TASSEL 5.0 Pan-genome Atlas (PanA) pipeline documentation

Authers: Fei Lu, Jeff Glaubitz, Terry Casstevens, Qi Sun, Robert Bukowski, Katie Hyma, Ed Buckler

Affiliation: Institute for Genomic Diversity, Cornell Univsersity, Ithaca, NY 14850

Note: This is a work in progress. If you have questions, please post messages in TASSEL Google group, Also we would be happy to collaborate on some research projects, please send emails to Fei Lu (fl262@cornell.edu) for further discussions.

May 28, 2014

Table of Contents

Introduction

What can PanA do?

The Pan-genome Atlas (PanA) is a TASSEL pipeline to genetically map genomic sequence of a species to the reference genome. The large amount of high-resolution sequence tags can be used as anchor points for ongoing pan-genome constructions of many species. They can also serve as a benchmark to evaluate and direct the de novo assembly of non-reference genomes.

Principle

PanA exploits abundance of sequence data of population genomics, including reduced representation library technologies (e.g. Genotyping by Sequencing (GBS)) based data and whole genome shotgun (WGS) sequencing data, and looks for association between sequence tags and alleles. Conducting a whole genome scan between millions of tags and millions of SNP markers is computationally expensive. To speed up the calculation, the population structure/population stratification is not controlled for in association test. Instead, a machine learning (ML) model is trained on unique reference tags to solve the false positives originating from population structure. Unique and accurately mapped tags (Pan-genome anchors) are kept after ML filtering.

Design

PanA is closely related to TASSEL GBS pipeline, which is available here at http://www.maizegenetics.net/tassel/docs/TasselPipelineGBS.pdf.
Many concepts in TASSEL GBS pipeline also apply in PanA. Please find the meaning of abbreviations in TASSEL GBS pipeline documentation, if you cannot find it here. The most challenging thing in PanA is the large amount of computing time. Therefore, many tactics are used to reduce computing time, including paralleling the computation in high performance computing (HPC) clusters. As **Fig. 1** shows, the computation is paralleled in HPC.

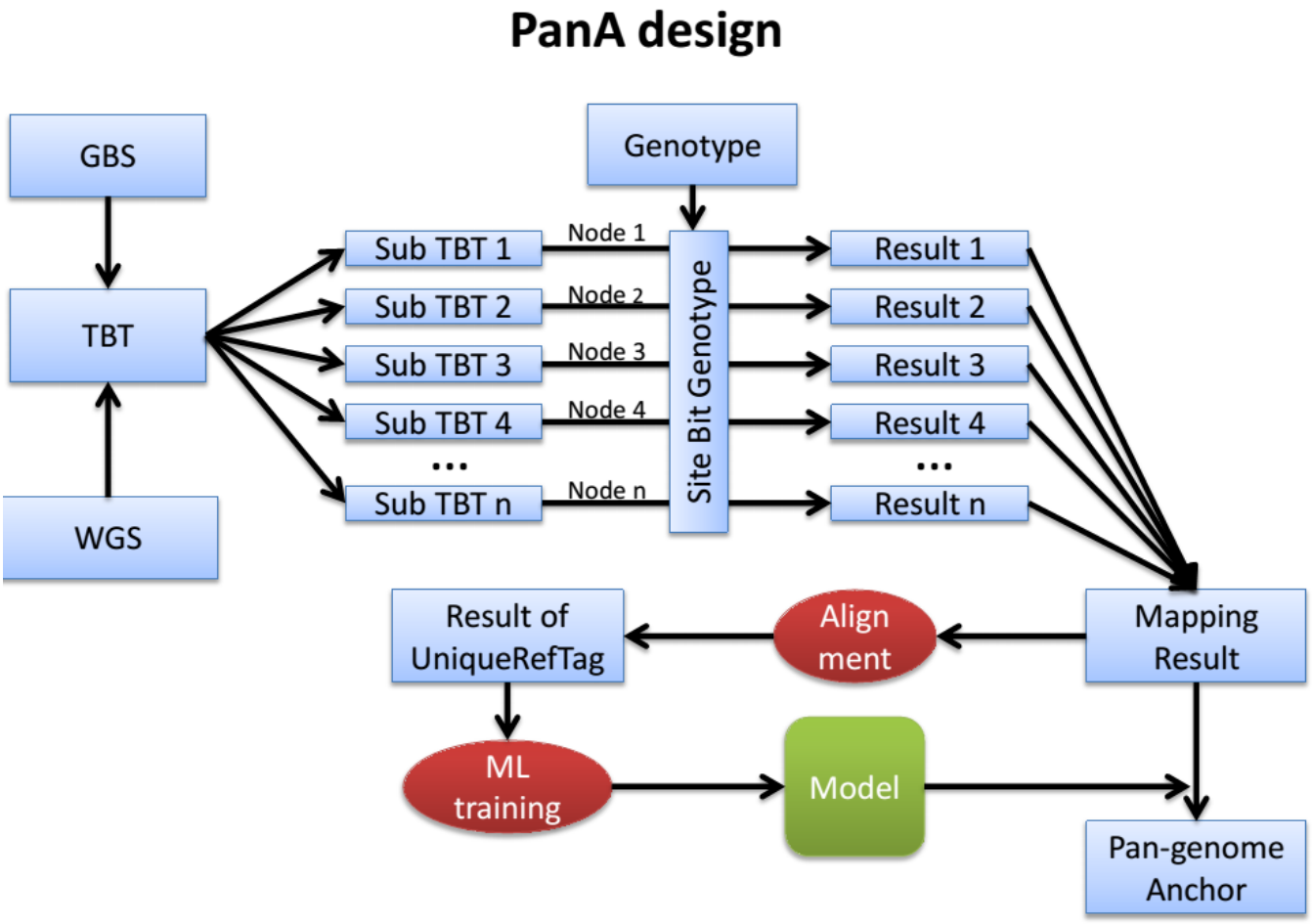


Figure 1. Design of PanA. GBS data or WGS data are processed and stored in TagsByTaxa (TBT) file. Genotype data are converted to bit set for each single site. TBT files are divided into sub TBTs for parallel computing in HPC. Genetic mapping of tags is conducted in each node on HPC. Mapping result from each nodes are merged together. By doing alignment via Bowtie 2, uniquely aligned reference tag (UniqueRefTag) are selected for machine learning training. The trained model is then used to predict mapping accuracy and filtering for accurately mapped tags, which are the final pan-genome anchors.

Prerequisites before running PanA

PanA works for both reduced representation library data, including GBS, Restriction-Associated DNA (RAD), etc, and WGS data. **Tab. 1** shows the prerequisites for running PanA.

Table 1. Prerequisites before running PanA

Data	Software/Library
A reference genome	Java runtime environment (JRE)
SNPs at genome level	R

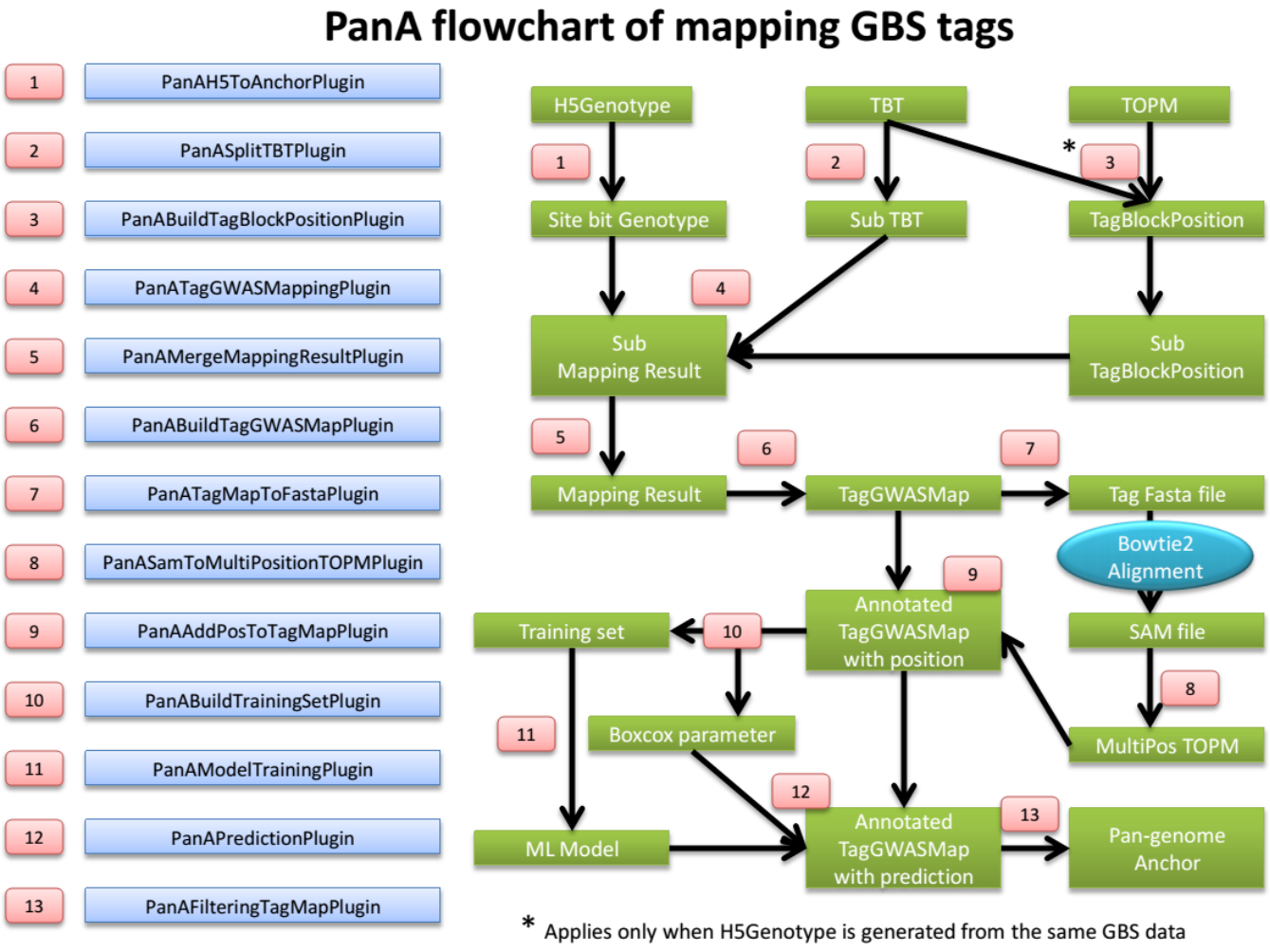
Reduced representation library data (GBS, RAD, etc) or requencing data	Weka library (http://www.cs.waikato.ac.nz/ml/weka/)

Two scenarios

There are two scenarios for PanA. One is mapping reduced representation library sequence (GBS, RAD, etc). The other is mapping sequences from WGS data. Different plugins are used in the two scenarios.

Scenario 1

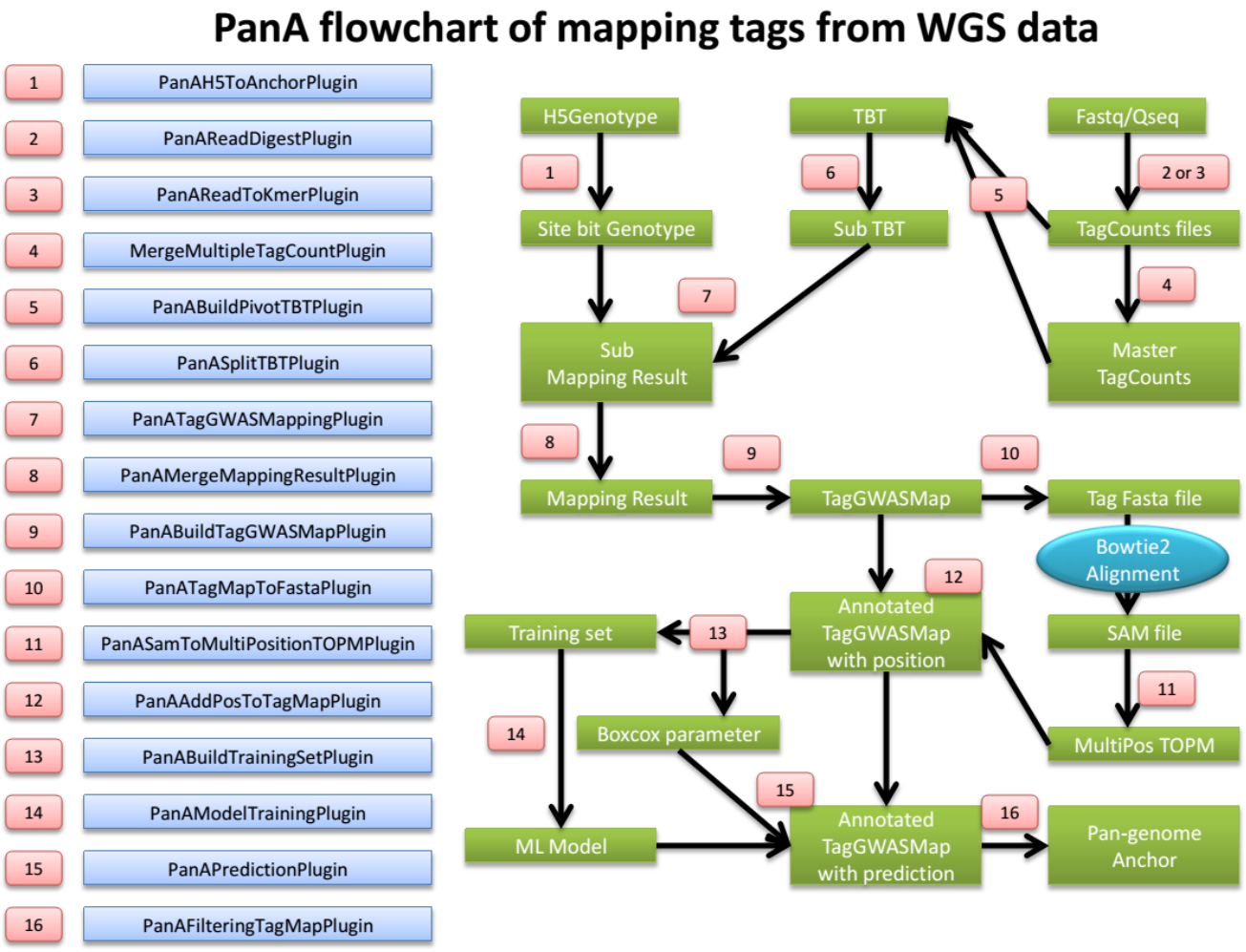
In the scenario, all the code was well tested. We already performed genetic mapping of GBS tags in 23,000 maize samples and generated 4.4M maize pan-genome anchors. About 54% anchors are within 10 Kb regions of their true positions, 95% within 1 Mb. Further analysis showed biological significance. Here are the plugins involved:



Scenario 2

Inspired by the success of genetically mapping GBS tags, we come up the idea to map sequence tags from WGS data (when there are hundreds of taxa have been shotgun sequenced). In this scenario, we use virtual digest to

sample sequence tags from Fastq or Qseq files. Using K-mers also works here, but sampling may significantly reduce computing time. K-mers in too high density may not be much more helpful than sampled sequence tags. But we also keep mapping K-mers as an option by implementing PanAReadToKmerPlugin. Using a portion of K-mers may also reduce the computing time significantly. Although the code for this scenario is well tested, it should be well noted that we have not performed any analysis based on real WGS data. Here are the plugins to map tags from WGS data:



How to run PanA?

The PanA is an extension of the Java program of TASSEL. PanA commands are run as TASSEL plugins via the command line in the following format (Linux or Mac operating system; for Windows use `run_pipeline.bat`):

```
run_pipeline.pl -fork1 -PluginName --plugin-option -endPlugin -runfork1
```

Each step of the pipeline is specified with a "fork" command and a number, since TASSEL can run several processes at once, and split and recombine their results. The fork option is followed by the name of the plugin, and any plugin-specific options. If no plugin options are provided, the program will print a list of available options. -endPlugin signals the end of plugin-specific options, and -runfork1 then runs the specified plugin. In all of our examples here for the PanA pipeline, we run only a single fork at a time.

Please see http://www.maizegenetics.net/tassel/docs/TasselPipelineCLI.pdf for general instructions on how to install the TASSEL 5.0 Standalone Build (http://www.maizegenetics.net/index.php?option=com_content&task=view&id=89&Itemid=119) on your computer. These PanA-specific instructions assume that you have unzipped the standalone into the directory (folder)

```
/programs
```

and then renamed the directory

```
/programs/tassel5.0_standalone
```

to

```
/programs/tassel
```

If not, you will have to edit the example commands appropriately (*e.g.*, replace "`tassel`" with "`tassel5.0_standalone`").

If you have more memory available on your machine than 1.5GB, then you can increase the amount of memory available to TASSEL by opening `run_pipeline.pl` (or `run_pipeline.bat`) in a text editor and modifying "`-Xms512m -Xmx1536m`" to (for example) "`-Xms4g`" (the `-Xms` option controls the amount of memory allocated to the program on startup).

The information about how to run PanA in HPC can be provided by your HPC administrator.

The latest documentation of PanA can be seen at http://www.maizegenetics.net/gbs-bioinformatics.

PanAH5ToAnchorPlugin

Summary:

Convert TASSEL5 HDF5 format genotype to HDF5 site bit (SBit) genotype, which is optimized for fast linkage disequilibrium (LD) calculation.

Input:

- TASSEL5 HDF5 gneotype file

Output:

- Site bit genotype file

Arguments:

PanAH5ToAnchorPlugin	
-i	HDF5 format genotype file
-o	site bit genotype file in SimpleGenotypeSBit format

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAH5ToAnchorPlugin -i
M:/genotype.hmp.h5 –o M:/genotype.sBit.h5
-endPlugin -runfork1
```

PanASplitTBTPlugin

Summary:

Split the big TagsByTaxa(TBT) file into sub TBTs. The TBT file should be in TagsByTaxaByteHDF5TagGroups format (See TASSEL GBS pipeline). Sub TBTs can then be submitted into nodes on HPC for genetic mapping calculation, one after another.

Input:

- TagsByTaxa file in TagsByTaxaByteHDF5TagGroups format

Output:

- Sub TBTs

Arguments:

PanASplitTBTPlugin	
-i	input TagsByTaxa(TBT) file, TagsByTaxaByteHDF5TagGroups format
-s	chunkSize, number of tags in a sub TBT. This determines the mapping calculation time usage in a node/computer. Default = 65536
-o	output directory of sub TBTs

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanASplitTBTPlugin –i
M:/Master.tbt.h5 -o M:/subTBTs/ -endPlugin -runfork1
```

PanABuildTagBlockPosPlugin

Summary:
Generate a file containing positions which should be blocked while conducting genetic mapping GBS tags. Since the anchor genotype may come from the same GBS data whose sequence tags will be mapped, mapping tags to the SNPs generated from themselves is not right. The positions information of GBS tags is stored in TOPM file. Those positions should be pulled out from TOPM and blocked. However, if you are mapping GBS Tags to genotypes from other platforms, this plugin can be ignored.

Input:
- TBT file
- TOPM file

Output:
- TagBlockPosition file

Arguments:

PanABuildTagBlockPosPlugin	
-t	input TagsByTaxa(TBT) file, TagsByTaxaByteHDF5TagGroup format
-p	input TOPM file used to generate the same genotype for mapping
-v	TOPM version value. Binary file = 1; HDF5 file = 2
-o	output tagBlockPosition file

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanABuildTagBlockPosPlugin
-t M:/Master.tbt.h5 -p M:/Master.topm.h5 –v 2 –o M:/Master.tbp.bin -
endPlugin -runfork1
```

PanASplitTagBlockPosPlugin

Summary:
Split TagBlockPosition (TBP) file into sub TBPs for parallelization on HPC.

Input:
- TBP file

Output:
- Sub TBPs

Arguments:

PanASplitTagBlockPosPlugin	
-i	input TagBlockPosition file
-s	chunkSize, number of tag positions in a sub TBP. This should extactly match number of tags in sub TBT. Default = 65536
-o	output directory of sub TBPs

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanASplitTagBlockPosPlugin
-i M:/Master.tbp.bin -o M:/subTBPs -endPlugin -runfork1
```

PanATagGWASMappingPlugin

Summary:
Conduct genetic mapping of tags on SNPs. Before large scale calculation, users can test one sub TBT for the estimation of computing time.

Input:
- SBit genotype
- TBT
- TBP

Output:
- Mapping results

Arguments:

PanATagGWASMappingPlugin	
-g	input anchor map file, SimpleGenotypeSBit format
-t	input TBT file, TagsByTaxaByteHDF5TagGroup format
-b	input TagBlockPosition file, correspongding to tags in TBT. Used to block the marker coming from the tag to be mapped. Default = null
-o	output directory
-m	minimum count when tag appear in taxa, default = 20, too low number lacks statistical power
-c	coreNum, value = max/Integer. Default:max, which means using all cores in a node, 4 threads/core. When the coreNum is set less than or equal to total core number, which means using coreNum cores, each core runs 1 thread

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 –PanATagGWASMappingPlugin –g
M:/genotype.sBit.h5 –t M:/subTBTs/pivotTBT_00000_0_65536.h5 –b M:/subTBPs/
TBP_00000_0_65536.bin –o M:/subMappingResult/ -c 16 -endPlugin -runfork1
```

PanAMergeMappingResultPlugin

Summary:
Merge mapping results. Note: please do not change file name of sub mapping results. They are in an order.

Input:
- Mapping results

Output:
- Merged mapping result

Arguments:

PanAMergeMappingResultPlugin	
-i	directory of mapping results of sub TBTs
-o	filename of merged mapping result

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanAMergeMappingResultPlugin
```

```
-i M:/subMappingResult/ -o M:/MasterMappingResult.txt -endPlugin -runfork1
```

PanABuildTagGWASMapPlugin

Summary:
Build TagGWASMap which is in HDF5 format. It incorporates mapping results and build attributes for machine learning.

Input:
- Mapping result file
- TagCounts file

Output:
- TagGWASMap file

Arguments:

PanABuildTagGWASMapPlugin	
-i	tag GWAS mapping result file
-t	tagCount file
-o	output file in TagGWASMap format

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanABuildTagGWASMapPlugin
-i M:/MasterMappingResult.txt –t M:/Master.tagCount.bin –o M:/tagMap.h5 -
endPlugin -runfork1
```

PanATagMapToFastaPlugin

Summary:
Output Fasta sequences from TagGWASMap

Input:
- TagGWASMap file

Output:
- Fasta sequence of TagGWASMap file

Arguments:

PanATagMapToFastaPlugin	
-i	TagGWASMap file
-o	output Fasta format sequence file of TagGWASMap

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanATagMapToFastaPlugin
-i M:/tagMap.h5 –o M:/tagMap.fasta.fa -endPlugin -runfork1
```

PanASamToMultiPositionTOPMPlugin

Summary:
Incorporate alignments information from Bowtie2 sam file to multiple position TOPM. Before running this

plugin, bowtie2 alignment of Fasta file from previous plugin should be performed. Options should include "-k 2 –very-sensitive-local –S tagMap.align.sam"

Input:

- TagGWASMap file
- Bowtie2 alignment sam file

Output:

- Multiple position TOPM file

Arguments:

PanASamToMultiPositionTOPMPlugin	
-i	bowtie2 alignemnt file in SAM format
-t	TagGWASMap file
-o	output multiple position TOPM file

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanASamToMultiPositionTOPMPlugin
-o M:/tagMap.h5 –i M:/tagMap.align.sam –o tagMap.multiTOPM.h5 -endPlugin -
runfork1
```

PanAAddPosToTagMapPlugin

Summary:

Annotate TagGWASMap file using multiple position TOPM file. Unique reference tags will be characterized for machine learning.

Input:

- TagGWASMap file
- Bowtie2 alignment sam file

Output:

Arguments:

PanAAddPosToTagMapPlugin	
-i	TagGWASMap file
-t	multiple position TOPM file

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAAddPosToTagMapPlugin
-i M:/tagMap.h5 –t tagMap.multiTOPM.h5 -endPlugin -runfork1
```

PanABuildTrainingSetPlugin

Summary:

Pull out unique reference tags and attributes from TagGWASMap and perform boxcox transformation for attributes. It also generates a boxcox parameter (lamda) file. The log10 value of distance between alignment position and genetic mapping position is the independent variable.

Input:

- TagGWASMap file

Output:

- Training set file
- Boxcox parameter file

Arguments:

PanABuildTrainingSetPlugin	
-m	TagGWASMap file
-t	training set file
-r	R path
-b	boxcox parameter file

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanABuildTrainingSetPlugin
-m M:/tagMap.h5 –t M:/train/training.arff –r M:/R-3.0.2/bin/Rscript –b
M:/train/boxcoxParameter.txt -endPlugin -runfork1
```

PanAModelTrainingPlugin

Summary:

Train a M5Rules model from training set. It generates a model which will be used in prediction. Also it generates two report files. One is "prediction.txt". The actual value and prediction values can be seen in this file. The other file is "accuracyTable.txt". The proportion of remaining tags and mapping resolution distribution at different cutoff can be seen in this file. Users can specify cutoff value for later filtering based on this table.

Input:

- Training set file

Output:

- Model file
- Training report files

Arguments:

PanAModelTrainingPlugin	
-t	training set file
-w	path of weka library
-m	model file
-r	directory of training report

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAModelTrainingPlugin
-t M:/train/training.arff –w M:/Weka-3-6/weka.jar –m M:/m5.mod –r
M:/train/report/ -endPlugin -runfork1
```

PanAPredictionPlugin

Summary:

Make predictions based on the trained model. Results are then written into TagGWASMap file

Input:

- TagGWASMap file
- Trained model
- Boxcox parameter file

Output:

Arguments:

PanAPredictionPlugin	
-t	TagGWASMap file
-m	trained machine learning model
-b	boxcox parameter file
-w	path of weka library

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAPredictionPlugin
-t M:/tagMap.h5 –m M:/m5.mod –b M:/train/boxcoxParameter.txt –w M:/Weka-3-
6/weka.jar -endPlugin -runfork1
```

PanAFilterTagMapPlugin

Summary:

Generate Pan-genome anchor file based on users' cutoff (bp).

Input:

- TagGWASMap file

Output:

- Anchor file

Arguments:

PanAFilteringTagMapPlugin	
-t	TagGWASMap file
-a	anchor file
-c	distance cutoff

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAFilterTagMapPlugin
-t M:/tagMap.h5 –m M:/m5.mod –a M:/anchor.txt –c 50000 -endPlugin -runfork1
```

PanAReadDigestPlugin

Summary:

Virtually digest Fastq or Qseq files into sequence fragments. Using short recognition sequence to sample genomic sequence to reduce computing time

Input:

- Fastq or Qseq files
- LaneTaxa key file (see Appendix 1)

Output:

- TagCount files

Arguments:

PanAReadDigestPlugin	
-i	input directory of Fastq or Qseq files
-f	input format value. 0 = Fastq. 1 = Qseq
-k	laneTaxa key file which links Fastq/Qseq file with samples
-s	recognition sequence for virtual digest. Default: GCTG
-l	customed tag length
-o	output directory of tag count files

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAReadDigestPlugin
-i M:/fastq/ -f 0 -k M:/laneTaxaKey.txt -s GCTG -l 80 -o M:/tagCounts/ -
endPlugin -runfork1
```

PanAReadToKmerPlugin

Summary:
Generate Kmers (tags) from Fastq or Qseq files

Input:

- Fastq or Qseq files
- LaneTaxa key file (see Appendix 1)

Output:

- TagCount files

Arguments:

PanAReadToKmerPlugin	
-i	input directory of Fastq or Qseq files
-f	input format value. 0 = Fastq. 1 = Qseq
-k	laneTaxa key file which links Fastq/Qseq file with samples
-l	customed tag length
-o	output directory of tag count files

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -PanAReadToKmerPlugin
-i M:/fastq/ -f 0 -k M:/laneTaxaKey.txt -l 80 -o M:/tagCounts/ -endPlugin -
runfork1
```

MergeMultipleTagCountPlugin

Summary:
Please see the TASSEL GBS pipeline

PanABuildPivotTBTPlugin

Summary:
Virtually digest Fastq or Qseq files into sequence fragments. Using short recognition sequence to sample genomic sequence to reduce computing time

Input:
- Mater TagCount file
- TagCount files of taxa

Output:
- Master TBT file

Arguments:

PanABuildPivotTBTPlugin	
-m	master TagCount file
-d	directory containing tagCount files
-o	output TBT

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -PanABuildPivotTBTPlugin
-m M:/Master.tagCount.bin -d M:/tagCounts/ –o M:/Master.tbt.h5 -endPlugin -
runfork1
```

Citation

Please cite the paper:

Fei Lu, Maria C Romay, Jeff C Glaubitz, Peter J Bradbury, Rob J Elshire, Tianyu Wang, Kassa Semagn, Yu Li, Edward S Buckler. High resolution mapping of sequence anchors in complex genomes: an example from maize. (*in review*)

Appendix 1: LaneTaxa Key file example

The LaneTaxa key file is formatted as tab-delimited text. You can create it from Excel if you save it as tab-delimited text. **The sample names must not contain spaces, colons (':') or underscores**. However, it is OK to include dashes, or parentheses.

Lane	Taxa
t0.fq	t0
t1.fq	t1
t2.fq	t2
t3.fq	t3
t4.fq	t4