

User Manual

HaploMerger2

Release 20161205, built Dec 05, 2016

Manual version: v3.6

Author and maintainer: Shengfeng Huang

(hshengf2@mail.sysu.edu.cn)

-- REMINDER --

**If your raw diploid assembly has >2000 scaffold sequences,
make sure to lift your system's openable file handle limit by
invocating "ulimit -n 655350" before running HM2!**

--more info in the manual--

Table of contents

| | |
|---|-----------|
| Liability and disclaimer | 3 |
| Citations | 3 |
| Introduction | 3 |
| Limitations | 4 |
| Resources consumption | 4 |
| Installation | 6 |
| Understanding the HaploMerger2 pipeline | 7 |
| Quick Start – testing HM2 with the example 1 and 2 | 9 |
| Start your own project | 11 |
| Details of Operations – a walk-through with the example 3 | 12 |
| Step 0: preparation..... | 12 |
| Step 1: run hm.batchA1-3 to remove major misjoins from the diploid assembly | 13 |
| Step 2: run hm.batchB1-5 to create the haploid assemblies | 15 |
| Step 3: run hm.batchC1-2 to further scaffold the haploid assemblies..... | 18 |
| Step 4: run hm.batchD1-3 to remove tandem errors from the haploid assembly | 18 |
| Step 5: run hm.batchE1 to try to fill some gaps in the haploid assembly | 19 |
| Advanced Topics..... | 21 |
| 1. How to lift the openable file handle limit? Could it cause a problem to the system?.. | 21 |
| 2. How to soft-mask the raw diploid genome assembly? | 21 |
| 3. How to speed up HaploMerger2? | 21 |
| 4. How to increase alignment specificity? | 22 |
| 5. How to increase alignment sensitivity? | 22 |
| 6. How to remove allele redundancy as much as possible?..... | 23 |
| 7. How to avoid removing non-redundant sequences?..... | 23 |
| 8. How to avoid falsely connecting two scaffolds? | 23 |
| 9. Can I use HaploMerger on polymorphic multi-ploid assemblies?..... | 24 |
| 10. Can I use HaploMerger on meta-genomic assemblies? | 24 |
| Other Tools..... | 25 |
| 1. lastz_D_Wrapper.pl | 25 |
| 2. count_scf_ctg | 25 |
| 3. countN50_from_agp.pl | 25 |

Liability and disclaimer

HaploMerger2 is free for academic use. This software comes without any warranty. Users should use it at their own risk and we cannot guarantee that it will fulfill particular purposes or needs.

Citations

An introduction to HaploMerger2

Huang, S. and Kang, M. HaploMerger2: rebuilding both haploid sub-assemblies from a heterozygous animal diploid genome assembly. *Submitted*.

An real application of HaploMerger

Huang, S., Chen, Z., Yan, X., Yu, T., Huang, G., Yan, Q., Pontarotti, P.A., Zhao, H., Li, J., Yang, P., *et al.* [Decelerated genome evolution in modern vertebrates revealed by analysis of multiple lancelet genomes](#). *Nat Commun* 5, 5896. (2014)

The original paper to describe the algorithms of HaploMerger

Huang, S., Chen, Z., Huang, G., Yu, T., Yang, P., Li, J., Fu, Y., Yuan, S., Chen, S., and Xu, A. [HaploMerger: reconstructing allelic relationships for polymorphic diploid genome assemblies](#). *Genome Res* 22, 1581-1588. (2012)

Introduction

HaploMerger2 is an automated pipeline designed to facilitate the construction of a highly polymorphic diploid genome assembly. It can be used to:

1. re-construct the allelic relationships within a diploid assembly;
2. detect and correct mis-joins in a diploid assembly;
3. reconstruct two haploid assemblies for a diploid assembly: the reference haploid assembly containing the primary alleles, and the alternative haploid assembly collecting all other alleles not included in the reference assembly;
4. further scaffold a haploid assembly (by using third-party software) to achieve better continuity;
5. detect and remove tandem mis-assemblies from haploid assemblies;
6. close some N-gaps in haploid assemblies (by using third-party software) to achieve better contiguity;

HaploMerger2 also provides tools for post-assembly genome analysis, as well as easy-to-use wrappers for sensitive Lastz-chainNet whole genome alignments, scaffolder SSPACE 3.0 and GapCloser.

Limitations

- HaploMerger2 is not a *de novo* genome assembler or scaffolder, and is not an annotation pipeline.
- HaploMerger2 has only tested on polymorphic diploid genome assemblies.
- HaploMerger2 is not suitable to work on the assemblies with a scaffold N50 size <50Kb; it barely works when the scaffold 100Kb>N50>50Kb; it works better when the scaffold N50>100Kb; and it works much better when scaffold N50>150Kb.
- The raw diploid genome assembly should be pre-soft-masked by using winMasker or/and repeatMasker; otherwise the program may be kept running for months.
- HM2 is sensitive to illegal characters and sequences. We strongly recommend cleaning up the fasta sequences of the input genome assembly before running HM2 by using faDnaPolishing.pl from HM2:

```
gunzip -c genome.fa.gz | ./bin/faDnaPolishing.pl --legalizing \
--maskShortPortion=1 --noLeadingN --removeShortSeq=1 >genome_cleaned.fa.gz
```
- Currently, HaploMerger2 only provides compiled Lastz and chainNet executables for CentOS 5/6 (redhat 5/6) and ubuntu10-64bit. For other systems or environments, you may need to recompile these executables by yourself.
- If your raw diploid assembly has >2000 scaffold sequences, make sure you have lift your system's openable file handle limit by invoking "ulimit -n 655350" or more before running HM2. This is a solution to a small glitch of several third-party executables. See the **Advanced Topics** for how to change this limit.

Resources consumption

- With a 900Mb diploid genome assembly, and with default parameter settings on a 4-core Xeon 2.2Gz machine with 8 Gb memory, it takes ~15 wall-time hours to finish the task.
- If multi-threading is allowed and 4 CPU cores are assigned to HaploMerger2, the running time can shorten to ~4 wall-time hours. We recommend using a machine with 12 CPU cores and 64Gb memory.
- With default parameter settings, each thread consumes less than 1.5 Gb memory.
- As for hard-disk space consumption, HaploMerger requires space about 4 times the size of the diploid genome assembly. However, this does not include the temporary raw Lastz alignments, which can occupy 10-100Gb, depending on the parameters you choose. So you may want to delete these temporary alignment files after the ChainNet stage has been successfully finished.
- If you have changed parameters affecting memory consumption, you may need to monitor the memory consumption closely by yourself (e.g., parameters in all_lastz.ctl).

Requirements

- **Multi-threading Perl version 5.10.1+.** [Required]
You may run the command “perl -v” to check if your Perl executable has multi-threading support enabled: “This is perl, v5.XX.X (*) built for x86_64-linux-gnu-thread-multi”.
- **LASTZ.** [Required; a copy is shipped with the HM2 package]
This is a local sequence alignment program used to generate highly-sensitive all-against-all genome alignments.
Website: <http://www.bx.psu.edu/~rsharris/lastz/>
Citation: Harris RS. 2007. Improved pairwise alignment of genomic DNA. PhD Thesis, The Pennsylvania State University.
And also: Schwartz S, et al. 2003. Human-mouse alignments with BLASTZ. *Genome Res* 13(1): 103-107.
- **ChainNet programs.** [Required; a copy is shipped with the HM2 package]
They are used to compute the reciprocally-best pairwise genome alignments.
Website: <http://hgdownload.cse.ucsc.edu/admin/jksrc.zip>
Citation: Kent WJ, et al. 2003. Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proc Natl Acad Sci U S A* **100**(20): 11484-11489.
- **WindowMasker.** [Optional]
This program does a very good job in soft-masking a draft genome sequence.
Website: http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/lxr/source/src/app/winmasker/
Citation: Morgulis A, et al. 2006. WindowMasker: window-based masker for sequenced genomes. *Bioinformatics*. 15;22(2):134-41.
- **SSPACE standard v3.0.** [Optional]
This perl script is a fast, robust hierarchical scaffolder for genome assemblies.
Website: <http://www.baseclear.com/genomics/bioinformatics/basetools/SSPACE>
For Perl version 5.16.1+, SSPACE v3.0 requires getopts.pl (Perl4::CoreLibs) to run.
Citation: Boetzer M, et al. 2010. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*. 27(4):578-579.
- **GapCloser v1.12.** [Optional]
This program is dedicated to fill the N-gaps emerged during the scaffolding process.
Website: <http://sourceforge.net/projects/soapdenovo2/files/GapCloser/bin/>
Citation: Luo R, et al. 2012. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*. 1(1):18.
- **Gmaj.** [Optional]
This is an alignment viewer for the MAF-formatted alignments output by HaploMerger2.
Website: <http://globin.bx.psu.edu/dist/gmaj/>
Description of the MAF format: <http://genome.ucsc.edu/FAQ/FAQformat.html#format5>

Installation

HaploMerger2 (HM2) has been tested on CentOS/RedHat v5/6 and Ubuntu v10.04-64bit.

Download the latest standalone HaploMerger2 package from our website (<https://github.com/mapleforest/HaploMerger2/releases/>) and unpack it to any directory you like.

You will see a directory tree like this:

```
>HaploMerger2_20151031
>>bin ## all perl and C programs of HM2
>>lastz_1.02.00_centOS5
>>lastz_1.02.00_centOS6
>>lastz_1.02.00_unbuntu64bit
>>chainNet_jksrc20100603_centOS5
>>chainNet_jksrc20100603_centOS6
>>chainNet_jksrc20100603_unbuntu64bit
>>winMasker
>>gapCloser_v1.12
>>Gmaj
>>SSPACE-STANDARD-3.0_linux-x86_64
>>project_template ## shell scripts and parameter files
>>project_example1
>>project_example2
>>project_example3
```

Directories `chainNet*` and `lastz*` contain the pre-compiled executables for chainNet and Lastz. Select the right version according to your Linux system and add these directories to your own \$PATH so that HM2 scripts can access them. If these executables do not work in your system, you may download the source code and recompile them by yourself.

Directories `winMasker`, `gapCloser_v1.12` and `SSPACE-STANDARD-3.0_linux-x86_64` are empty. If you want to benefit from these software packages, you should download and compile them by yourself and put them to these directories (see the “Requirements” section). Please add these directories to your own \$PATH so that HM2 scripts can access them.

If you like to test HM2 with a real example, you may consider downloading these extra files:

- 1) Download the diploid assembly (bbv18wm.fa.gz) of the Chinese amphioxus from <https://github.com/mapleforest/HaploMerger2/releases/> and put it under `project_example3`;
- 2) For re-scaffolding, download the 2/3/8/20kb mate-pair libraries (scaffolding_libraries.fa.gz) from <https://github.com/mapleforest/HaploMerger2/releases/> and unpack them to `project_example3/libraries`;
- 3) For gap filling, download two short-gun libraries (accession: SRX1364942 and SRX1364943) from the NCBI SRA database and put them under directory `project_example3/libraries`;
- 4) Please read the `README.txt` in directory `project_example3` for more information.

Understanding the HaploMerger2 pipeline

Enter the directory named `project_template` and list its files, you will see:

```
>run_all.batch      ## the master shell script of all the following shell scripts
>hm.batchA1.initiation_and_all_lastz    ## run all-against-all whole genome alignment
>hm.batchA2.chainNet_and_netToMaf      ## compute reciprocally-best alignment
>hm.batchA3.misjoin_processing          ## misjoin detection and removal
>hm.batchB1.initiation_and_all_lastz
>hm.batchB2.chainNet_and_netToMaf
>hm.batchB3.haplomerger                ## create initial haploid assemblies
>hm.batchB4.refine_unpaired_sequences  ## refine unpaired sequences (single alleles)
>hm.batchB5.merge_paired_and_unpaired_sequences## create final haploid assemblies
>hm.batchC1.hierarchical_scaffolding   ## further scaffold the haploid assembly
>hm.batchC2.update_reference_and_alternative_assemblies ## output new assemblies
>hm.batchD1.initiation_and_all_lastz
>hm.batchD2.chainNet_and_netToMaf
>hm.batchD3.remove_tandem_assemblies ## remove tandems from haploid assemblies
>hm.batchE1.wrapper_for_gapCloser_v1.12 ## close N-gaps in haploid assemblies
>scoreMatrix.q                        ## template: score matrix for lastz
>scoreMatrix.default.q
>scoreMatrix.stringent.q
>scoreMatrix.more_stringent.q
>all_lastz.ctl                        ## template: configuration for lastz
>all_lastz.default.ctl
>all_lastz.fast.ctl
>libraries/                          ## to hold reads libraries for scaffolding and gap filling
>> gapCloser.cfg                     ## template: configuration for gapCloser
>> sspace_libraries.list             ## template: library list and configuration for SSPACE
```

HM2 programs have been organized into several shell scripts (`hm.batchA-E`) which are supposed to be run in order, but certainly some of them can be skipped and some of them can be iterated. Open and read the master shell script (`run_all.batch`) to get an idea on how to run these scripts.

The master shell script is meant to be frequently modified by users to fulfill their particular purposes.

The `all_lastz.ctl` file contains parameters which control the behavior of LASTZ. This file has been optimized for diploid genome assemblies with 4-5% heterozygosity and a medium haploid genome size (i.e., 500Mb). The `scoreMatrix.q` file contains a scoring scheme for alignments, which is optimized for the alignments (with 90-100% identity) of the amphioxus diploid genome assemblies (4-5% heterozygosity and 42% GC content).

Open and read each shell scripts. All scripts consist of three major parts: 1) setting variables, which is meant to be frequently modified by users, 2) printing out the names of important output files, and 3) invoking executables.

```

0      10      20      30      40      50      60      70      80      90      100
1 ##### hm.batchA1.initiation_and_lastz v1.50
2 #####
3 ##### This shell script does two tasks:
4 ##### 1) to create an environment that contains all the files and directories required for later use,
5 ##### 2) and to perform whole genome pairwise self alignments by using lastz (all against all).
6
7 #### setting variables
8 name=$1          # the species_name or the prefix of the genome file name
9 threads=1        # the number of cpu cores to use (default=1)
10 identity=80      # output only lastz alignments >= $identity (default=80)
11                # normally, you do not need to change identity because the program will optimize it
12 targetSize=50000000 # split target fasta file by size N bp (default=50000000)
13                # NOTE: if you use 8 or 16 CPUs, you may need to choose a proper value to
14                # divide the target sequence file in order to exploit the benefit of multiple CPUs
15 querySize=1600000000 # split query fasta file by size N bp (default=1600000000)
16
0      10      20      30      40      50      60      70      80      90      100
37
38 #### print out output file names
39 echo
40 echo "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
41 echo "!!! ChainNet & lastz may require to simultaneously open >100,000 file handles, !!!"
42 echo "!!! so the file handle limit of the system should be lifted for this script   !!!"
43 echo "!!! By invoking (with root privilege) >ulimit -n 655350 .                    !!!"
44 echo "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
45 echo
46 echo "These files/directories are going to be output : "
47 echo "  ${name}.x.fa.gz - a copy of the genome sequence file ${name}.fa(.gz)"
48 echo "  ${name}.sizes - the size of each scaffold sequences"
49 echo "  ${name}.x.sizes - the size of each scaffold sequences"
50 echo "  ${name}.seq/*.fa and *.nib - fasta files and nib files for scaffold sequences"
51 echo "  ${name}.x.seq/*.fa and *.nib - fasta files and nib files for scaffold sequences"
52 echo "  *** ${name}.${name}.x.result/raw.axt/*.axt, *.axt.*, and *.log - raw lastz result files and their loc
53 echo "log files:"
54 echo "  _mp.initiation.log - log file"
55 echo "  _mp.all_lastz.log - log file"
56 echo
57
0      10      20      30      40      50      60      70      80      90      100
58 ##### =====
59 #### create the project environment
60 ##### =====
61
62 rm -f -r ${name}.seq ${name}.x.seq ${name}.sizes ${name}.x.sizes ${name}.x.fa.gz
63 perl ../bin/initiation.pl --faSplit --faToNib --faSize --Species ${name} --Force --Delete \
64 1>_mp.initiation.log 2>>_mp.initiation.log
65 ln -s ${name}.fa.gz ${name}.x.fa.gz
66 ln -s ${name}.sizes ${name}.x.sizes
67 ln -s ${name}.seq ${name}.x.seq
68
69 ##### =====
70 #### run lastz all against all
71 ##### =====
72
73 ## require two control files: 1) all_lastz.ctbl|${name}.${name}.x.ctbl, and 2) scoreMatrix.q|${name}.${name}.x.q
74
75 rm -f -r ${name}.${name}.x.result/raw.axt
76 perl ../bin/HM_all_lastz_mThreads.pl --Species ${name} ${name}.x --noself --threads=${threads} --identity=${identity}
77 --targetSize=${targetSize} --querySize=${querySize} --Force --Delete \
78 1>_mp.all_lastz.log 2>>_mp.all_lastz.log
79

```


Quick Start – testing HM2 with the example 1 and 2

Enter the directory named `project_example1/2`. There are two files and one directory in it, including `README.txt`, `genome.fa.gz` and `test1/2_correct_output`. The `genome.fa.gz` file contains the example diploid genome sequences.

Now, let us test if the core part of HM2 works correctly on your system (test1).

1. Copy `hm.batchB1-5`, `all_lastz.ctf` and `scoreMatrix.q` from `project_template` to `project_example1`.
2. Switch to the directory `project_example1`, and make sure that you set `$PATH` correctly so that all HM2 executables can be accessed.
3. Use `genome.fa.gz` as the input file (no need to type the suffix “fa.gz”) and run the following shell scripts step by step while observing the information on the monitor screen:

```
$ sh ./hm.batchB1.initiation_and_all_lastz genome
$ sh ./hm.batchB2.chainNet_and_netToMaf genome
$ sh ./hm.batchB3.haplomerger genome
$ sh ./hm.batchB4.refine_unpaired_sequences genome
$ sh ./hm.batchB5.merge_paired_and_unpaired_sequences genome
```
4. Each script will finished in seconds and you will see each script outputs certain directories, files and logs in the current directory. After `hm.batchB5` is done, you are supposed to get two final result files named `./genome_ref.fa.gz` and `./genome_alt.fa.gz` with four scaffold sequences in each of them. You may also compare your outputs with those in `test1_correct_output` for trouble shooting.

Then, let us test a little more (test2).

1. Copy `run_all.batch`, `hm.batchA/B/D*`, `all_lastz.ctf` and `scoreMatrix.q` from `project_template` to `project_example2`.
2. Switch to the directory `project_example2`, and make sure that you set `$PATH` correctly so that all HM2 executables can be accessed.
3. This time, we use the master shell script to do the test. Open, modify and save `run_all.batch`: (a) set the `$PATH` correctly in line 13; (b) you should comment out all other shell scripts except `hm.batchA/B/D*` by adding a “#” to the beginning of the corresponding command lines; (c) replace “your_assembly_name” with “genome” in line 22-24; replace “your_assembly_name_A” with “genome_A” in line 33-37; (d) replace “your_assembly_name_A_ref_C” with “genome_A_ref” in line 55-57.
4. Return to the terminal and run:

```
$ sh ./run_all.batch >run_all.log 2>&1 ## collect all screen output to run_all.log
```

You can also run each script of `hm.batchA/B/D*` step by step to see how it is working.
5. Each script will finished in seconds and you will see each script outputs certain directories, files and logs in the current directory. You are supposed to get these final results:
from `hm.batchA1-3`
`./genome_A.fa.gz` *## the diploid assembly with misjoins removed*
from `hm.batchB1-5`
`./genome_A_ref.fa.gz` *## the reference haploid assembly*

```
./genome_A_alt.fa.gz  ## the alternative haploid assembly  
from hm.batchD1-3
```

```
./genome_A_ref_D.fa.gz  ## the reference assembly with tandems removed
```

6. You may compare you outputs with those in `test2_correct_output` for trouble shooting.

Start your own project

The minimum requirement for your own project is **a diploid genome assembly in fasta format and is contained in a gzip-compressed file** with suffix “**.fa.gz**”. This assembly must be **soft-masked**, which means that the repetitive sequences (simple repeats, transposons and highly duplicated coding sequences) and low-complexity sequences should be represented in lowercases. Both RepeatMasker and winMasker can be used to soft-mask the diploid assembly. We recommend beginning with **winMasker** because it is easier and faster to run without requiring prior knowledge of the repetitive sequences (which is usually not readily present for a newly-assembled genome). Please see “advance topic” for more information about softmasking.

HM2 is sensitive to illegal characters and sequences. We strongly recommend cleaning up the fasta sequences by using faDnaPolishing.pl from HM2:

```
gunzip -c genome.fa.gz | ./bin/faDnaPolishing.pl --legalizing \
--maskShortPortion=1 --noLeadingN --removeShortSeq=1 >genome_cleaned.fa.gz
```

You may not want to mess up the **project_template** directory. So you build a new project directory such as **myProject** in the HM2 directory and copy files from **project_template** to it.

If you want to run hierarchical scaffolding and gap filling, please also put the reads libraries under **myProject/libraries**. Symbol links and gzipped fasta/fastq files are accepted.

NOTE THAT **run_all.batch**, the “setting variable” section of all shell scripts **hm.batchA-E** and all configuration files (**all_lastz.ctl**, **scoreMatrix.q**, **libraries/sspace_libraries.list** and **libraries/gapCloser.cfg**) are meant to be modified by users to suit their particular preferences and purposes. For example, to speed up the process by allowing multiple threads, you may open the shell scripts and modify the following variable (if it is presented and configurable):

```
threads=1    # the number of cpu cores to use (default=1).
```

NOTE THAT you are supposed to run all shell scripts and commands in your project directory!

NOTE THAT if you have more than several thousand scaffolds, you need to increase the openable file handle number by invoke “ulimit -n 655350”. You may need the root privilege to do so. See the **Advanced Topics for how to set this limit.**

The shell scripts will print out the names of the major output files. Important output files are those fasta-formatted genome assembly file placed in your project directory. These gzipped fasta files can be used as input files for the next step or next round of HM2 procedures. Different suffixes will be added to the resulted file names to show different steps of HM2 procedures:

- “_A” added by **hm.batchA** stands for “misjoin_processing / mp”;
- “_ref/alt” added by **hm.batchB** stands for “reference/alternative”;
- “_C” added by **hm.batchC** stands for “secondary scaffolding / ss”;
- “_D” added by **hm.batchD** stands for “removing_tandems / rt”;
- “_E” added by **hm.batchE** stands for “gap_filling / gf”.

Details of Operations – a walk-through with the example 3

Step 0: preparation

Enter the directory named `project_example3` and check around. The environment has been nearly set up.

If you want to test HM2 with `project_example3`, you need to download the diploid assembly file (`bbv18wm.fa.gz`) from <https://github.com/mapleforest/HaploMerger2/releases/> (also deposited in GenBank under accession GCA_001625405.1) and place it in `project_example3`. This fasta file contains the `soft-masked (by winMasker)` diploid genome assembly of a wild-type Chinese amphioxus. We are going to run HM2 on this diploid genome assembly.

If you want to run hierarchical scaffolding on haploid assemblies, you may also download the 2kb, 3kb, 8kb and 20kb mate-pair libraries (packed together in `scaffolding_libraries.tar.gz`) from <https://github.com/mapleforest/HaploMerger2/releases/> and unpack them to `project_example3/libraries/`. These mate-pair reads are also available on NCBR SRA database:

Accession SRX1421071:

Store reads of run `2kb_libraries_pair1` as a gzipped Fasta file named `pe2kb40bp.1.fa.gz`.

Store reads of run `2kb_libraries_pair2` as a gzipped Fasta file named `pe2kb40bp.2.fa.gz`.

Note that SSPACE does not care about the read name; only the read order matters.

Accession SRX1421072:

Store reads of run `3kb_libraries_pair1` as a gzipped Fasta file named `pe3kb40bp.1.fa.gz`.

Store reads of run `3kb_libraries_pair2` as a gzipped Fasta file named `pe3kb40bp.2.fa.gz`.

Note that SSPACE does not care about the read name; only the read order matters.

Accession SRX1421073:

Store reads of run `8kb_libraries_pair1` as a gzipped Fasta file named `pe8kb40bp.1.fa.gz`.

Store reads of run `8kb_libraries_pair2` as a gzipped Fasta file named `pe8kb40bp.2.fa.gz`.

Note that SSPACE does not care about the read name; only the read order matters.

Accession SRX1421074:

Store reads of run `20kb_libraries_pair1` as a gzipped Fasta file named `pe20kb40bp.1.fa.gz`.

Store reads of run `20kb_libraries_pair2` as a gzipped Fasta file named `pe20kb40bp.2.fa.gz`.

If you want to run the N-gap filling functions, please download the short-gun and paired-end reads libraries (in Fastq format) from the NCBI SRA database and place them in the directory `project_example3/libraries`.

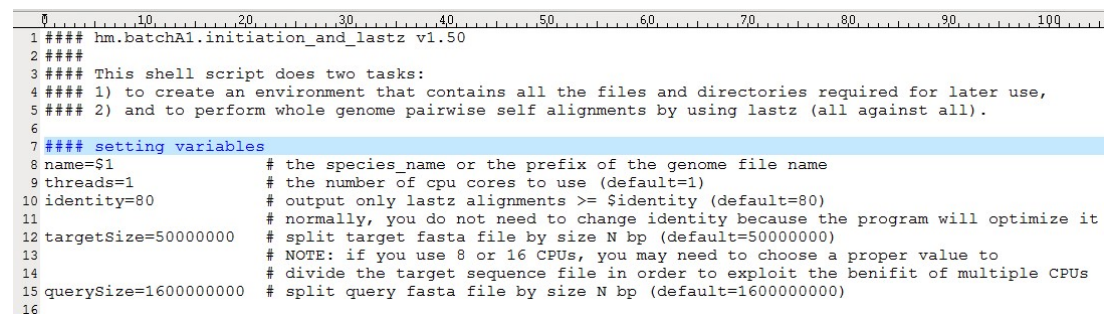
Accession SRX1364942:

Please separate read#0 and read#1 into two gzipped Fastq file named `5_1/2.cor.fq.gz`.

Accession SRX1364943:

Please separate read#0 and read#1 into two gzipped Fastq file named `7_1/2.cor.fq.gz`.

HM2 have been organized into several shell scripts named as `hm.batchA-E`. They serve as both script files and parameter files. To set parameters you should open them with a text editor to modify the parameters. The following figure shows the content of `hm.batchA1`.



```

1 ##### hm.batchA1.initiation_and_lastz v1.50
2 #####
3 ##### This shell script does two tasks:
4 ##### 1) to create an environment that contains all the files and directories required for later use,
5 ##### 2) and to perform whole genome pairwise self alignments by using lastz (all against all).
6
7 ##### setting variables
8 name=$1          # the species_name or the prefix of the genome file name
9 threads=1        # the number of cpu cores to use (default=1)
10 identity=80      # output only lastz alignments >= $identity (default=80)
11                 # normally, you do not need to change identity because the program will optimize it
12 targetSize=500000000 # split target fasta file by size N bp (default=500000000)
13                 # NOTE: if you use 8 or 16 CPUs, you may need to choose a proper value to
14                 # divide the target sequence file in order to exploit the benefit of multiple CPUs
15 querySize=1600000000 # split query fasta file by size N bp (default=1600000000)
16

```

Finally, open and read `run_all.batch` to understand the whole process. Edit the `$PATH` variable to suit your environment. You may want to examine the result after each step, so you may comment out the steps you do not want to run currently.

Step 1: run `hm.batchA1-3` to remove major misjoins from the diploid assembly

`hm.batchA1.initiation_and_all_lastz bbv18wm`.

It creates a set of required files, and then invokes LASTZ to produce raw all-against-all whole-genome alignments for your diploid genome sequences. Alignment files are stored in the directory `bbv18wm.bbv18wmx.result/raw.axt`. Depending on what sensitivity you choose to use, raw alignments might occupy a lot of disk space; you can delete it once you have successfully finished the next script.

`hm.batchA2.chainNet_and_netToMaf bbv18wm`.

It uses the ChainNet algorithm to create reciprocally-best single-coverage pairwise whole-genome alignments.

Both `hm.batchA1` and `2` can be parallelized to speed up the alignment process. Open both script files and set the thread number that you want to use:

```
threads=1          # the number of cpu cores to use (default=1).
```

Make sure you have that number of spare CPU/core to allocate. Each thread requires less than 1Gb memory at the default setting, and that a total of 4-8Gb memory should be available for dynamic allocation (required by the chainNet process).

To reduce non-orthologous alignments, edit `hm.batchA1` and set a proper identity threshold:

```
identity=80      # output only lastz alignments >= $identity (default=80).
```

Identity of 80% works fine in many situations, but you may increase it for more stringent filtering. Empirically, if allelic difference of your genome is 4-5%, you may set it to 88-92% to achieve a more stringent filtering. You are recommended to use a more flexible method to discriminate between allelic and non-allelic alignments by using a genome-specific `scoreMatrix.q`.

There are two important parameter files (`all_lastz.ctl` and `scoreMatrix.q`) which are used to

control the behavior of alignment process. The file `all_lastz.ctl` controls the behavior of LASTZ program. For example, you may reset “--step=1” to “--step=20” to speed up the LASTZ process by >3 time faster. Theoretically, “--step=20” may cause some loss of alignment sensitivity. Note that the parameters in this file may greatly affect running time, memory consumption, and alignment sensitivity/specificity. The default setting in this file is a good starting point for your project, but you may modify any of these parameters to meet your needs. For detailed usage of these parameters, you may refer to the LASTZ manual. Generally, higher sensitivity will yield better quality of alignment and consume more running hours.

The file `scoreMatrix.q` contains a nucleotide substitution score matrix which is used by both LASTZ and chainNet. The default matrix is inferred from the Chinese amphioxus diploid genome assembly (4-5% polymorphism rate and 41% GC content) and is a good starting point for your project. However, because difference diploid genomes have different heterozygosity rates and GC bias, so it is highly recommended to infer a score matrix specific for your own genome assembly to achieve optimal alignment sensitivity and specificity! Theoretically, a more stringent scoring scheme can both increasing alignment specificity and saving running time, but in the cost of losing alignment sensitivity. LASTZ provide a program (lastz_D) to infer the score matrix for a provided genome sequence. Please see “Advanced topics” and “Other tools” to learn how to infer score matrix. And see the LASTZ manual for more information.

`hm.batchA3.misjoin_processing bbv18wm.`

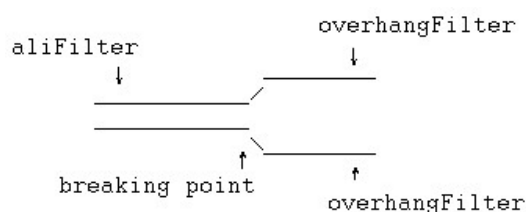
It invokes three Perl programs to do five tasks:

1. to pre-filter the reciprocally-best whole-geome chainNet alignments and make them perfectly-mirrored;
2. to convert whole-genome alignment into a directed graph (the DGA graph);
3. to traverse the DGA graph using a greedy algorithm and search for misjoins;
4. to break the two involved scaffolds at the misjoining point;
5. output the diploid assembly based on the processed DGA graph.

A set of parameters control the behavior of this script.

`breakingMode`, `aliFilter`, and `overhangFilter`.

There are events in which the long-term co-linearity of two haplotypes is violated:



These events may represent assembly errors, or to a less probability, allele-specific chromosome rearrangements. HM2 breaks both scaffolds at the breaking point (`breakingMode` must be set to 2). The option `aliFilter` controls the minimum alignment score (the default is about 50kb), whereas the option `overhangFilter` controls the minimum length of the unaligned scaffold portions (the default is about 50kb). The higher of these two options are, the more reliable an event of

co-linearity violation is found.

`scoreScheme=<score|ali>`.

HM2 ranks the alignments in certain order. A higher rank means an alignment is longer and is more reliable in representing a pair of allelic sequences. Alignments can be ranked by their score or their alignment length (the length excluding small indel, gaps and Ns). Currently, you should choose to use the “score” scheme.

`filterScore_1`, `escapeFilter`, `filterScore_2` and `NsLCsFilter`.

Most of the low-scored alignments are spurious and can be classify as “noise”. Three options are used to control the filtering of noise. Option `filterScore_1` control the filtering on the graph creating stage; option `filterScore_2` is the second round of filtering in the course of the DGA graph traversing; option `escapeFilter` allow to retain those low-scored alignments (`<filterScore_2`) if they cover XX% of the scaffold sequence length. Note that there is no exact demarcation between noise and non-noise alignments, but the default setting should serve as a suitable starting point. Empirically, to meet the criterion of `filterScore_2=200000`, an alignment should have at least 2000 perfect match nucleotides without gaps, indels and Ns. For some scaffold sequences short than, like, 2000bp, the setting `filterScore_2=200000` makes no sense. In this case, option `escapeFilter` helps to reclaim the alignments involving short scaffold sequences. However, these short alignments still have to pass the criterion `filterScore_1`. Some alignments mainly comprise of Ns and repeats, option `NsLCsFilter` can help to remove these alignments. This option operates independently.

In addition, there are several output files describing the co-linearity violation events:

`hm.scaffolds` - *information of original scaffolds*

`hm.nodes` - *information of the alignment blocks (a block is a node)*

`hm.portions` - *information of the portions of original scaffolds*

`hm.new_scaffolds` - *the solved relations between two haplotypic scaffolds*

`hm.assembly_errors` - *potential assembly errors in the original assembly*

**** About running multiple rounds of misjoin removing on the diploid assembly ****

Because misjoins can affect the genome alignments, one round of `hm.batchA1-3` may not identify all potential misjoins. You can use `the output diploid assembly file as the input` for a second round of `hm.batchA1-3`. Usually 2-3 rounds of `hm.batchA1-3` will be sufficient. You can also lower `aliFilter`, and `overhangFilter` (from the default 50kb to 40kb, but caution against values lower than 30kb) to allow more misjoins to be processed in the second and the third rounds.

Step 2: run `hm.batchB1-5` to create the haploid assemblies

`hm.batchB1.initiation_and_all_lastz bbv18wm_A.`

`hm.batchB2.chainNet_and_netToMaf bbv18wm_A.`

This two scripts are similar to those of the Step1, except that 1) the input diploid assembly file should be the output of the Step 1, and 2) and the pairwise chainNet alignments are output in MAF format. You may examine these MAF alignments with Gmaj or other alignment viewers.

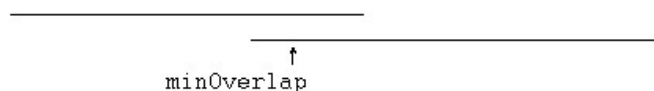
`hm.batchB3.haplomerger bbv18wm_A`.

This script invokes three Perl programs to do four tasks:

1. to pre-filter the reciprocally-best whole-genome alignments and make them perfectly-mirrored;
2. to convert whole-genome alignment into a directed graph (the DGA graph);
3. to traverse and linearize the DGA graph using a greedy algorithm;
4. to output the reference and alternative haploid assemblies based on the linearized DGA graph.

The parameters of this script are similar to that of `hm.batchA3`. We note that increasing `filterScore_2` and `escapeFilter` may reduce the chance of merging two sequences which are not really alleles. However, it will cause the resulted haploid assembly to lose some correct alignments and to include more redundant sequences (redundancy means that both alleles are included in the resulted assembly). Some alignments mainly comprise of Ns and repeats, option “`NsLCsFilter`” can help to remove these alignments (this option operates independently).

HaploMerger is able to connect two scaffolds into one if they share an overlapping region (alignments).



The option `minOverlap` controls the minimum alignment length (excluding Ns, gaps and indels) that can trigger a connecting operation. You can set `minOverlap` to zero (the default) to let HaploMerger to connect any connectable scaffolds. However, since all low-scored alignments will be filtered out by the option `filterScore_2`, so the minimum alignment length for scaffold connecting is actually large than `filterScore_2`. Increase `minOverlap` can make the retained connections more trustable in the cost of losing many possible connections. Because any connection will mix two haplotypes together, and in some cases you don't want to mix up haplotypes or to generate switches between haplotypes. In these cases, you may set `minOverlap` to a very large value, such as 99,999,999.

Three important output files will be generated and placed under the directory `bbv18wm_mp.bbv18wm_mpx.result`.

`optiNewScaffolds.fa.gz` - contains new scaffolds with alignment supports
`optiNewScaffolds_alt.fa.gz` - new alternative scaffolds with alignment supports
`unpaired.fa.gz` - new scaffolds with no alignment supports

The third file collects scaffolds that have no trustable alignments shared with other scaffolds.

In addition, there are many other output files describing the DGA graph structure and the co-linearity violation events:

`hm.scaffolds` - information of original scaffolds
`hm.nodes` - information of the alignment blocks (a block is a node)
`hm.portions` - information of the portions of original scaffolds

`hm.new_scaffolds` - *the solved relations between two haplotypic scaffolds*

`hm.assembly_errors` - *potential assembly errors in the original assembly*

`hm.unpaired` - *a list of the unpaired scaffolds*

`hm.connections` - *the alignment blocks connecting two different scaffolds*

`hm.unpaired_updated` - *an updated list of the unpaired scaffolds*

These data can be used for further refinement, manual inspection and also for secondary exploitation. Note that all data files are output in TAB-separated format, and there is detailed explanatory description inside these files. You may open them with a text editor or Excel.

`hm.batchB4.refine_unpaired_sequences bbv18wm_A`.

The `unpaired.fa.gz` file contains those scaffolds without alignment supports. Several reasons will cause this:

- 1) some sequences have no corresponding allelic sequences in the diploid assembly;
- 2) a pair of alleles/haplotypes is collapsed into one sequence;
- 3) some true alignments between alleles are relatively weak and filtered out in the Stage I & II;
- 4) some alignments mainly consist of Ns and repeats and hence have been filtered out;
- 5) some alignments are related to tandem duplication, inversion and translocation, and they don't considered in the linearized DGA graph currently;
- 6) besides, in the course of graph traversing, some portions of scaffolds can be cut off (such as the free ends of a scaffold, which fail to be aligned), which also collect into the `unpaired.fa.gz` file.

Considering the origin of these sequences, one may expect that a large proportion of these sequences may actually redundant. This script is developed to identify those redundant sequences and to remove them from the `unpaired.fa.gz` file.

This script aligns the sequences in `unpaired.fa.gz` against to those in `optiNewScaffolds.fa.gz`; then it removed the potentially-redundant sequences from `unpaired.fa.gz` if they can be found (aligned) in `optiNewScaffolds.fa.gz`. It produces a new output Fasta file named `unpaired_refined.fa.gz`. You may use multiple CPU/cores by setting the "threads" option. You may set the "identity" option to control the sensitivity and specificity of the alignment. The value may set lower than that used in the Stage I to recover more alignments. You may set the "maskFilter" option to filter out those scaffolds mainly comprised of Ns and repeats. You may set the "redundantFilter" option to filter out those scaffolds having corresponding allelic sequences in the `optiNewScaffolds.fa.gz` file.

`hm.batchB5.merge_paired_and_unpaired_sequences bbv18wm_A`.

It merges `optiNewScaffolds.fa.gz` and `unpaired_refined.fa.gz` to produce the reference haploid assembly (`bbv18wm_A_ref.fa.gz`), and merges `optiNewScaffolds_alt.fa.gz` and `unpaired_refined.fa.gz` to produce the alternative haploid assembly (`bbv18wm_A_alt.fa.gz`).

When there are two alleles representing a genomic locus, they will be separately placed in `bbv18wm_A_ref.fa.gz` and `bbv18wm_A_alt.fa.gz`. When there is only one representing allele for a genomic locus, this allele will be placed in both the reference and the alternative assemblies.

Step 3: run hm.batchC1-2 to further scaffold the haploid assemblies

`hm.batchC1.hierarchical_scaffolding bbv18wm_A_ref`

This script invokes the SSPACE v3.0 to further scaffold the haploid assembly. We only need to feed the reference haploid assembly to scaffolding. There are several options for this script:

```
threads=1                # the number of cpu cores to use (default=1)
minimumLinks=5           # Minimum number of links (read pairs) to compute scaffold
libraryList="libraries/sspace_libraries.list" # the library list file for SSPACE
```

The `libraries/sspace_libraries.list` file contains the mate-pair libraries. We note that short distance mate-pairs should be used before the long distance mate pair. For more information on the usage of SSPACE v3.0, please refer to its manual.

`hm.batchC2.update_reference_and_alternative_assemblies bbv18wm_A_ref`

Usually only the reference haploid assembly needs to be further scaffold. `hm.batchC2` can extract the new-to-old scaffold layout and use this layout to re-scaffold the alternative haploid assembly. So, this script finalizes the scaffolding and outputs the new reference (`bbv18wm_A_ref_C.fa.gz`) and the new alternative (`bbv18wm_A_alt_C.fa.gz`) haploid assemblies.

**** About running multiple rounds of `hm.batchC1` on the haploid assembly ****

After the first round of scaffolding, some new joining possibilities will emerge for the new assembly. You may use `the output haploid assembly file as the input` for a second round of `hm.batchC1`. Usually 2-3 rounds of `hm.batchC1` will be sufficient. You may also lower `minimumLinks` to allow less reliable joining of scaffolds in a second round of scaffolding.

**** About using other scaffolders ****

Scaffolders other than SSPACE can be used to further scaffold the haploid assembly. `hm.batchC2` determines the new-to-old scaffold layout by searching the exact 1:1 match between the old scaffold sequences and the new scaffold sequences. Therefore, as long as other scaffolders do not introduce or discard sequence content (i.e., filling gaps, adding new sequence, closing gaps by merging sequences and cut away some sequences, etc), `hm.batchC2` can be used to process the new assembly.

Step 4: run hm.batchD1-3 to remove tandem errors from the haploid assembly

Normally, only the reference haploid assembly needs to be processed, but if you want to process the alternative haploid assembly, you may feed the alternative assembly file to `hm.batchD1-3`.

It should be noted that this is the only step in HM2 that will lose genomic information, so we strongly recommend being careful with this module. To avoid permanent loss of the genomic information, MH2 will collect all the excised tandem sequences in an independent fasta file (`_D3.tandem_removal_excised_seq.fa`).

Previous studies showed that genome assemblies with high polymorphism (such as *Ciona savignyi* and *Branchiostoma floridae*) contain many tandem assembly errors, which means two alleles are mistakenly assembled in tandem. This kind of errors affects up to 12% of the genome sequence (or 6% of the allelic sequences).

```
hm.batchD1.initiation_and_all_lastz bbv18wm_A_ref_C
hm.batchD2.chainNet_and_netToMaf bbv18wm_A_ref_C
```

These two scripts do things similar to `hm.batchA1-2` and `hm.batchB1-2`.

```
hm.batchD3.remove_tandem_assemblies 18wm_A_ref_C
```

This script detects and cuts away the tandem mis-assembled region and rejoins the sequence. The script utilizes a novel algorithm to automatically detect and remove these tandem assembly errors from the haploid assembly with high precision.

There are two options to control the size of tandems for processing: `filterAli` and `minLen`. `minLen` controls the minimum alignment length for consideration of removing, while `filterAli` directly filters out alignments whose length lower than the given value. Normally, both options could be set to the same value. Unlike HM1, HM2 can detect and removal potential tandem errors with size as small as 200bp, but we recommend `minLen>2500` bp. For the two tandem assembled alleles, HaploMerger chooses to remove the shorter ones with more Ns.

**** About running multiple rounds of `hm.batchD1-3` on the haploid assembly ****

Different sizes of tandem mis-assemblies can interfere with each other, therefore multiple rounds of tandem removal may be needed. You can use `the output haploid assembly file as the input` for a second round of `hm.batchD1-3`. We recommend to remove tandems from large sizes to small sizes, for example:

```
round1: filterAli=4000 and minLen=5000
round2: filterAli=2400 and minLen=3000
round3: filterAli=1000 and minLen=1500
```

The removed sequences are stored in `_D3.tandem_removal_excised_seq.fa`.

Step 5: run `hm.batchE1` to try to fill some gaps in the haploid assembly

`Normally, only the reference haploid assembly needs to be processed`, but if you want to process the alternative haploid assembly, you may feed the alternative assembly file to `hm.batchE1`.

```
hm.batchE1.wrapper_for_gapCloser_v1.12 bbv18wm_A_ref_C_D
```

It invokes the `gapCloser v1.12` to try to fill the N-gaps in the reference haploid assembly. GapCloser is too aggressive because it tries to tile reads into a gap, no matter whether it can close the gap in the end (some gaps can be closed but the others cannot). This script one retains those closed gaps because they are supposed to be more reliable.

There are two options meant to be modified by users:

```
threads=1                # the number of cpu cores to use (default=1)
configFile="libraries/gapCloser.cfg"  # the configFile for GapCloser v1.12; this
parameter is transferred directly to the '-b' option of GapCloser v1.12
```

There are two output files:

```
bbv18wm_A_ref_C_D_E.fa.gz  # the haploid assembly with some gap filled; The contigs
used to filled in the gaps are soft masked (in lower cases).
```

```
bbv18wm_A_ref_C_D_E.agp  # the AGP file showing the contig-scaffold structure of the
assembly; The contigs used to filled in the gaps are labeled with a prefix "GF_" in the
agp file.
```

BUG: Sometimes GapCloser will be stuck on some gaps. A quick solution is to stop the affected threads manually.

You can still finish hm.batchE after you manually stop the problematic threads. All you lose are the chances to close those affected N-gaps.

Advanced Topics

1. How to lift the openable file handle limit? Could it cause a problem to the system?

The easy way: run as root, to execute the command `ulimit -n 655350` (default is 1024).

The hard way: run as non-root user. First to modify the limit file `/etc/security/limits.conf`, add two lines to it and save it (you still need root privilege to do this, or you may have to ask your system administrator to do this):

```
your_user_name soft nofile 655350
your_user_name hard nofile 655350
```

Then save the file, and logout and login again to let the change take effect.

Now you can execute the command `ulimit -n 655350`. You need to set this each time you re-login or start a new shell, because each time the system will set file handle limit back to the default value (1024). You may want to reverse these changes after finish running HM2.

This is just a solution to a small glitch of several third-party executables. HM2 and third-party programs will NOT actually read/write many files simultaneously. So it will not pose a problem to your servers.

2. How to soft-mask the raw diploid genome assembly?

You can use RepeatMasker to soft-mask the raw diploid genome assembly, but that would require priori information about the repeat content of the genome, which is often not available for a newly-created genome assembly. Therefore, we recommend using WinMasker. You can run WinMasker on your genome assembly in two separate stages.

stage 1: create the reusable masking library

```
../winMasker/windowmasker_20120730 \
  -checkdup true \
  -mk_counts \
  -in bbv16.fa \           # the Fasta formatted genome assembly
  -out masking_library.ustat \ # the library used for future masking
  -mem 6500
```

stage 2: use the masking library to soft-mask the genome assembly

```
../winMasker/windowmasker_20120730 \
  -ustat masking_library.ustat \
  -in bbv16.fa \           # the input genome assembly
  -out bbv18wm.fa \       # the softmasked genome assembly
  -outfmt fasta \
  -dust true
```

The `masking_library.ustat` can be reused on other assembly from the same sequencing data.

3. How to speed up HaploMerger2?

The best way is to allocate more CPUs to HaploMerger.

To change `-step` in `all_lastz.ctl` from 1 to 20 can shorten the alignment time by >75%.

Theoretically, large step value may cause the loss of alignment sensitivity.

To use a more stringent score matrix and to lower the gap cost threshold `-ydrop` in `all_lastzctl` can also save lots of time, but these will cause the loss of true alignments. So, it is not recommended unless you know what you do.

To change `-identity` from 80 to 90 for `hm.batchA` or `HM_all_lastz_mThreads.pl` can force LASTZ to output only alignments with identity >90% and hence save some time. Again, it causes the loss of useful alignments. So, it is not recommended unless you know what you do.

4. How to increase alignment specificity?

You may need to use soft-masking instead of hard-masking. LASTZ can take advantage of soft-masking by extending alignment into the soft-masked regions, hence improving the score and length of the alignment.

You are highly recommended to use a score matrix specific for your genome sequence. It is probably the best way to control the alignment specificity without losing much sensitivity. You may use LASTZ_D to infer such a score matrix. Supposed that you have a genome sequence with 5% heterozygosity, then as a starting point, you may extract those alignments with 90-100% to infer a score matrix. See the section “Other tools” for more details.

To change `-identity` from 80 to 90 for `hm.batchA` or `HM_all_lastz_mThreads.pl` can force LASTZ to output alignments with identity >90%. This will increase specificity greatly. You very unlikely lose any true pair of alleles by increasing the identity threshold provided that you are not setting it too high (e.g., higher than the heterozygosity rate). In fact, it helps to remove spurious allele pairs (=increase the specificity). Usually, a pair of allele contains a set of high-identity alignment blocks (i.e., core alignments) and a set of low-identity alignment blocks. An allele pair is better defined by long-term co-linearity and a set of core alignments. The chaining scheme of HaploMerger (i.e., the chainNet algorithm) can recover the allele pair by chaining up the core alignments, regardless the presence/non-presence of low-identity alignments.

However, it should be noted that low-identity alignments for a pair of alleles are sometimes useful, for example, guiding N-gap filling, but if you are like me, don't really care about N-gap filling, then you are recommended to increase the identity threshold! However, perhaps instead of using this rigid way, you may use a score matrix with a slightly high threshold, like using alignments with 92-100% to generate the score matrix.

5. How to increase alignment sensitivity?

You may need to use soft-masking instead of hard-masking. LASTZ can take advantage of soft-masking by extending alignment into the soft-masked region, hence recover many useful alignments.

You can modify the `all_lastzctl` file: increase the `-ydrop` threshold for gap penalty, set `-step` to 1, and allow `-transition` for seeding patterns. A drawback of these settings is that before you know how many alignments has been recovered, you have to see the greatly increased running time and space occupation for LASTZ. It is also difficult to predict the outcome of these changes. You may need to try several times to see the effects.

You can also use a less stringent score matrix like inferred from alignments with 80-100% identity (not recommended though), or a low `-identity` threshold like 70%. Again, it is difficult to

predict the outcome of these changes. You may have to try several times to see the effects.

6. How to remove allele redundancy as much as possible?

HaploMerger2 has several designs to help remove true redundancy without making false calls.

You are recommended to feed all sequence data to HaploMerger2. HaploMerger is not only able to handle large data set, but benefit from it! A complete sequence data set is always needed for determining allele redundancy correctly and efficiently.

A whole-genome alignment scheme based on LASTZ-ChainNet is a key, which helps to accurately identify true allele pairs. You may read the advanced topic 2&3 about the balance between alignment sensitivity and specificity.

The DGA graph-based structure is another key, which accommodates full homologous/allelic relationships and permits a natural optimizing algorithm to separate true allele pairs from homologous sequence pairs. **Since this part has been optimized, normally you don't need to change anything.** but you can still lower -filterScore_2 and -scapeFilter in hm.batchB to allow removing more redundant sequences. However, lower these two parameters will increase the risk of removing non-redundant sequences (false removal).

HaploMeger uses a novel algorithm (hm.batchD) to detect tandem alleles. Three parameters -minLen, -maxInterval and -minCoverage in hm.batchD can be used to control the sensitivity and specificity of the detection. You are recommended to use this function.

HaploMerger also uses a second round of whole-genome alignment to reclaim/refine the redundant sequences (hm.batchB4). Three parameters, -identity, maskFilter, and -redundantFilter will help you to control the sensitivity and specificity. You are recommended to use this function.

7. How to avoid removing non-redundant sequences?

This topic is quite the opposite of the topic 4. **HaploMerger2 has been designed to achieve a balance on the problem.** but you can still tweak it to what you like.

You may increase the alignment specificity to suppress false calls of redundancy.

You may increase -filterScore_2 and -scapeFilter in hm.batchB to suppress false calls of redundancy, if you prefer high specificity than sensitivity.

It is worth noting that if you use the above methods to prevent false calls, then there is a risk for some true redundant sequences eluding you. Fortunately, many, if not all, of these sequences go to the unpaired.fa.gz file. You may reclaim them by using hm.batchB4. This is actual a mechanism which HaploMerger2 uses to achieve a high sensitivity without losing specificity.

Finally, you may adjust three parameters, -identity, maskFilter, and -redundantFilter in hm.batchB4 to decide how many sequence to be re-classified as “non-redundant”.

8. How to avoid falsely connecting two scaffolds?

Two scaffolds can be connected based on their overlapping alignment in order to extend the continuity of sequence. However, the overlapping alignment may be spurious. How to handle this?

First, you are recommended to feed the complete genome sequence data to HaploMerger2, which helps to fight against spurious alignments.

You may adjust the alignment specificity for this purpose.

You may increase -filterScore_2 and -scapeFilter in hm.batchB to prevent low-scored

alignments used for connection, in the cost of missing some true redundant alleles.

You may set the `--minOverlap` parameter in `hm.batchB` to force HaploMerger2 to break a connection when the alignment for connection does not meet the minimum required length. In fact, you can set this parameter to a very large value so that no scaffolds will be connected at all.

9. Can I use HaploMerger on polymorphic mulit-ploid assemblies?

Not tested.

10. Can I use HaploMerger on meta-genomic assemblies?

No.

Other Tools

1. lastz_D_Wrapper.pl

The LASTZ package includes a program called lastz_D. This executable can be used to infer score matrix specific to a given genome sequence.

I wrote this wrapper to facilitate the use of lastz_D in HaploMerger2. To infer the score matrix for a diploid genome assembly, you should first divide the genome sequence file into two parts, one contains 5~15% sequences in size, and the other contains the rest of the genome sequences. Empirically, you may select the longest scaffold sequences into the first part. Save these two parts in separate files, then run:

```
lastz_D_Wrapper.pl --target=part1.fa.gz --query=part2.fa.gz --identity=90.
```

Input fasta files can be gzip or non-gzip files. The “*--identity*” option tells lastz_D to use only the alignments whose identity is $\geq 90\%$ (excluding Ns, indels, and gaps) for inference. The higher the identity is, the more stringent the scoring matrix will be, you are recommended to try at least two stringency level: $\geq 90\%$ and $\geq 95\%$.

Empirically, if allelic difference of your genome is 4-5%, you may set it to 90% as a starting point; if difference is 3% or 1%, you may set it to 94% or 96% as a starting point.

An output file called `part1.part2.raw.time.xx_xx.q` will be generated after minutes or hours. You may open this file and copy the score matrix into the `scoreMatrix.q` file in your project directory.

Note that lastz_D may take a lot of time and cannot be parallelized. So if you have a very large genome, then you may just select 5% of sequences into the target part. The use of $>10\%$ genome sequence is not recommended, because the chance of finding no allele for a scaffold is high and hence the inference is less reliable.

Lastz_D sometimes may be stuck somewhere and provide strange results. Besides, different sequences may produce different inference. So, you may select different sequences into the `part1.fa.gz` file and rerun the program again. After that, you can choose a suitable one from several score matrices for further use.

For more information of lastz_D, please refer to the LASTZ’s manual.

2. count_scf_ctg

This executable calculates the scaffolds’ and contigs’ N05-N95 numbers and length. Run the executable with option “*--help*” for detailed usage.

If the executable does not work, enter the directory named “*installDir/bin/count_scf_ctg.source*” and run “*make*” to recompile the utility and copy it to the “*installDir/bin*” directory.

3. countN50_from_agp.pl

This perl script calculates scaffolds’ and contigs’ N50 sizes based on the AGP file. It is much faster than `count_scf_ctg`.