Supplementary PDF File

All of these procedures, unless otherwise specified, should be performed on the Linux operating system. We recommend using Docker to reproduce our results. The versions of all software or packages, if not mentioned here, are listed in our main text.

1. Extraction of Simple Sequence Repeats and GO Enrichment Analysis

1.1 Environment creation for the analysis

If you wish to replicate the result in our main text, please use our Conda environment to minimize difficulties. It prepared the vast majority of Python packages and some of the softwares. However, just having the `R_environment` is not sufficient to run our R script; you need to manually install the packages of `clusterProfiler` and `argparse` . In addition to this, we have provided all dependencies that needed in our R environment for your reference. The dependency file `R_dependency.csv` is located in the same path with the conda environment yml file.

Shell

```
conda env create -f python_environment.yml
conda env create -f R_environment.yml
conda env create -f orthofinder2.yml
```

1.2 Data download and extraction of the associated data

Please download the genome and annotation files using the links provided in Table 1 of the main text. Using `gffread` to extract CDS and proteins. We used the proteins encoded by the longest CDS as representatives when annotating their functions with eggNOG-mapper.

Shell

```
conda activate variety3
gffread -g genome.fasta -x cds_output.fa annotations.gff
select_longest_cds.py -i CDS_file -o longest_cds.file
#select_longest_cds
gffread -g genome.fasta -y protein_output.fa annotations.gff
select_longest_pep.py -i pep_file -o longest_pep_file
#select_longest_pep, we use the number of sequences in this file as th
e gene count, as the GFF annotation is not entirely reliable.
```

1.3 Extracting SSRs in CDS/genomes

The file **`misa.ini`** needs to be placed in the same directory with **`mias.pl`** command. For example, if using **`mias.pl`** in the **`/Rho/pycat`** directory, then **`misa.ini`** should also be in this directory.

Please note that the two `misa.ini` files in the supplementary materials are different. At this stage, you need to use the `misa1.ini` file and manually rename it to `misa.ini`.

Shell

```
mias.pl CDS_file
ssr_type_analyze.py -i CDS_file.misa -o CDS_SSR_type
```

1.4 Analysis of SSR types on the genome and CDS

After obtaining the corresponding data, if you wish to directly use our code, please integrate the data into the following types of formats:

Plain Text

```
Species_name    type    P1    P2    P3    P4    P5    P6    c    c*
  Multiple of 3    Multiple not 3
R. ovatum    genomic    171502    86409    13818    1579    783    176
  5    23118    1982    96773    204183
R. ovatum    CDS    182    338    2658    4    5    180    152    5
  3190    334
R. simsii    genomic    178400    84275    15727    1604    949    185
  0    22940    2258    99572    208431
R. simsii    CDS    245    879    1929    8    5    147    133    11
   2618    739
```

Then simply modify the input and output files in the two scripts: Chi-square test for SSR types.py and Enrichment analysis of SSR types on CDS.py.

1.5 Annotation and Filtering of SSRs in CDS

`CDS_SSR_anno.py` takes two files as input. One is the MISA outout file for CDS, and the other is the FASTA file for CDS. The output results provide annotation information on SSRs present in each gene's CDS. This includes the CDS ID, SSR start and end sites, SSR length, the corresponding protein sequence, and the length of the CDS itself.

Plain Text

```
1 key     ssr_site        ssr_length      ssr_sequence    ssr_pep cds_len
  gth
2 Rhdel01G0001100.1       [(2955, 3022)]  68      ['(CT)14(AT)10(AC)10']
   [Seq('SLSLSLSLSIYIYIYIHTHTHT')] 3081
3 Rhdel01G0005100.1       [(174, 188)]    15      ['(CGG)5']      [Seq('G
  GGG')]   2325Rhdel01G0005500.1       [(1558, 1587)]  30      ['(GCACCA)
  5']   [Seq('APAPAPAPAP')]     3345
4 Rhdel01G0005700.1       [(38, 55), (1864, 1878)]        33      ['(CTT)
  6', '(AAG)5']    [Seq('SSSSS'), Seq('KKKKK')]    2367
```

`SSR_anno_cutoff.py` enables users to further filter the results from the previous step. In this example, the filtering criteria are a CDS length of ⩾ 600 and an SSR length of ⩾ 18. The output_file will be a single-column file containing the CDS ID or protein ID.

Shell

```
1 CDS_SSR_anno.py -ssr CDS_file.misa -cds CDS_file -o output_file
2 SSR_anno_cutoff.py -i output_file -o output_file -cds 600 -ssr 18
```

1.6 Extract specific sequences from the protein file

This code takes a single list of protein IDs as an index and uses it to process protein data.

Shell

```
1 extract_specific_pep_from_id.py -txt pep_id.txt -p longest_pep_file -o
  species_pep.fa
```

1.7 Eggmap-annoation

Since the localized deployment requires a significant amount of space, it is recommended to use the online version directly. The parameters from the web version can be referenced in our command by setting the "Taxonomic Scope" to "33090". If you need to use the standalone version, it is preferable to deploy it within a separate container and environment. For deployment (https://github.com/eggnogdb/eggnog-mapper/wiki/eggNOG-mapper-v2.1.5-to-v2.1.12), please refer to the GitHub repository: eggNOG-mapper Deployment Guide. The command we is listed bellow.

Shell

```
emapper.py --data_dir /database/eggnog_data_dir -i input_pep.fa --cpu cpu_number --mp_start_method forkserver -o output_file_name --output_dir ./ --override -m diamond --dmnd_ignore_warnings --evalue 0.001 --score 60 --pident 40 --query_cover 20 --subject_cover 20 --itype proteins --tax_scope 33090 --target_orthologs all --go_evidence non-electronic --pfam_realign none --report_orthologs --decorate_gff yes -excel
#-i input_pep_path, the longest protein file for each species.
```

To remove sequences containing illegal characters, you can use the following script and command. This script should be used before running any software that utilizes DIAMOND, including subsequent procedures.

Shell

```
pep_clean.py -i input_path -o output_path
```

1.8 GO enrichment alalysis

All GO enrichment scripts should be run within the R environment. First, you need to process the annotation results from eggNOG-mapper. For this, use the following two Python scripts `parse_go_obofile.py` and `parse_eggNOG.py`, which are available at Hua-CM's GitHub repository https://github.com/Hua-CM/HuaSmallTools/tree/master/parse.

Shell

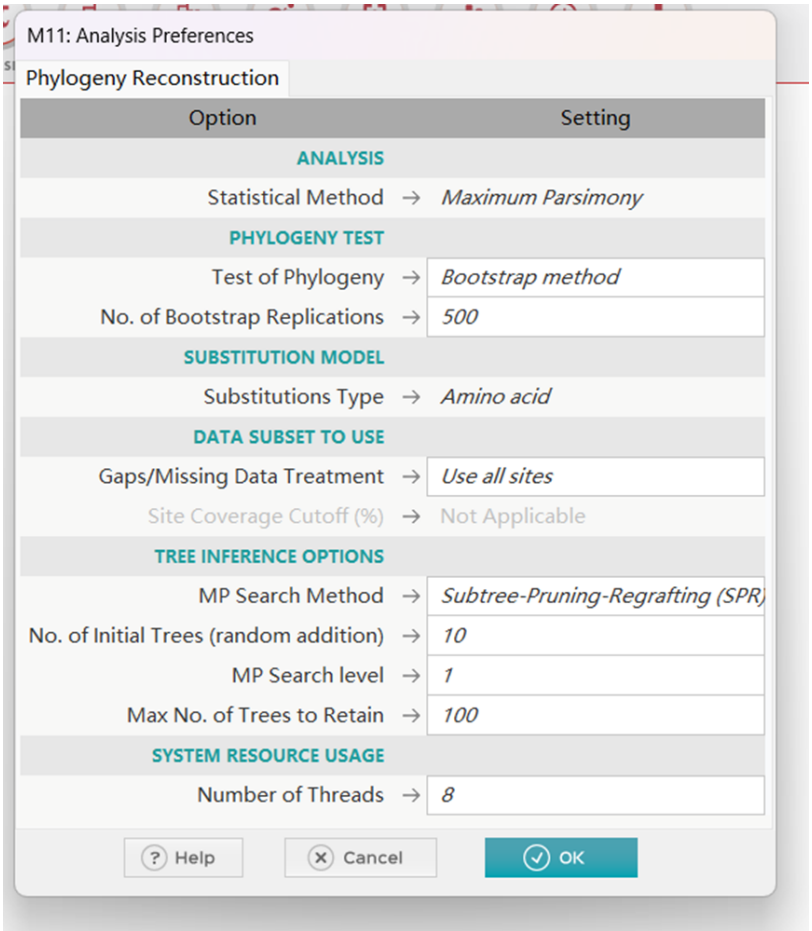
```
Conda activate R_environment
wget 'http://purl.obolibrary.org/obo/go/go-basic.obo'
parse_go_obofile.py -i go-basic.obo \
                                -o go.tb
parse_eggNOG.py -i xxxx.Eggnog.emapper.annotations -g ../go.tb -O Plants -o ./
#The input_file is the output file of eggnog-mapper. This script will generate two files, but we only need to use the GOannotation.tsv file.
go_enrichment.R -g gene_id.txt -goa GOannotation.tsv -Gotb /SSR_gene_data/go.tb -o output_file
#gene_id.txt is the file that containing Foreground pep_id, which is generate in the Supplementary_scripts1. This output file is the final enrichment results file.
```

1.9 Construction of the phylogenetic tree

Shell

```
Conda activate orthofinder2
orthofinder2 -f /SSR_gene_data/tongyong_data/Phylogenetic_tree/ -t 80
#This folder should contain the longest protein files for all species.
cd ../OrthoFinder/xxxx/Single_Copy_Orthologue_Sequences/
mafft_tree.py -i ../OrthoFinder/xxxx/Single_Copy_Orthologue_Sequences/
#This folder contains all orthologous single-copy gene families, and this script will perform MAFFT alignment on all orthologous single-copy genes.
awk '/^>/ {header++} !/^>/ {if(header % 7 == 1) {seq1 = seq1 $0} else if(header % 7 == 2) {seq2 = seq2 $0} else if(header % 7 == 3) {seq3 = seq3 $0} else if(header % 7 == 4) {seq4 = seq4 $0} else if(header % 7 == 5) {seq5 = seq5 $0} else if(header % 7 == 6) {seq6 = seq6 $0} else if(header % 7 == 0) {seq7 = seq7 $0}} END {print ">seq1\n" seq1 "\n>seq2\n" seq2 "\n>seq3\n" seq3 "\n>seq4\n" seq4 "\n>seq5\n" seq5 "\n>seq6\n" seq6 "\n>seq7\n" seq7}' *.mafft > output.mafft
#Combine the results into a superprotein.
#Up to this point, we attempted to calculate the similarity of our sequences using this script. No other software, such as BLAST, was used because the BLAST alignments are local rather than global.If you need to repeat, please modify the file names of the input and output path.
mafft_similarity.py
trimal -in output.mafft -automatedl -out
#Trim the alignment results
```

After completing all these steps, use MEGA to construct the phylogenetic tree. This process is performed on Windows. Be sure to use the Maximum Parsimony (MP) method for tree construction and conduct bootstrap analysis with 500 replicates.



settings of MEGA

2.Variety classification base on phenotype-associated SSRs

2.1 Extraction of phenotype-related SSR-containing genes

First, we need to obtain the IDs of the phenotype-related SSR-containing genes. Since we do not have the IDs for all the genes, we can start from extracting the IDs of the proteins. Using the `GOannotation.tsv` file, which annotates and parses all protein information from the enrichment analysis in the first part, and the gene ID file that meets specific conditions (CDS>=600, SSR>=18), we can obtain the pep_id corresponding to the specific genes. By leveraging the naming characteristics of the pep_id and gff files, we can derive the gene IDs. At this point, we can use bedtools to extract the corresponding genes. Use misa to detect SSRs, and then use the SSR locus information to obtain the complete SSR fragments from the gene sequence file.

2.1.1 Extraction of gene IDs

Since the second column of GOannotation.tsv contains the GO ID, we need to obtain the protein IDs based on two GO terms (flower development and leaf development). Here is an example.

```shell
awk -F'\t' '$2 == "GO:0009908" || $2 == "GO:0048366"' GOannotation.tsv > all_pep_associate_with_flower_and_leafs
awk -F'\t' '{print $1}' all_pep_associate_with_flower_and_leafs > all_pep_associate_with_flower_and_leaf_pep_id.txt
awk 'NR==FNR{a[$0]; next} $0 in a' all_pep_associate_with_flower_and_leaf_pep_id.txt Rhododendron_simsii_CDS_habor_ssr_info_ssr18_cds600_gene_id.txt >pep_id_associate_with_flowers_and_leafs_with_cds_600_ssr_18.txt
#Rhododendron_simsii_CDS_habor_ssr_info_ssr18_cds600_gene_id.txt is the Protein IDs of foreground genes used for enrichment analysis in the previous step.
```

We need to simplify the protein ID to the gene ID, otherwise it will not be recognized. We recommend using Vim's batch replacement feature directly. Here is an example:

```plain text
rna-gnl|WGS:WJXA|Rhsim01G0112900.1-------→Rhsim01G0112900
```

2.1.2 Extraction of SSR fragments

The following script will retain the "gene" lines in the GFF.

```shell
awk '$3 == "gene"' input.gff > genes_only.gff
#we need to process the GFF file to retain only the gene lines.
```

Then use the gene IDs obtained in step 2.1.1 to check this file.

```shell
awk 'NR==FNR{a[$1]; next} {for (i in a) if ($0 ~ i) print $0}' gene_ids.txt genes_only.gff > Phenotype.gff
#the "gene_ids.txt" represents the gene ID number file that can conform to the GFF file forma
```

Now we have filtered the original GFF file to retain only phenotype-related genes. Next, we will use bedtools to extract sequences.

```shell
awk -F"\t" '$3 == "gene" {print $1"\t"$4-1"\t"$5"\t"$9}' Phenotype.gff > Phenotype.bed
bedtools getfasta -fi genome.fa -bed Phenotype.bed -fo Phenotype.fa -name
misa.pl Phenotype.fa
awk '{print $1"\t"$6"\t"$7}' Phenotype.fa.misa > Phenotype.fa.misa.bed
awk -F"\t" 'BEGIN {OFS="\t"} {print $1, $2-9, $3, $4, $5, $6}' Phenotype.fa.misa.bed > Phenotype+8.fa.misa.bed
bedtools getfasta -fi Phenotype.fa -bed Phenotype+8.fa.misa.bed-fo ssr_markers_6_26.fa -name
awk '{if($0 ~ /^>/) {print $0} else {print toupper($0)}}' ssr_markers_6_26.fa > ssr_markers_6_26_upper.fa
#Extract 8 bases from one end of the SSR sequence.
```

Now the sequences are ready, use the following script to detect the presence of these fragments in the resequencing species assembly files.

2.2 Search SSRs fragment

```shell
search_fasta_sequences_in_files_6_26.py ssr_markers_6_26_upper.fa folder_containing_ assembly_files output.csv
```

The sequencing fastq was assembled using Minia. The command is listed follows:

```shell
minia -in input_file -kmer-size 31 -out-dir output_file -nb-cores 10 -nb-glue-partitions 200
```

For more information about Minia, please refer to its official documentation.

3.SSR_VDFDA

SSR_VDFDA is a tool that establishes a DNA fingerprint database based on whole genome SSR characteristics and is used to predict variety for unknown individuals. It consists of two parts: model construction and individual variety identification.

3.1SSR_VDFDA_model_build.py

This script combines three steps: 1) assembly of sequencing data using Minia, 2) extraction of SSR information using MISA, and 3) screening of SSR information and model construction. Users can optionally execute the first two steps by themselves.

3.1.1 Preparation work

The sequencing files were used for assembly and to generate SSR information files (MISA output files). Ensure that the MISA configuration file, `misa.ini`, and all other relevant files are in the same directory. Additionally, the Python command must be executed from this directory. At this stage, use the `misa2.ini` file and manually rename it to `misa.ini`.

```shell
conda activate variety3
```

3.1.2 Usage

```shell
SSR_VDFDA_model_build.py -s 1 -e 3 -pp 0.8 -index index_file -o output_path
```

The following parameters are essential:

- **-s**: Specifies the start stage.
- **-e**: Specifies the end stage.

- **-pp**: Indicates the polymorphism (as described in the main body of the paper).
- **-index**: Refers to the index file.
- **-o**: Designates the output path folder for storing the final identification model and SSR lists.

We have two index files. The first-level index file, shown in the screenshot below, contains two columns. The first column lists the variety for each individual, while the second column provides the corresponding input file for Minia, here is an example:

Plain Text

```
1 1    /Rho/data/W-1-01
2 1    /Rho/data/W-1-02
3 2    /Rho/data/W-2-01
4 2    /Rho/data/W-2-02
5 3    /Rho/data/W-3-01
6 3    /Rho/data/W-3-02
7 4    /Rho/data/W-4-01
8 4    /Rho/data/W-4-02
9 5    /Rho/data/W-5-01
10 5   /Rho/data/W-5-02
11 6   /Rho/data/W-6-01
12 6   /Rho/data/W-6-02
13 7   /Rho/data/W-7-01
14 7   /Rho/data/W-7-02
15 8   /Rho/data/W-8-01
16 8   /Rho/data/W-8-02
```

When Minia’s input includes multiple files, an index file is used to specify the input files. This is referred to as the second-level index file. For example, the second-level index file for "/Rho/data/W-1-01" is shown in the screenshot below. Since our files are paired-end sequencing files, each line in this file represents one end of a sequencing pair. If you are working with different types of files, you can modify the index file accordingly. If you have already completed the Minia assembly, you do not need the second-level index file. Here is an example:

Plain Text

```
1 /Rho/data/W-1-01_R1.fq
2 /Rho/data/W-1-01_R2.fq
```

The first-level index file is necessary under any circumstances.

Next, I will explain the output files generated by this software. For example, if your index file is named "index_file" and your polymorphism indicator is "0.8," the output files will include the following:

- all_data.csv;
- polymorphism_ssr0.8.csv;
- polymorphism_ssr_list.pkl;
- predict_model.pkl.

Shell

```
output_path/index_file_0.8/
      └── all_data.csv
      └── polymorphism_ssr0.8.csv
      └── polymorphism_ssr_list.pkl
      └── predict_model.pkl
```

3.2 SSR_VDFDA_model_predict.py

In Section 3.2, the input file used for SSR classification performance testing is polymorphism_ssr0.8.csv. This script employs an SSR identification model to predict the variety of unknown individuals.

Shell

```
SSR_VDFDA_model_predict.py -i W-1-10.contigs.fa.misa -d /out/index_file_6_27.txt_0.8 -index index_file_6_27.txt -k 37
```

- **-i**: Specifies the SSR information file for the individual whose variety is to be predicted.
- **-d**: Indicates the folder containing the KNN recognition model file and the filtered SSR list, which are the final results from the previous step.
- **-index**: Refers to the same index file used in the first step.
- **-k**: Represents the number of nearest neighbors (k) in the KNN prediction.

Please note that the prediction results are not returned directly. Instead, because the number of individuals per variety in the model may vary, the output will list the k closest points to the predicted individual based on their distance. For example, if `K=5`, the output will display the 5 nearest points to the predicted individual, sorted by distance.

3.3 Performance evaluation

3.3.1 t-SNE and K-means cluster

The script for this section has already been provided in this directory. Please remember to modify the input and output file names accordingly. And clear the parts in the individual names of the data that are unrelated to the variety information; otherwise, the script will throw an error.

For example:

"/Rho/sequence_data/Rho_SLAF/W-3-04.contigs.fa.misa" should be "W-3"

3.3.2 Cross-validation

The cross-validation process involves removing one individual at a time, using the remaining individuals to build a model, predicting the variety of the removed individual, and recording the distance. With 40 individuals, this process yields 40 cross-validation results.