

This tutorial describes the use of scaffhunter tools to assemble scaffolds into pseudo-molecules using markers genotyped in a population.

Go to scaffhunter folder (Scripts can be run from any folder but the command lines in this tutorial assume you are in this folder and that you have python26 version).

Available data:

data/sequence/markers.fasta is a multifasta file containing all markers sequences of markers belonging to a linkage group.

data/sequence/scaffolds.fasta is a multifasta file containing all scaffolds belonging to a linkage group.

data/pairwise/pairwise.pwd is a pairwise file generated by Joinmap that contains by markers pair, the recombination rate and LOD score.

data/pairwise/pairwise_LOD.tab is a pairwise file (that can be passed to the tools if you don't have Joinmap to generate the pairwise file).

data/genotyping/marker_data.txt is the genotyping file simulating the genotyping of the progeny of a self-fertilization of an accession of unknown parents. This dataset contains genotyping information for one chromosome and has been simulated with a genotyping error of 5%. Missing data are coded with "--". As markers have been phased, "h" correspond to one haplotype and "k" correspond to the other haplotype.

For this tutorial, we assume the markers grouping has already been performed and scaffold belonging to each linkage groups have been identified.

Scaffold attribution to linkage group can be performed by aligning markers sequence of grouped markers onto the reference scaffolds and identifying scaffolds that contains markers. This step can be performed using locOnref.py program of this package.

A – Conversion of the pairwise data into a matrix file

If you have Joinmap software:

```
python bin/JMp2matrix.py --pwd data/pairwise/pairwise.pwd --lod LOD.mat --rec REC.mat
```

When running this program, two files are generated: the LOD.mat file contains the matrix of pairwise LOD score and the REC.mat file contains the matrix of pairwise recombination rate.

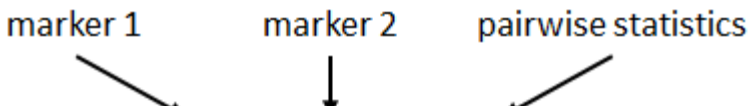
For the following treatments these two matrixes can be used, depending on the statistics you prefer to use. In this tutorial we will use the LOD matrix. If you chose to use the other matrix (recombination rate)

the **--type** argument should be set to **REC** in matrix2ortho.py and pwd2figure.py and this argument should be set to **DIF** in reorderient.py and UPGMA.py.

If you do not have Joinmap software but have a pairwise file, the matrix can be created running the following command line:

```
python bin/pwd2matrix.py --pwd data/pairwise/pairwise_LOD.tab --out LOD.mat
```

The pairwise file provided in --pwd argument should contain 3 columns (Figure 1). The two first columns gives markers 1 and two names and column 3 give the statistics between the two markers.



marker 1	marker 2	pairwise statistics
Musa1	Musa10	47.7747904
Musa1	Musa100	7.35989703
Musa1	Musa101	7.92280177
Musa1	Musa102	9.03913834
Musa1	Musa103	10.2925711
Musa1	Musa104	10.4918663
Musa1	Musa105	8.20404288
Musa1	Musa106	6.06285897
Musa1	Musa107	8.29096207
Musa1	Musa108	9.25652057

Figure 1 : Example of pairwise file format for pwd2matrix.py.

B – Positioning markers onto scaffolds

Markers can be positioned onto the scaffold running the following command line:

```
python bin/locOnRef.py --ref data/sequence/scaffolds.fasta --blast y --bwa_mem y --bow y --bow_loc y --fasta data/sequence/markers.fasta --margin 500 --index y --rmindex y --out mark_on_scaff.tab
```

The output file contains markers location on the scaffold. This file is a table file of 3 columns. The first column contains marker name, the second contains scaffold which marker belong and the third contains the position of the marker onto the scaffold. The markers in this file are ordered on marker position in scaffolds.

When looking at the standard output, in addition to several processing information, the two last lines indicates that two markers could not be located:

“Musa487”

“Musa278 scaff4 819690 scaff4 828250”

No location can be found for markers Musa487 and the different tools reported different location for markers Musa278.

C - Ordering of scaffolds using an UPGMA like method:

Now that the markers have been located on scaffolds and matrix is created. These informations will be used to group and order scaffolds one relative to the others. This can be done running the following command line:

```
python bin/UPGMA.py --scaff mark_on_scaff.tab --mat LOD.mat --out1 LOD_scaff_order.tab --out2 LOD_mark_order.tab --type IDENT
```

The IDENT value passed to --type argument indicates to the program that we are working on a resemblance statistics rather than a difference statistics.

The file LOD_scaff_order.tab contains scaffold scaffold order suggested by UPGMA.py. The file generated is a 2 column file containing in column 1 scaffold ordered and in column 2 scaffold orientation.

The second output file (LOD_mark_order.tab) contains markers ordered in the scaffold order proposed by UPGMA.py. It is a table file with in column 1 marker name and in column 2 scaffold name.

D – Optimization of scaffold order and orientation:

This step aims at refining and optimizing scaffold position and orientation calculated in UPGMA.py. The optimization is performed by calculating a score for the scaffold order, trying rearrangements (scaffold or scaffold group permutations and/or inversion) followed by score re-calculation. If the new score is better than the previous, the new order is conserved; else, the previous order is conserved. The program stops when more than a defined number of consecutive rearrangement (passed in --iter argument) do not improve the ordering.

This optimization can be done by running the following command:

```
python bin/reorderient.py --mark LOD_mark_order.tab --scaff LOD_scaff_order.tab --mat LOD.mat --out1 LOD_scaff_order_opt.tab --out2 LOD_mark_order_opt.tab --type IDENT --iter auto
```

Warning: This step can be long if you have a lot of scaffolds to order and you chose “auto” in --iter argument. In this case the number of consecutive rearrangements tested without improvement before stop is equal to: $(n - 1)^2 * 2 + n$ with n=scaffold number.

As for UPGMA.py, the IDENT value passed to --type argument indicates to the program that we are working on a resemblance statistics rather than a difference statistics.

The file LOD_scaff_order_opt.tab contains scaffold scaffold order suggested by UPGMA.py. The file generated is a 3 column file containing:

in column 1 scaffold ordered,

in column 2 scaffold orientation,

in column 3 if scaffold can be orientated in the this order : if “Ord” -> the scaffold can be orientated, if “No_Ord” -> the scaffold cannot be orientated.

The second output file (LOD_mark_order_opt.tab) contains markers ordered in the scaffold order proposed by reorderient.py. It is a table file with in column 1 marker name and in column 2 scaffold name.

As there are relatively few scaffolds, the order proposed by UPGMA.py is equivalent to the order proposed by reorderient.py. With more scaffolds and scaffolds containing less markers, this order can be refined by reorderient.py.

Ε – Markers linkage visualization:

Marker linkage between markers ordered along the ordered scaffolds can be visualized through dot-plot representation by running the following command line:

```
python bin/matrix2ortho.py --mat LOD.mat --order LOD_mark_order_opt.tab --type LOD --png  
LOD_marker_linkage.png
```

The output is a dot-plot in a png file that depict linkage between markers ordered along the provided marker order (here the order provided by reorderient.py) (Figure 2).

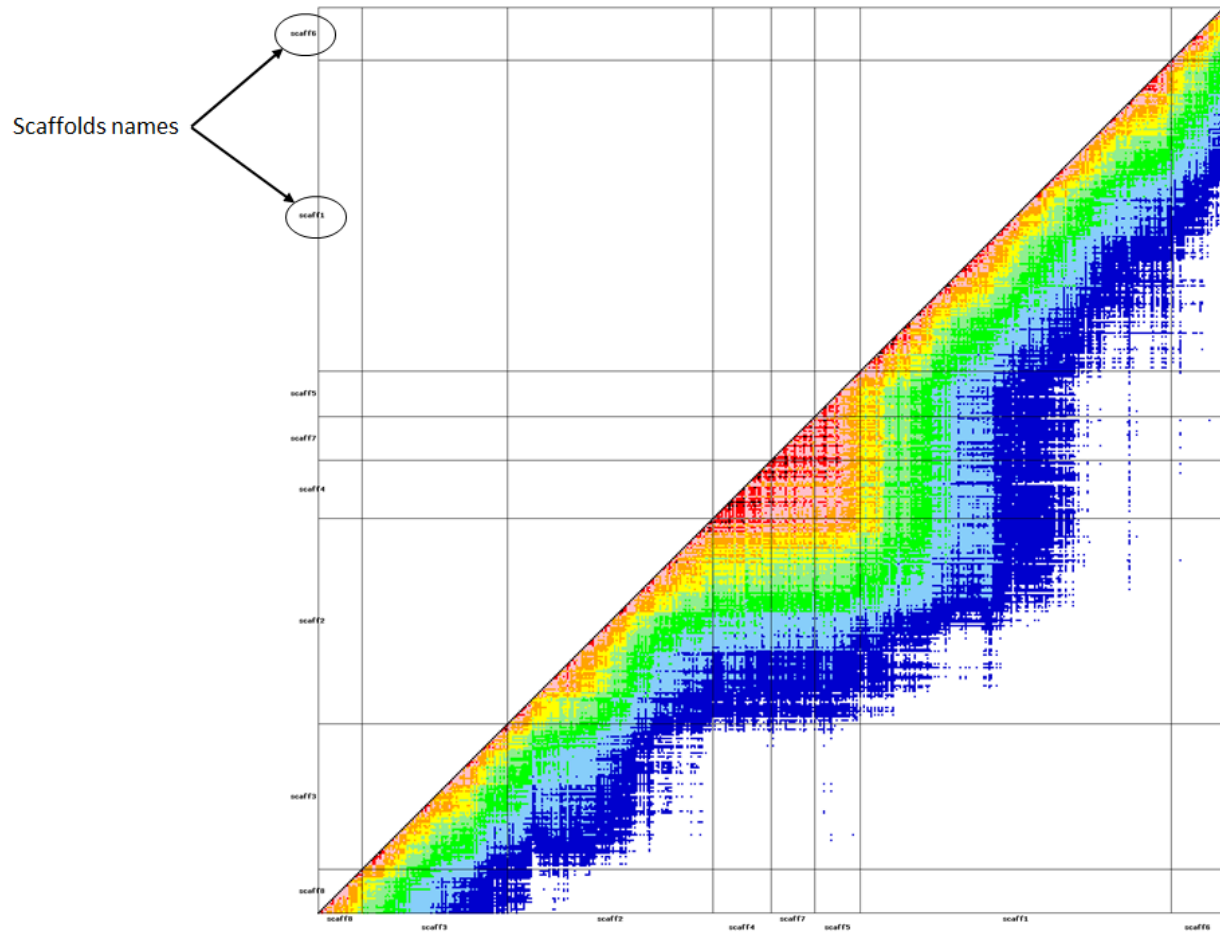


Figure 2 : Linkage between markers ordered along the provided marker order (here the order provided by reorderient.py).

The statistics (marker linkage here) is color coded and the color code of the statistics in the dot-plot is printed in the standard output and looks like:

```
[74.1164621158;82.3516245731] black
[65.8812996585;74.1164621158] red
[57.6461372012;65.8812996585] pink
[49.4109747439;57.6461372012] orange
[41.1758122865;49.4109747439] yellow
[32.9406498292;41.1758122865] lightgreen
[24.7054873719;32.9406498292] green
[16.4703249146;24.7054873719] lightskyblue
[8.23516245731;16.4703249146] blue3
```

Absence of observation of strong linkage between distant scaffolds suggests that the order proposed by reorderient.py is valid.

If you want to visualize marker linkage in a random order of scaffold you can run the following command line:

```
python bin/matrix2ortho.py --mat LOD.mat --order mark_on_scaff.tab --type LOD --png
LOD_marker_linkage_random.png
```

This command creates the dot-plot (Figure 3) with markers ordered on scaffolds order returned by locOnRef.py. The “messy” structure of the dot-plot illustrates the miss-ordering in this case.

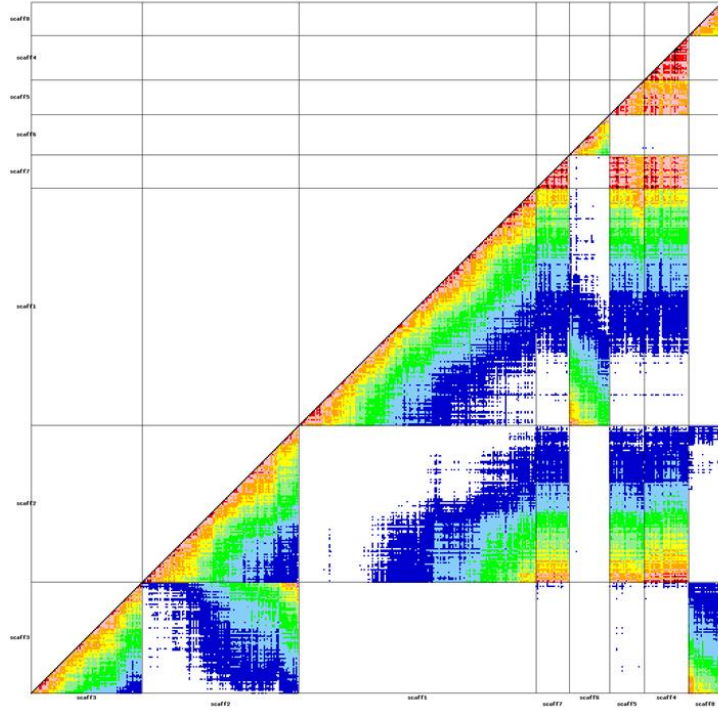


Figure 3: Linkage between markers ordered in a random order of the scaffolds.

F – Chromosome reconstruction:

Once scaffolds order has been verified, scaffolds sequence can be assembled into a pseudomolecule. This can be done in two steps:

1) Create a new file by running the following command:

```
sed 's/^/chr01\t/' LOD_scaff_order_opt.tab > LOD_chr01_scaff_order_opt.tab
```

This command line adds a new column at the beginning of LOD_scaff_order_opt.tab. This column contains the name of the future chromosome that will be constructed. This new file is recorded in LOD_chr01_scaff_order_opt.tab.

2) Create the chromosome chr01:

```
python bin/scaff2chrom.py --table LOD_chr01_scaff_order_opt.tab --seq data/sequence/scaffolds.fasta -
-out Chromosome.fasta --agp Chromosome.agp
```

This command generates two files:

- A fasta file containing the newly built sequence (and an unknown chromosome containing scaffolds that have not been grouped if any are found). In the newly build sequences scaffolds are separated by 100 'N'.
- An agp file containing scaffold order in the reconstructed sequence.

If you want to assemble more than one chromosome in one step, the LOD_chrX_scaff_order_opt.tab files can be concatenated and should look like:

```
chr01  scaff8  REV   Ord
chr01  scaff3  FWD   Ord
chr01  scaff2  REV   Ord
chr01  scaff4  REV   Ord
chr01  scaff7  FWD   Ord
chr01  scaff5  FWD   Ord
chr01  scaff1  REV   Ord
chr01  scaff6  FWD   Ord
chr02  scaffx  FWD   Ord
chr02  scaffy  REV   Ord
chr02 ...
```

G – Chromosome assembly verification:

This step is optional but it is a way to show the use of the pwd2figure.py program.

First locate markers onto the newly built reference sequence:

```
python bin/locOnRef.py --ref Chromosome.fasta --blast y --bwa_mem y --bow y --bow_loc y --fasta
data/sequence/markers.fasta --margin 500 --index y --rindex y --out mark_on_ref.tab
```

Second, draw directly the dot-plot picture by running pwd2figure.py program:

```
python bin/pwd2figure.py --pwd data/pairwise/pairwise.pwd --type LOD --order mark_on_ref.tab --png
linkage_on_ref.png --dis linkage_on_ref.dis --don linkage_on_ref.don
```

This program use the Joinmap file and the file containing markers located along the newly assembled chromosome to draw the dot-plot showing markers linkage between markers ordered along the newly assembled chromosome (Figure 4A). Two additional file are generated that can be used by Darwin software (<http://darwin.cirad.fr/>) to draw trees representing marker linkage (Figure 4B). These files contain pairwise recombination rate converted into mapping distance using the Kosambi mapping function.

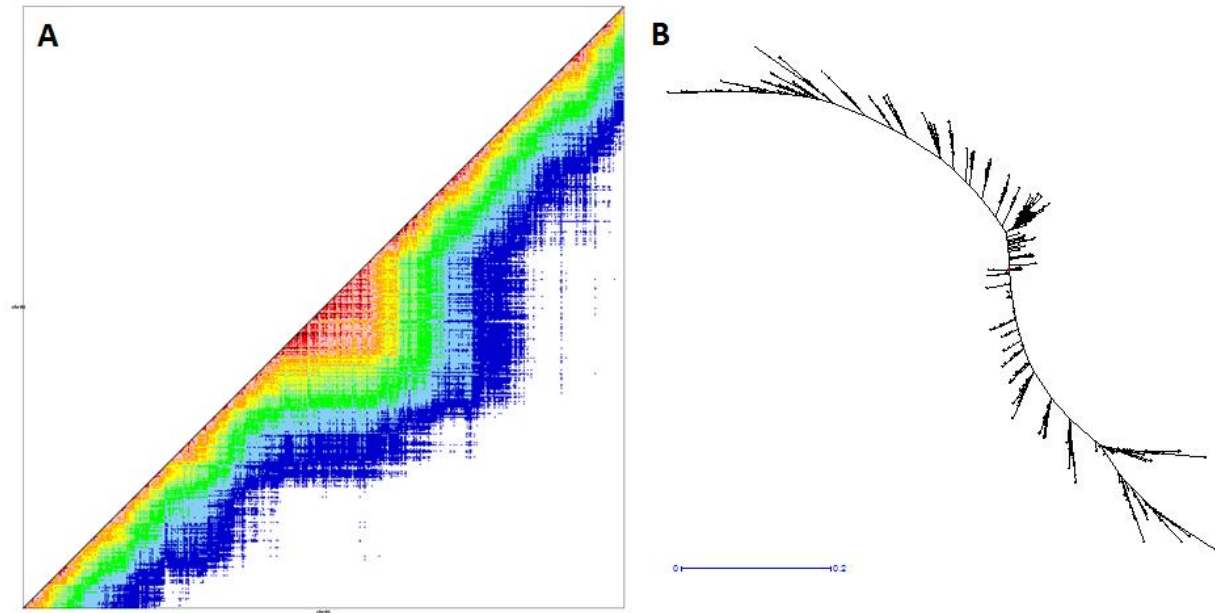


Figure 4 : Linkage between markers. **A** Markers linkage between markers ordered along the newly assembled chromosome. **B** Tree representing marker linkage built using Darwin software.

Alternatively, you can run JMpwd2matrix.py or pwd2matrix.py (depending on the pairwise information available) followed by matrix2ortho.py to obtain the same dot-plot representation. But in this case the .don and .dis file are not generated.

H – Additional tools:

Two additional programs can be found in this package.

a- **mat2submat.py** program:

When working on markers subset, the matrix used can be also reduced to this sub-set to increase calculation speed. This program can be used to create this sub-matrix. It takes in input a square matrix (--mat argument) file and create de sub-matrix containing all IDs provided in another file (--mark argument).

If you only want to observe marker linkage of scaff3. You can get the markers ID of this scaffold by running the following command line:

```
grep 'scaff3' mark_on_scaff.tab > mark_on_scaff3.tab
```

The obtained file (mark_on_scaff3.tab) contains in column 1 all markers ID belonging to scaffold3 that will be used by mat2submat.py to create the sub-matrix running the following command line:

```
python bin/mat2submat.py --mark mark_on_scaff3.tab --mat LOD.mat --out scaff3_sub_mat.tab
```

The dot-plot can be generated running the following command line:


```
python bin/matrix2ortho.py --mat scaff3_sub_mat.tab --order mark_on_scaff3.tab --type LOD --png  
LOD_marker_linkage_on_scaff3.png
```

The obtained picture representing marker linkage in scaff3 is presented in Figure5.

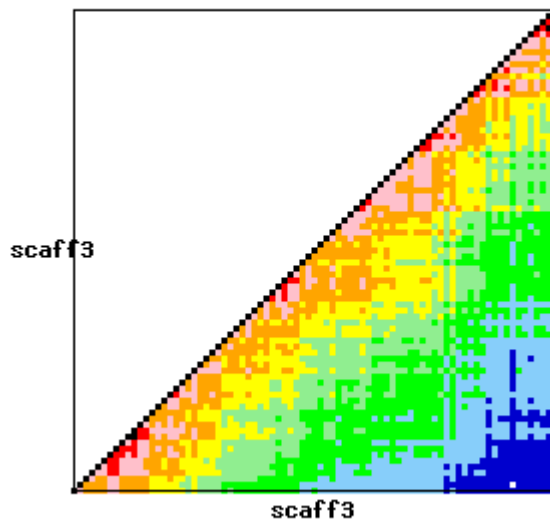


Figure 5: Dot-plot of marker linkage of markers located on scaff3.

b- GBS_corrector.py program:

This program identifies markers genotyping errors recorded in a table file based on their order (obtained from genetic map or reference sequence) provided in a table file. When constructing genetic map, these errors increase the size of genetic map, can lead to miss ordering and create markers with different profiles while they should be identical. This program identifies, correct and filter these genotyping errors. This program is based on the principle that no more than one recombination event can occur in a window of x around an observed marker (x is passed in --fen argument). If more than one recombination event is observed, a genotyping error is identified.

For example if you want to correct markers based on markers order in different scaffolds, you can run the following command line:

```
python bin/GBS_corrector.py --table data/genotyping/marker_data.txt --fen 10=3 --order  
mark_on_scaff.tab --suspect 20 --nr corrected_non_redundant_mark.tab --cor corrected_mark.tab --  
Nosu filtered_mark.tab
```

The value 10=3 passed in --fen argument indicates that only markers located on scaffolds containing **10** or more markers will be corrected/genotyping error will be identified. An error will be identified if more

than 1 recombination event is observed in a window size of [-3 ; +3] markers around the observed markers.

The 20 value passed in --suspect arguments indicate all markers having at least 20 errors detected are discarded from the output file passed in --Nosu argument (filtered_mark.tab here). This file contained filtered markers but the genotyping errors in this file have not been corrected.

The two other files contained corrected markers. In this two file, an additional line and an additional column have been added in second line and fifth column respectively and contained genotyping errors number in individual and markers respectively. As genotyping errors have been corrected, some markers may have the same profile. In the file created in the --cor argument, all corrected markers are reported, but in the file created by --nr argument, if markers belonging to the same scaffold/chromosome have the same profile, only one is reported in the file.

If you want to correct markers based on markers order in the newly constructed chromosome, you can run the following command line:

```
python bin/GBS_corrector.py --table data/genotyping/marker_data.txt --fen 10=3 --order  
mark_on_ref.tab --suspect 20 --nr ref_corrected_non_redundant_mark.tab --cor ref_corrected_mark.tab  
--Nosu ref_filtered_mark.tab
```

Arguments and output are the same as those described above.