:

AS_PER_ReadStruct
AS_PER_genericStore
AS_PER_fragStore
AS_PER_distStore
AS_MSG_pmesg

AUTHORS

Art. Delcher:	
Created:	28 Jan 99
	Moved material out of the design document. Changes due to GateKeeper.
Last Revised:	6 Apr 99
	Description of parallel features.