velvetReadMatcher.pl

The velvetReadMatcher.pl program extracts reads, contigs and tiling information from a velvet assembly. By linking the Velvet output back into the Illumina namespace, we can assign quality scoring to the contig, and better enable contig editors to differentiate between misjoins from sequencing errors and SNPs.

The program first rummages through the velvet Sequences file and the input files to get the more informative Illumina GA Pipeline names, and then grabs the quality string for the reads from the Gerald folder. These are put into the database. Next, it reads the contig and tiling information from the afg file. Finally it computes the contig quality values.

Quick (sic) Start.

1. Edit the first line of the script to point to your perl installation and make the script executable.
2. If you are using a database other than Postgres, edit line 99 to reflect the proper DBI module. You will likely also need to edit or comment out line 100 depending upon the string interpolation behavior of the database. Finally, you will want to check the store and the \*SQL subroutines to make sure the SQL corresponds to your database’s syntax (got to love developers who regard the SQL standards as simple suggestions…). See the technical note below for more information.
3. Set up a target database. The schema for the database is given below.
4. Run the samples through the Illumina GA pipeline to produce s\_\*\_sequence.txt files. Typically, this means you will want to run GERALD with ANALYSIS sequence or ANALYSIS sequence\_pairs. Note the format of the s\_\*\_sequence.txt files needs to be fastq.
5. Extract the reads from s\_\*\_sequence.txt into fasta files for input into velvet. There needs to be a single fasta file for each lane of data, and the fasta defline needs to match the defline in the s\_\*\_sequence.txt file. See below for more information.
6. Run velveth and velvetg. The final iteration of velvetg must have read tracking and afg file production enabled.
7. Run the velvetReadMatcher script with the appropriate directory switches set.

Detailed Instructions

The script relies on lots of files. These files are produced by the GA Pipeline, velvet, and you. Thus we need to tell the program where these files are via a set of parameters. All the parameters are optional, and the program will run using a set of default values for all the file directories. But I’m pretty sure you won't like them much, so you will want to change them.

The first parameter tells the program where the Velvet files are:

-v <velvet directory>

This is path to the velvet directory you specified in the velveth command. It contains the Log, Sequence, and velvet\_asm.afg files. The default path is the current directory.

Along with the Velvet output files, we need to know where the original velvet input files are. At the start of the Velvet analysis, the velveth program creates a single fasta file from the original input files. Unfortunately, during this process the Illumina sequence names are replaced with slightly less informative names. The new Velvet names are then propagated into the final assembly (where they are given yet another name). Thus the first step in the process of computing contig quality is to reunite the reads with their original names. Fortunately, the order is sequential with the order as supplied to velveth. We parse the input file names and their order from the velvet log file, but we need to get the real path:

-f <input fasta directory>

The default path for the input files is the current directory.

To get the read qualities, we need to look back at the original fastq files. The Illumina defline contains almost enough information to get to the source Gerald directory, if we know the path to the Illumina run folder directory.

-r <run folder root>

This is the same path you set on the SCS & IPAR computers: which is the directory where the experiment folders are placed. The default path is the current directory (told you you wouldn’t like the defaults). Of course, this prediction isn’t perfect – if there are multiple analyses the program will ask you to help decide which folders to use.

As the reads are correlated and the assembly parsed, the data is stored into a database. Two parameters are used to specify the database

-g <assembly database>

-d <db user name>

The –g option specifies the database to load the current assembly into. The default is 'velvet'. This will work if you only have a single organism being sequenced; otherwise you probably want to use your organism name. The –d specifies the user name to connect to the database as. This user must have write permission to the database. The default user is 'postgres', which will work fine if you know the root password.

Finally, the contig sequence and quality strings can be a bit large. Since the syntax to load large objects varies between databases, code portability is increased if we store the files externally.

-o <contig output directory>

The contig output directory place where you want the fasta files and quality files to be stored. The default directory is "./contig\_dir". If the directory does not exist, the program will attempt to create it.

For example, I ran the following set of commands on my computer to generate the quality files for our genome sequence:

> cd /Volumes/sequence/PC

> [copy and create my fasta files in this directory]

> velveth v\_all\_31 31 -fasta -short i76.fa -shortPaired i125.fa \

-shortPaired2 i376\_1.fa i376\_2.fa i376\_3.fa i376\_4.fa i376\_5.fa

> velvetg v\_all\_31/ -exp\_cov 20.6 -ins\_length 125 -ins\_length2 400 \

-min\_contig\_lgth 100 -read\_trkg yes -amos\_file yes

> velvetReadMatcher.pl -v v\_all\_31 -r /Volumes/illdata2 \

-o ~/Seq/PC\_Contigs -g PC

First, note from the velveth command that of the seven files, six were paired-end runs. The program detects the pairing, and deconvolutes them. Also, because I did not specify them, the fasta file directory defaulted to the current directory, which was fine, and the database user name defaulted to ‘postgres’, which was also fine as I new the root login.

Technical notes

Database Schema

The database should contain the following tables.

--

-- Name: contig; Type: TABLE;

--

CREATE TABLE contig (

aid integer,

vid character varying(50),

sequence text,

quality text,

length int

);

--

-- Name: read; Type: TABLE;

--

CREATE TABLE read (

aid integer,

vid character varying(50),

gaid character varying(50),

sequence text,

quality text,

mate integer

);

--

-- Name: scaffold; Type: TABLE;

--

CREATE TABLE scaffold (

aid integer primary key,

vid character varying(50),

contig integer,

origin integer,

clr\_start integer,

clr\_stop integer

);

--

-- Name: tiling; Type: --

--

CREATE TABLE tiling (

read integer,

contig integer,

origin integer,

clr\_start integer,

clr\_stop integer

);

The tiling table links the read table to the contig table: tiling.read = contig.aid and tiling.contig = contig.aid. The read.mate field indicates pairing, it should point to read.aid.

Indices and constraints greatly improve database performance. For example, during data loading we createthe following three indices to speed up the quality calculations:

create index contig\_idx on tiling(contig);

create index contig\_origin\_idx on tiling(contig, origin);

create index aid\_idx on read(aid);

The latter index is especially important, as it affects a join in the query.

Note that the reads are loaded first, then the assembly file. The contigs and tiling data is in a block format, and we add the contig first, followed by the individual tilings. Then we make a final pass to link the mate pairs. Theoretically, given this order there would be no need to drop constraints before loading.

Data Loading

Data is loaded via INSERT and UPDATE commands. There is no error or integrity checking. With the simple schema, if you run the command twice you will duplicate rows. If for any reason the script is interrupted, you need to truncate the tables and start again.

Obviously, setting constraints on the fields will improve this situation. You can define any or all of the contig and read fields containing “id” as “unique not null.” (The aid field is the amos id, vid is the velvet id, and gaid is the GA id; all are unique). You will then be able to rerun the program without worrying about duplicate entries.

Note that if you are doing inserts and/or updates after the initial data load, you will want to drop any indices on the tables.

Performance

We’re looking at millions of reads, one by one. Things are not going to zip along. Inserting the reads is fairly quick (~30,000 reads a minute), as is parsing the contigs. Things bog down quite a bit when we start querying the database to get the read qualities back. As noted above, indices will greatly improve performance: there is still a noticeable pause with contigs that contain more than 500 or so reads, but the system is appreciably faster. Also, if the database server is on a multi-use machine (such as your personal workstation), you will notice a definite performance hit as the database inactivates sections of memory to serve as a query cache.

Caveats

From the perldocs:

1. The software requires more hardening to guard against misuse. Specifically, I assume the database and tables have been set up properly, and there will be no issues inserting data. Obviously this is an extremely optimistic assumption.

2. The base-wise quality values for the contigs are created by averaging the individual read qualities for each position. This might not be the most mathematically accurate method. From past experience, I know I could use the Statistics::Descriptive perl module and a mildly more complex step to provide other potential methods. Alternatively, if this becomes a more stable product I could perform the quality calculation using an SQL function triggered by the insert of each tile entry. But the current version works fine for my present use (see below).

3. I would also like to store the sequences & quality strings in the DB rather than files. The original draft of the program was strictly db-driven, until I learned that while an INSERT using very long text fields works fine, the UPDATE command truncates the command string after a certain point. This occurred right in the middle of the quality string, producing an error for the update. I was not able to immediately find any guidance on the error, so I resorted to text files to accomplish my task (again, see below) as it was clear that fixing the error would take quite some work.

4. For real masochism, the quality step could be threaded and run parallel. DB connection would have to be record locking. Ick.

Honesty Disclaimer

When will I fix these issues? Maybe never. This program was written to support a specific sequencing project. Since we may never again do such a project, I may never revisit this program.

I spent the development time to write it in order to get to an answer and to fill in the gaps and shortcomings associated with the current de novo assembly software. To whit: velvet has rather rudimentary read tracking capabilities (especially with regard to going back to the original reads), the read quality information is lost during assembly, and even a modest afg file is difficult to view and manipulate given nothing in AMOS is geared to deal with the volume of short reads needed for even a small eukaryotic genome. Loading the data into a database was the quickest and easiest way to get to the data munging steps (but let’s save the database vs. flat file rant for some other time…) and more specifically the only way to get to the answers I wanted.

Now, I do think it would be fun to engineer a system that addresses the shortcomings listed above. And maybe I will someday, when someone gives me the money to do so. But until then, take the program for what it is – freeware code produced for my own idiosyncratic purposes – and don’t expect a whole lot of flexibility or finished prettiness.