Meraculous
De Novo Assembly of the *Ariolimax dolichophallus* Genome

Charles Cole, Jake Houser, Kyle McGovern, and Jennie Richardson

Meraculous Assembler

- Published by the US Department of Energy Joint Genome Institute
  - Managed by the Lawrence Berkeley National Laboratory
- Originally designed for exclusively for haploid assembly
- Current version supports diploid assembly as well
- Stated Advantages
  - Multi-threaded and parallelized computation
  - Lightweight hash structure resulting in low RAM footprint
  - No error correction step for faster processing

Meraculous Overview

- Designed for deep paired-end short reads (e.g., Illumina)
- Efficient and conservative traversal of subgraphs of the deBruijn graph (like ALLPATHS, ABySS, and SOAPdenovo)
  - Unique high quality extensions in the dataset
  - Avoids explicit error correction step used in other assemblers, instead relying on base quality scores
- Novel memory-efficient hashing scheme
- Contigs are ordered and oriented using paired-reads
- Gaps closed using paired-end placements

Meraculous Modules

- Selects kmer set
- Produces set of maximal linear sub-paths of the deBruijn graph
- Aligns reads to assembly in order to identify useful read-pair information
- Uses paired-reads and splinting singletons to produce a scaffolding by “ordering and orienting” a set of contigs
- Closes gaps

Meraculous Publication

- Tested on *Pichia stipitis* genome assembly
  - 15.4 megabase genome
  - 75 bp paired Illumina reads
  - 425x coverage of genome
- Results
  - 95% of genome recovered
  - N50 = 101kb

Meraculous Algorithm

1. Count the number of occurrences for each k-mer in the data set
2. Remove all k-mers occurring less frequently than a user-defined threshold
3. For each k-mer, count the number of high-quality single-base extensions

Meraculous Algorithm

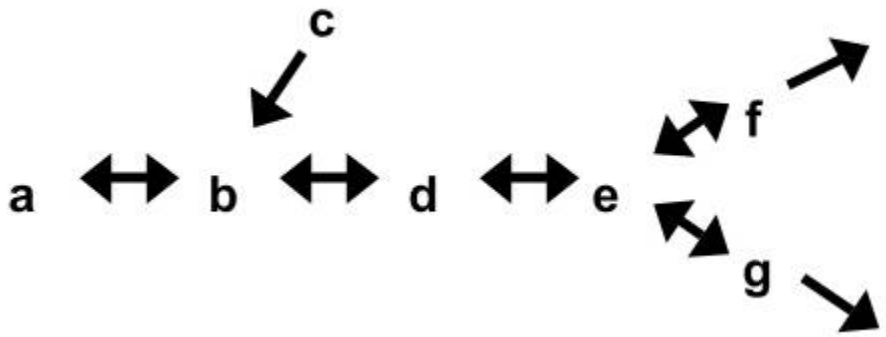
4. Classify the 5’ and 3’ end of each k-mer
    a. *X* if there are no high-quality extensions
    b. *U* if there is exactly one high-quality extension
    c. *F* if there is more than one high-quality extension
5. Store the extensions of k-mers with a *U-U* classification in a hash
6. Remove non-reciprocal linkages between k-mers

Meraculous Algorithm

7. Select k-mers at random as “seeds” and extend outwards to produce contigs
8. Align all reads to these contigs via BLAST
9. Contigs are assembled into scaffolds using paired-end data
10. Unaligned reads are searched as potential gap-closers using mate-pair data

Meraculous deBruijn Graph

a [X]ACTGCTG[U]
b [U]CTGCTGC[U]
c [U]TGCTGCa[X]
d [U]TGCTGCT[U]
e [U]GCTGCTT[F]
f [U]CTGCTTA[U]
g [U]CTGCTTC[U]



Meraculous Hash

- In genome assembly, most of the memory required to store a hash is in the keys
- Each value (the extensions of a k-mer) is only 4 bits
- The Meraculous *U-U* hash of the human genome requires only 8Gb of memory
- Relies on all keys being known in advance and static
- Recursive hashing method which requires a series of independent hash functions defined
- Stores hash keys in files determining which hash function to use

Meraculous Hash

1. Initialize hash and write all keys to a file (F_d)
2. Evaluate all keys with hash function h_d, and keep track of keys which collide
3. Write all colliding keys to F_{d+1}, increment d
4. Repeat 2, 3 until no collisions
5. Assign hash functions based on file which key resides in

Meraculous Hash

- Hash functions are defined recursively
  - $H_1 = h_1(key)$
  - $H_i = H_{i-1} + h_i(key)$
- The i subscript corresponds to the hash depth (d) from the initialization step
- Hashing lookup does not depend on genome size, but cannot be modified after initialization

Meraculous Limitations

- Relies on an abundance of high quality data in order to avoid the error correction step
- Initial release did not support polyploid genomes due to only allowing for linear subgraphs of the deBruijn graph
- Low memory footprint, but writes all k-mer keys to the file system

User Experience

- Overview:
  - The program requires an array of dependencies including a recent version of GCC,Perl,CMake, Boost, etc…
  - The program is built through a shell script which calls a CMake script which then calls Make.
  - Most of the high level stuff is done through perl scripts
  - We tested the program on the small data set the program came with and the contigs looked like contigs.
  - We are currently trying to assemble the genome.

Installation

- Getting all the dependencies together took a while, but this appears to be a problem inherent with working on computers you don’t own.
- There was one non-standard perl module you need(Log::Log4perl)
- Some of the perl scripts contain errors so when you run test_install.sh it crashes, but these aren’t hard to fix.

Running Meraculous

- You run meraculous by executing the run_meraculous.sh scripts along with the configuration file
- The configuration file should contain info on where your data is and what form it comes in. It also contains information on how much memory and how many CPUs you want per task.

Running Meraculous

- It creates a timestamped folder which includes directories containing the results of each step, the log files for each step as well as a set of executables which you can use to resume, restart or suspend the current run.
- If a run failed you can check the error log, correct the problem and then resume the run.
- It looks like this feature mostly works.
- The logs are fairly informative.

Overall Impression

- It’s kind of buggy but it’s fairly straightforward to figure out what went wrong and it’s nothing that can’t be fixed with a basic understanding of Perl.
- It handles all of the directory creation for you so organizing the data isn’t a problem
- The logs tell you exactly what commands it ran which is incredibly useful.

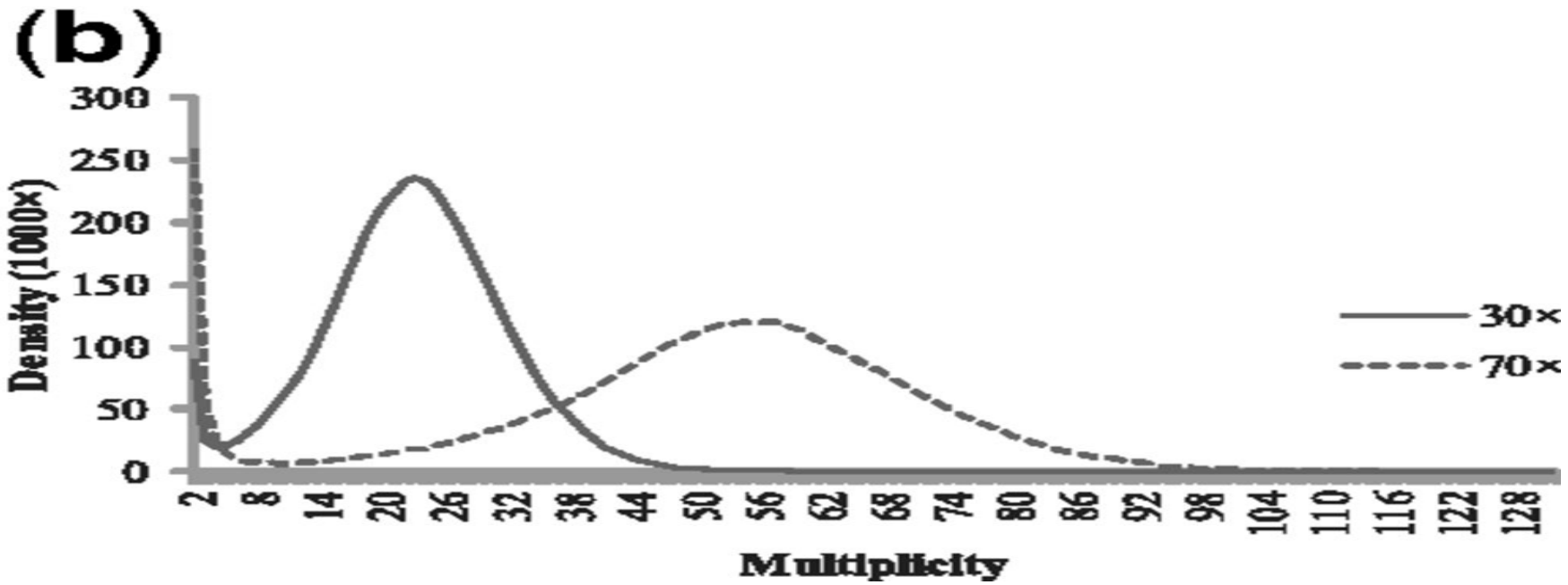
Error Correction and Meraculous

- Meraculous requires error correction as well as adapter removal as these steps are not completed by the program. Trimming however, is unnecessary.
- Skips this error/trimming step by using Kmer coverage and base quality scores: it ignores low quality bases on 3’ end.
- High error rates (insertions, etc.) clog up assembler, need to be removed.
- Kmer size chosen directly affects assembly quality.

KmerGenie

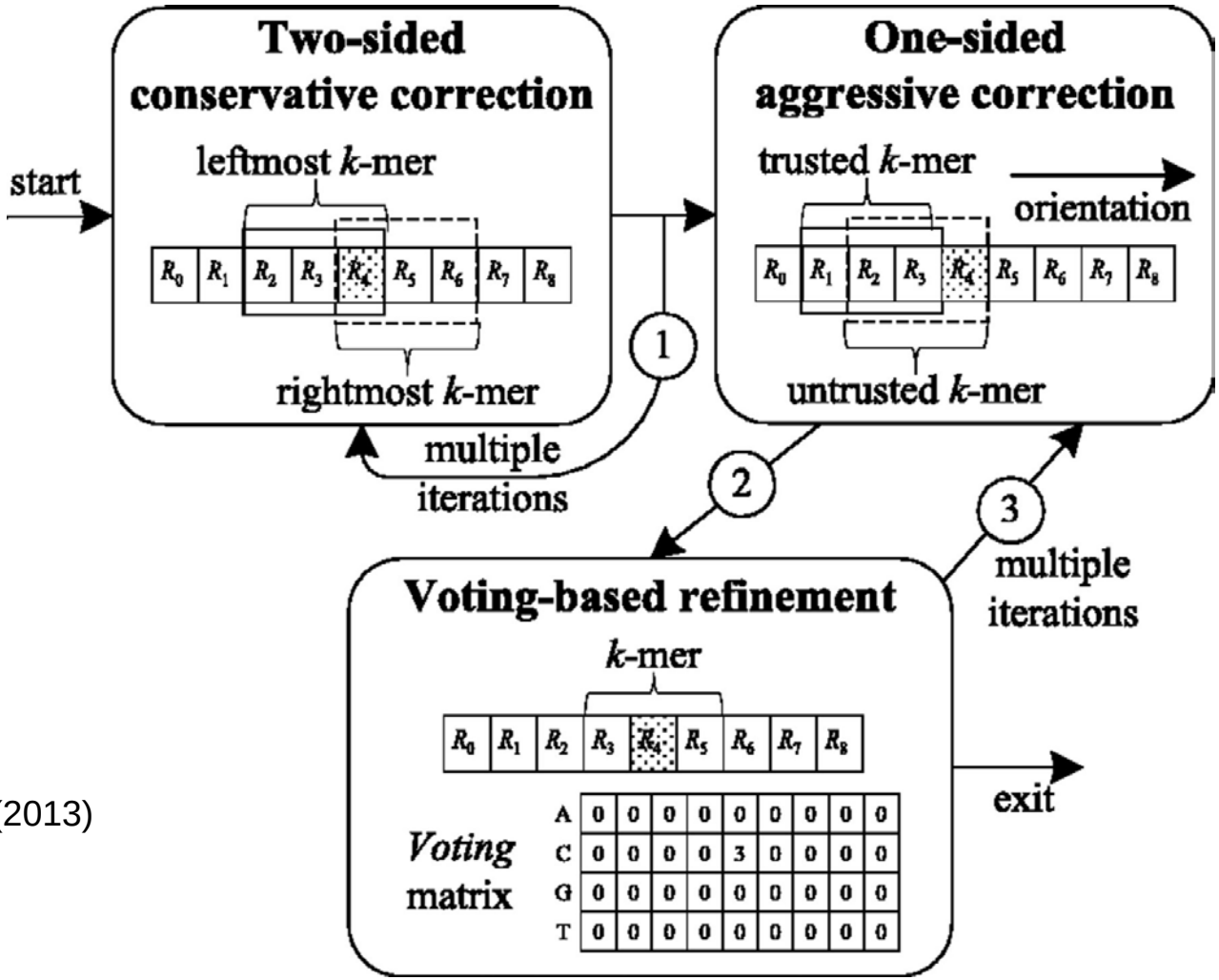
- Meraculous requires an optimal kmer size for runs, a Kmer size too short results in too few unique Kmers
- A Kmer size too large introduces too many sequencing errors.
- KmerGenie is a program that gives an optimal assembly kmer size by generating abundance histograms for many values of k (kmer size) and using this info to predict a best k value.
- Makefile written, problems running programs on cluster.

Musket



Liu et. al
(2013)

Musket



Liu et. al (2013)

Previous Analysis

- Quality Assessment
  - FastQC
    - GC content ~41%
    - Adapter contamination
      - First ~15 bases of reads have lower quality score
      - Overrepresentation of k-mers in the first ~10 bases
  - Preqc
    - Genome size ~2.3Gb at ~20x coverage
    - Low heterozygosity
    - High repeat content
    - Low sequencing error rate
    - Low duplication levels
    - High quality reads

- Adapter Trimming (Skewer)

Results

Pending

Future Direction

- Improve Assembly
  - Run GapCloser
  - Resolve mis-assemblies with REAPR
  - Assess using CEGMA

- Re-run assembler after:
  - Improving k-mer size estimate
  - Using error correction (BLESS, Musket, or SGA)
  - Collecting additional HiSeq data

Future Direction

- Combine with scaffold assembly
  - Mate-pair library
  - MinION data
- Make meta-assembly from the best assemblies
- Annotate genome
- Publish paper and thank contributors!

References

1. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3158087/
2. http://jgi.doe.gov/data-and-tools/meraculous/