Comparative chloroplast genomes analysis of nine *Limonium* (Plumbaginaceae) species, including two endangered species from the island of Malta

Dolores R. Agius & Peter Poczai

Supplemantary Figures and Tables:

Species	**Infrageneric classification**			**Native distribution****
L. aureum (L.) Hill ex Kuntze	*L*. subg. *Limonium* M	*L*. sect. *Plathymenium* B Li Ba Bo M	*L.* subsect. *Chrysanthae* B Ba Bo	Buryatiya, N & C China, Chita, Mongolia, Qinghai, Tuva
L. bicolor Kuntze	*L*. subg. *Limonium* K	L. sect. *Plathymenium* K		Mongolia to N. & E. China.
L. franchetii (Debeaux) Kuntze	*L*. subg. *Limonium* K	L. sect. *Plathymenium* K		N & SE China, Manchuria
L. otolepis (Schrenk) Kuntze	*L*. subg. *Limonium* M	*L.* sect. *Nephrophyllum* R A M *L.* sect. *Limonium* B Li	*L*. subsect. *Hyalolepideae*	Afghanistan, N & C China, Kazakhstan, Kirgizstan, Tadzhikistan, Turkmenistan, Uzbekistan, Xinjiang
L. sinense (Girard) Kuntze	*L*. subg. *Limonium* L	*L*. sect. *Plathymenium* B Ba Bo	*L.* subsect. *Chrysanthae* B Ba Bo	N & SE China, Manchuria, Nansei-shoto, Taiwan, Vietnam
L. tenellum (Turcz.) Kuntze	*L*. subg. *Limonium* L	*L*. sect. *Plathymenium* B Ba	*L*. subsect. *Rhodanthae* B Ba	Mongolia
L. tetragonum (Thunb.) Bullock	*L*. subg. *Limonium* L	L. sect. *Plathymenium* B	*L.* subsect. *Chrysanthae* B	S. Korea, S. Russian Far East, Central & S. Japan to N. Nansei-shoto, New Caledonia
L. melitense Brullo*	*L*. subg. *Limonium* K			Malta
L. zeraphae Brullo*	*L*. subg. *Limonium* K			Malta

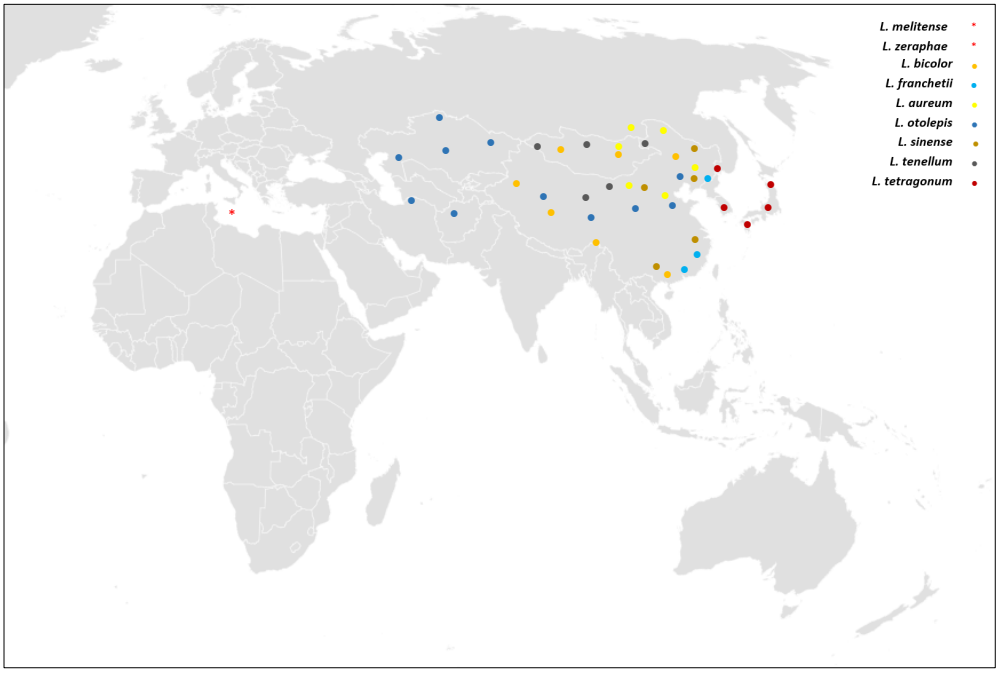
* 'Mediterranean lineage' Koutroumpa PhD thesis

** Distribution according to Plants of the World Online website except for the Maltese species

https://powo.science.kew.org/ (accessed 19th November, 2024)

K = Koutroumpa PhD thesis; L = [4]; M = [9]; B = [42-44]; Li = [45, 46]; R = [47]; A = [28]; Ba = [1]; Bo = [48]

Supplementary Table 1: The nine species of *Limonium* spp. included in this study and their infrageneric classification.



Supplementary Figure 1: Current distribution of 9 *Limonium* species included in this study as specified by Plants of the World online (accessed 13th Sept., 2024)

Commands used to in oatk to assemble the plastome of *L. melitense* and *L. zeraphae*

./oatk -k 1001 -c 100 -t 24 --nhmmscan /bin/nhmmscan -m <path>/embryophyta_mito.fam -p <path>/embryophyta_pltd.fam -o <prefix> <prefix>_hifi_reads.fa.gz

Alignment_stats_py:

```
"""
Author: Peter Poczai
Institution: University of Helsinki
Date: March 14, 2025
Description:
This script computes alignment statistics (constant sites, variable sites, entropy, and informativeness scores (Townsend 2007))
for nucleotide sequences in FASTA or PHYLIP format. It supports batch processing of multiple alignment files
from an input directory and saves results to an output directory. If the output directory is not specified,
it is automatically created as 'output_directory'. If it already exists, a new directory with a numerical suffix
(e.g., 'output_directory2', 'output_directory3', etc.) is created.

Additionally, the script generates a single PDF for violin plots and a separate PDF for box-and-whiskers plots
for the computed statistics and saves them in the output directory.

Installation:
- Requires Python 3+
- Install dependencies using:
  pip install biopython ete3 pandas numpy argparse matplotlib seaborn

Usage:
  python alignment_stats.py -I input_directory -O output_directory
"""

import argparse
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from Bio import AlignIO
from Bio.SeqUtils import GC
from Bio.Phylo.TreeConstruction import DistanceCalculator
from collections import Counter

def ensure_output_directory(base_name="output_directory"):
    """Create an output directory, adding a numerical suffix if it already exists."""
    dir_name = base_name
    count = 2
    while os.path.exists(dir_name):
        dir_name = f"{base_name}{count}"
        count += 1
    os.makedirs(dir_name)
    print(f"Output directory created: {dir_name}")
```

```
    return dir_name

def compute_informativeness(alignment):
    """Computes Townsend 2007 Informativeness Score."""
    informativeness_scores = []
    for col in range(alignment.get_alignment_length()):
        column_data = [record.seq[col] for record in alignment]
        freq_counts = Counter(column_data)
        probs = np.array(list(freq_counts.values())) / sum(freq_counts.values())
        entropy = -np.sum(probs * np.log2(probs)) if len(probs) > 1 else 0
        informativeness_scores.append(entropy)
    return np.mean(informativeness_scores)

def compute_alignment_stats(input_file):
    file_extension = input_file.split(".")[-1].lower()
    file_format = "fasta" if file_extension in ["fa", "fasta"] else "phylip"

    try:
        alignment = AlignIO.read(input_file, file_format)
    except Exception as e:
        print(f"Error reading {input_file}: {e}")
        return None

    if len(alignment) == 0:
        print(f"Empty alignment in {input_file}. Skipping...")
        return None

    num_taxa = len(alignment)
    alignment_length = alignment.get_alignment_length()

    seq_array = np.array([list(str(rec.seq)) for rec in alignment])
    gap_mask = (seq_array == '-') | (seq_array == 'N')
    gap_proportion = gap_mask.sum() / (num_taxa * alignment_length)

    gc_contents = [GC(str(rec.seq).replace("-", "").replace("N", "")) for rec in alignment if
len(str(rec.seq).replace("-", "").replace("N", "")) > 0]
    mean_gc_content = np.mean(gc_contents) if gc_contents else 0

    unique_sites = np.apply_along_axis(lambda col: len(set(col) - {'-', 'N'}), axis=0,
arr=seq_array)
    constant_sites = np.sum(unique_sites == 1)
    variable_sites = np.sum(unique_sites > 1)
    parsimony_informative_sites = np.sum(unique_sites > 2)
    singleton_sites = np.sum(unique_sites == 2)

    def compute_entropy(column):
        counts = np.array(list(Counter(column[column != '-']).values()))
        probs = counts / counts.sum() if counts.sum() > 0 else np.array([])
        return -np.sum(probs * np.log2(probs)) if len(probs) > 1 else 0
```

```
    entropy_per_site = np.array([compute_entropy(seq_array[:, i]) for i in
range(alignment_length)])
    mean_entropy = np.mean(entropy_per_site)
    informativeness_score = compute_informativeness(alignment)

    results = {
        "File": os.path.basename(input_file),
        "Alignment length": alignment_length,
        "Number of constant sites": constant_sites,
        "Number of variable sites": variable_sites,
        "Number of parsimony-informative sites": parsimony_informative_sites,
        "Number of singleton sites": singleton_sites,
        "Proportion of gaps/missing data": gap_proportion,
        "Mean GC content": mean_gc_content,
        "Mean entropy per site": mean_entropy,
        "Informativeness Score (Townsend 2007)": informativeness_score
    }

    return results

def generate_combined_plots(data, output_dir):
    metrics = [
        "Alignment length", "Number of constant sites", "Number of variable sites",
        "Number of parsimony-informative sites", "Number of singleton sites",
        "Proportion of gaps/missing data", "Mean GC content", "Mean entropy per site",
        "Informativeness Score (Townsend 2007)"
    ]

    fig, axes = plt.subplots(1, len(metrics), figsize=(len(metrics) * 4, 6))
    for i, metric in enumerate(metrics):
        sns.violinplot(y=data[metric], ax=axes[i], palette="Set2")
        axes[i].set_title(metric, rotation=45, ha="right")
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, "violin_plots.pdf"))
    plt.close()

    fig, axes = plt.subplots(1, len(metrics), figsize=(len(metrics) * 4, 6))
    for i, metric in enumerate(metrics):
        sns.boxplot(y=data[metric], ax=axes[i], palette="Set1")
        axes[i].set_title(metric, rotation=45, ha="right")
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, "box_plots.pdf"))
    plt.close()

def process_directory(input_dir, output_dir):
    output_dir = ensure_output_directory(output_dir)
    all_results = []
    for filename in os.listdir(input_dir):
        if filename.endswith((".fasta", ".fa", ".phy", ".phylip")):
            input_path = os.path.join(input_dir, filename)
```

```
            result = compute_alignment_stats(input_path)
            if result:
                all_results.append(result)

    if all_results:
        df = pd.DataFrame(all_results)
        df.to_csv(os.path.join(output_dir, "summary_statistics.csv"), index=False)
        generate_combined_plots(df, output_dir)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Compute alignment statistics from
FASTA or PHYLIP files.")
    parser.add_argument("-I", "--input_directory", required=True, help="Directory containing
input alignment files (FASTA or PHYLIP)")
    parser.add_argument("-O", "--output_directory", default="output_directory",
help="Directory to save output CSV files (default: output_directory)")

    args = parser.parse_args()
    process_directory(args.input_directory, args.output_directory)
```

Bootstrap_analysis.py:

"""
Author: Peter Poczai
Institution: University of Helsinki
Date: March 15, 2025

Description:
This script extracts and analyzes bootstrap values and SH-aLRT values from IQ-TREE .treefile output files.
It computes summary statistics and generates visualizations including histograms, density plots, violin plots, and box plots.
The script supports:
- Analyzing a single file
- Comparing up to three specific files
- Analyzing all .treefile files in a given directory

Installation:
Requires Python 3+ and the following dependencies:
```sh
pip install biopython pandas numpy argparse matplotlib seaborn
```

Usage:
For a **single .treefile file** analysis:
```sh
python bootstrap_analysis.py -i SET.treefile -O results
```

For **comparing two .treefile files**:
```sh
python bootstrap_analysis.py -i SET1.treefile -I SET2.treefile -O results
```

For **comparing three .treefile files**:
```sh
python bootstrap_analysis.py -i SET1.treefile -I SET2.treefile -J SET3.treefile -O results
```

For **analyzing all .treefile files in a directory**:
```sh
python bootstrap_analysis.py -D input_directory -O results
```

If the `results` folder already exists, the script creates a new folder `results2`, `results3`, etc.

Options:
- `-i`, `--input1` (optional) : Path to the first .treefile file.
- `-I`, `--input2` (optional) : Path to the second .treefile file (for comparison).

```
- `-J`, `--input3`  (optional) : Path to the third .treefile file (for comparison).
- `-D`, `--input_directory`  (optional) : Directory containing .treefile files to analyze.
- `-O`, `--output`  (default=`results`) : Directory to save analysis results.
- `-h`, `--help` : Show this help message and exit.
"""

import argparse
import os
import re
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('Agg')  # Ensure non-GUI backend for HPC/server use
import matplotlib.pyplot as plt
import seaborn as sns

def ensure_output_directory(base_name="results"):
    """Create an output directory, adding a numerical suffix if it already exists."""
    dir_name = base_name
    count = 2
    while os.path.exists(dir_name):
        dir_name = f"{base_name}{count}"
        count += 1
    os.makedirs(dir_name)
    print(f"Output directory created: {dir_name}")
    return dir_name

def extract_support_values(treefile):
    """Extracts bootstrap and SH-aLRT values from an IQ-TREE .treefile."""
    bootstrap_values = []
    sh_alrt_values = []

    with open(treefile, 'r') as f:
        tree_data = f.read().strip()

    match = re.search(r'\((.+);', tree_data, re.DOTALL)
    if not match:
        print(f"Error: No valid Newick tree found in {treefile}.")
        return np.array([]), np.array([])

    newick_tree = match.group(0)
    matches = re.findall(r'\)(\d+\.\d+)?/(\d+):', newick_tree)
    sh_alrt_values = [float(x[0]) if x[0] else np.nan for x in matches]  # Handle missing SH-
aLRT values
    bootstrap_values = [float(x[1]) for x in matches]

    return np.array(bootstrap_values), np.array(sh_alrt_values)

def generate_plots(df, output_dir):
    """Generates histograms, violin plots, and box plots for bootstrap and SH-aLRT values."""
```

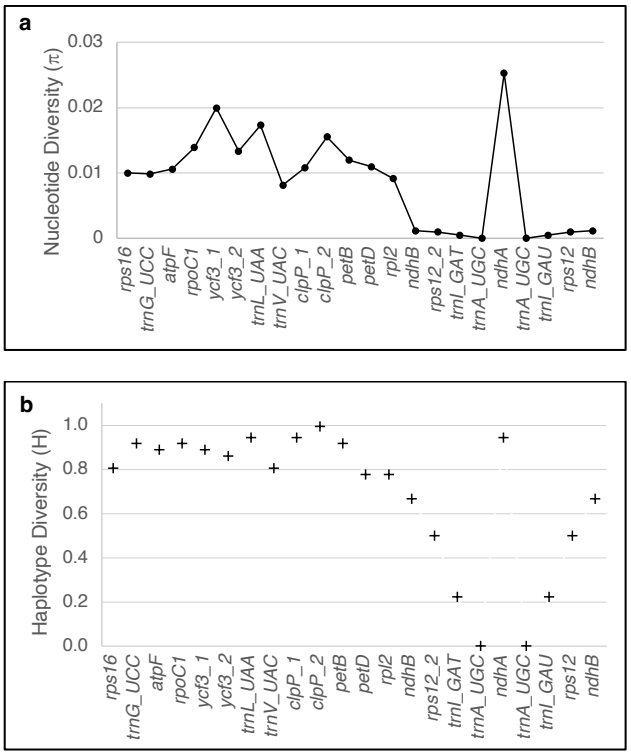
```
    if df.empty:
        print("No valid data for plotting. Skipping plots.")
        return

    plt.figure(figsize=(8, 6))
    sns.histplot(df, x="Bootstrap Value", hue="Alignment", kde=True, bins=20, alpha=0.5)
    plt.title("Histogram & Density Plot of Bootstrap Values")
    plt.savefig(os.path.join(output_dir, "bootstrap_histogram.pdf"))
    plt.close()

    plt.figure(figsize=(8, 6))
    sns.histplot(df, x="SH-aLRT Value", hue="Alignment", kde=True, bins=20, alpha=0.5)
    plt.title("Histogram & Density Plot of SH-aLRT Values")
    plt.savefig(os.path.join(output_dir, "sh_alrt_histogram.pdf"))
    plt.close()

    plt.figure(figsize=(8, 6))
    sns.violinplot(data=df, x="Alignment", y="Bootstrap Value", palette="Set2")
    plt.title("Violin Plot of Bootstrap Values")
    plt.savefig(os.path.join(output_dir, "bootstrap_violin.pdf"))
    plt.close()

    plt.figure(figsize=(8, 6))
    sns.violinplot(data=df, x="Alignment", y="SH-aLRT Value", palette="Set3")
    plt.title("Violin Plot of SH-aLRT Values")
    plt.savefig(os.path.join(output_dir, "sh_alrt_violin.pdf"))
    plt.close()

    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, x="Alignment", y="Bootstrap Value", palette="Set1")
    plt.title("Box Plot of Bootstrap Values")
    plt.savefig(os.path.join(output_dir, "bootstrap_box.pdf"))
    plt.close()

    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, x="Alignment", y="SH-aLRT Value", palette="Set3")
    plt.title("Box Plot of SH-aLRT Values")
    plt.savefig(os.path.join(output_dir, "sh_alrt_box.pdf"))
    plt.close()

def analyze_support(files, output_dir):
    """Analyzes bootstrap and SH-aLRT values and generates plots."""
    support_data = []

    for file in files:
        bootstrap_values, sh_alrt_values = extract_support_values(file)
        if len(bootstrap_values) > 0:
            support_data.append(pd.DataFrame({
                "Bootstrap Value": bootstrap_values,
                "SH-aLRT Value": sh_alrt_values,
```

```
                "Alignment": os.path.basename(file)
            }))

    if not support_data:
        print("Error: No support values found in the provided .treefile files. Exiting.")
        return

    df = pd.concat(support_data, ignore_index=True)
    summary_stats = df.groupby("Alignment")[["Bootstrap Value", "SH-aLRT
Value"]].describe()
    summary_stats.to_csv(os.path.join(output_dir, "support_summary.csv"))
    generate_plots(df, output_dir)
    print("Analysis complete. Results saved in", output_dir)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input1", help="Path to the first .treefile file")
    parser.add_argument("-I", "--input2", help="Path to the second .treefile file")
    parser.add_argument("-J", "--input3", help="Path to the third .treefile file")
    parser.add_argument("-D", "--input_directory", help="Directory containing .treefile files")
    parser.add_argument("-O", "--output", default="results", help="Output directory")
    args = parser.parse_args()
    output_dir = ensure_output_directory(args.output)
    files_to_analyze = [f for f in [args.input1, args.input2, args.input3] if f]
    if args.input_directory:
        files_to_analyze.extend([os.path.join(args.input_directory, f) for f in
os.listdir(args.input_directory) if f.endswith(".treefile")])
    analyze_support(files_to_analyze, output_dir)

if __name__ == "__main__":
    main()
```

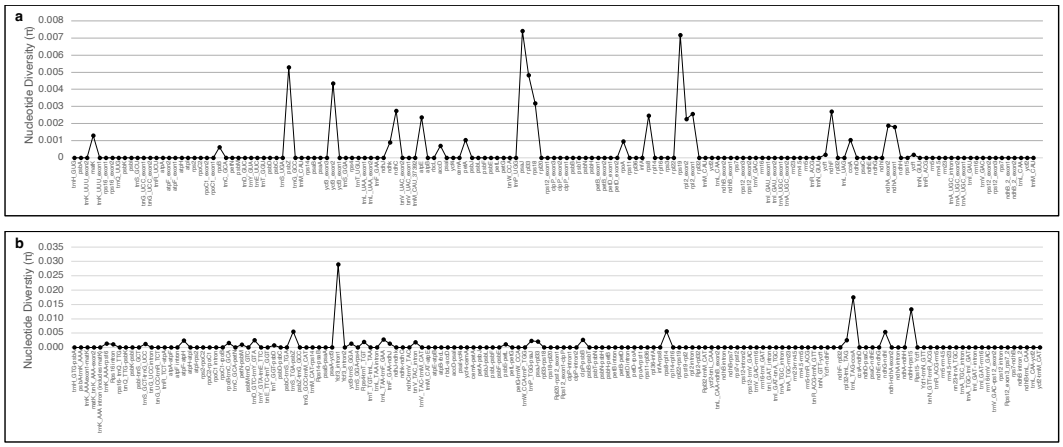
Family	Genus	Specific name	NCBI number
Frankeniaceae	*Frankenia*	*laevis*	NC_041277.1
Frankeniaceae	*Frankenia*	*pulverulenta*	NC_041278.1
Plumbaginaceae	*Ceratostigma*	*griffithii*	NC_071213.1
Plumbaginaceae	*Ceratostigma*	*minus*	NC_071214.1
Plumbaginaceae	*Ceratostigma*	*plumbaginoides*	NC_071212.1
Plumbaginaceae	*Ceratostigma*	*ulicinum*	NC_071211.1
Plumbaginaceae	*Ceratostigma*	*willmottianum*	NC_041261.1
Plumbaginaceae	*Limonium*	*melitense*	PP968847
Plumbaginaceae	*Limonium*	*zeraphae*	PQ274899
Plumbaginaceae	*Limonium*	*aureum*	NC_045399.1
Plumbaginaceae	*Limonium*	*bicolor*	NC_059915.1
Plumbaginaceae	*Limonium*	*franchetii*	NC_085143.1
Plumbaginaceae	*Limonium*	*otolepis*	NC_065861.1
Plumbaginaceae	*Limonium*	*tenellum*	NC_041279.1
Plumbaginaceae	*Limonium*	*tetragonum*	NC_059914.1
Plumbaginaceae	*Limonium*	*sinense*	MN599096.1
Plumbaginaceae	*Plumbago*	*auriculata*	NC_041245.1
Plumbaginaceae	*Plumbago*	*zeylanica*	NC_084458.1
Polygonaceae	*Atraphaxis*	*bracteata*	NC_059952.1
Polygonaceae	*Atraphaxis*	*decipiens*	NC_070100.1
Polygonaceae	*Atraphaxis*	*irtyschensis*	NC_070099.1
Polygonaceae	*Atraphaxis*	*spinosa*	NC_070098.1
Polygonaceae	*Bistorta*	*emodi*	NC_066665.1
Polygonaceae	*Bistorta*	*macrophylla*	NC_068876.1
Polygonaceae	*Bistorta*	*ochotensis*	NC_082240.1
Polygonaceae	*Bistorta*	*officinalis*	NC_065784.1
Polygonaceae	*Bistorta*	*paleacea*	NC_082267.1
Polygonaceae	*Calligonum*	*aphyllum*	NC_049137.1
Polygonaceae	*Calligonum*	*arborescens*	NC_049140.1
Polygonaceae	*Calligonum*	*bakuense*	NC_062619.1
Polygonaceae	*Calligonum*	*caput-medusae*	NC_049141.1
Polygonaceae	*Calligonum*	*colubrinum*	NC_049142.1
Polygonaceae	*Coccoloba*	*uvifera*	NC_068873.1
Polygonaceae	*Fagopyrum*	*dibotrys*	NC_037705.1
Polygonaceae	*Fagopyrum*	*esculentum*	NC_064334
Polygonaceae	*Fagopyrum*	*gracilipes*	NC_082243.1
Polygonaceae	*Fagopyrum*	*leptopodum*	NC_056984.1
Polygonaceae	*Fagopyrum*	*luojishanense*	NC_037706.1
Polygonaceae	*Fallopia*	*aubertii*	NC_068872.1
Polygonaceae	*Fallopia*	*convolvulus*	NC_082245.1
Polygonaceae	*Fallopia*	*dentatoalata*	NC_082246.1
Polygonaceae	*Fallopia*	*dumetorum*	NC_082247.1
Polygonaceae	*Fallopia*	*multiflora*	NC_041239.1
Polygonaceae	*Knorringia*	*sibirica*	NC_082248.1
Polygonaceae	*Koenigia*	*alpina*	NC_066667.1
Polygonaceae	*Koenigia*	*cyanandra*	NC_066668.1
Polygonaceae	*Koenigia*	*divaricata*	NC_066669.1
Polygonaceae	*Koenigia*	*forrestii*	NC_066670.1
Polygonaceae	*Koenigia*	*lichiangensis*	NC_066672.1
Polygonaceae	*Muehlenbeckia*	*australis*	NC_059029.1
Polygonaceae	*Muehlenbeckia*	*axillaris*	NC_059030.1
Polygonaceae	*Muehlenbeckia*	*complexa*	NC_066815.1
Polygonaceae	*Muehlenbeckia*	*gracillima*	NC_059031.1
Polygonaceae	*Muehlenbeckia*	*gunnii*	NC_059032.1
Polygonaceae	*Oxyria*	*digyna*	NC_082249.1
Polygonaceae	*Oxyria*	*sinensis*	NC_032031.1
Polygonaceae	*Persicaria*	*amphibia*	NC_071233.1
Polygonaceae	*Persicaria*	*bungeana*	NC_082250.1
Polygonaceae	*Persicaria*	*capitata*	NC_073007.1

Polygonaceae	*Persicaria*	*chinensis*	NC_050358.1
Polygonaceae	*Persicaria*	*criopolitana*	NC_079578.1
Polygonaceae	*Polygonum*	*ajanense*	NC_066666.1
Polygonaceae	*Polygonum*	*argyrocoleon*	NC_082264.1
Polygonaceae	*Polygonum*	*aviculare*	NC_058892.1
Polygonaceae	*Polygonum*	*cognatum*	NC_082265.1
Polygonaceae	*Polygonum*	*cuspidatum*	NC_057435.1
Polygonaceae	*Pteroxygonum*	*denticulatum*	NC_068877.1
Polygonaceae	*Pteroxygonum*	*giraldii*	NC_082272.1
Polygonaceae	*Reynoutria*	*japonica*	NC_059800.1
Polygonaceae	*Rheum*	*alexandrae*	NC_061617.1
Polygonaceae	*Rheum*	*delavayi*	NC_063092.1
Polygonaceae	*Rheum*	*nobile*	NC_046506.1
Polygonaceae	*Rheum*	*officinale*	NC_058627.1
Polygonaceae	*Rheum*	*palmatum*	NC_027728.1
Polygonaceae	*Rumex*	*acetosa*	NC_042390.1
Polygonaceae	*Rumex*	*hypogaeus*	NC_050054.1
Polygonaceae	*Rumex*	*nepalensis*	NC_057504.1
Polygonaceae	*Rumex* subgen. *Acetosa*	*hastatus*	NC_050928.1
Polygonaceae	*Triplaris*	*americana*	NC_068874.1
Tamaricaceae	*Myricaria*	*bracteata*	NC_088075.1
Tamaricaceae	*Myricaria*	*elegans*	NC_063470.1
Tamaricaceae	*Myricaria*	*laxiflora*	NC_072270.1
Tamaricaceae	*Myricaria*	*paniculata*	NC_041270.1
Tamaricaceae	*Myricaria*	*prostrata*	NC_046761.1
Tamaricaceae	*Reaumuria*	*songarica*	NC_041273.1
Tamaricaceae	*Reaumuria*	*trigyna*	NC_041265.1
Tamaricaceae	*Tamarix*	*androssowii*	NC_084254.1
Tamaricaceae	*Tamarix*	*aphylla*	NC_084252.1
Tamaricaceae	*Tamarix*	*arceuthoides*	NC_067397.1
Tamaricaceae	*Tamarix*	*chinensis*	NC_040943.1
Tamaricaceae	*Tamarix*	*gracilis*	NC_084253.1
OUTGROUP			
Aizoaceae	*Tetragonia*	*tetragonoides*	NC_036991.1
Barbeuiaceae	*Barbeuia*	*madagascariensis*	NC_041301.1
Talinaceae	*Talinum*	*paniculatum*	NC_037748.1
Montiaceae	*Calandrinia*	*eremaea*	NC_041259.1
Montiaceae	*Calandrinia*	*granulifera*	NC_041260.1

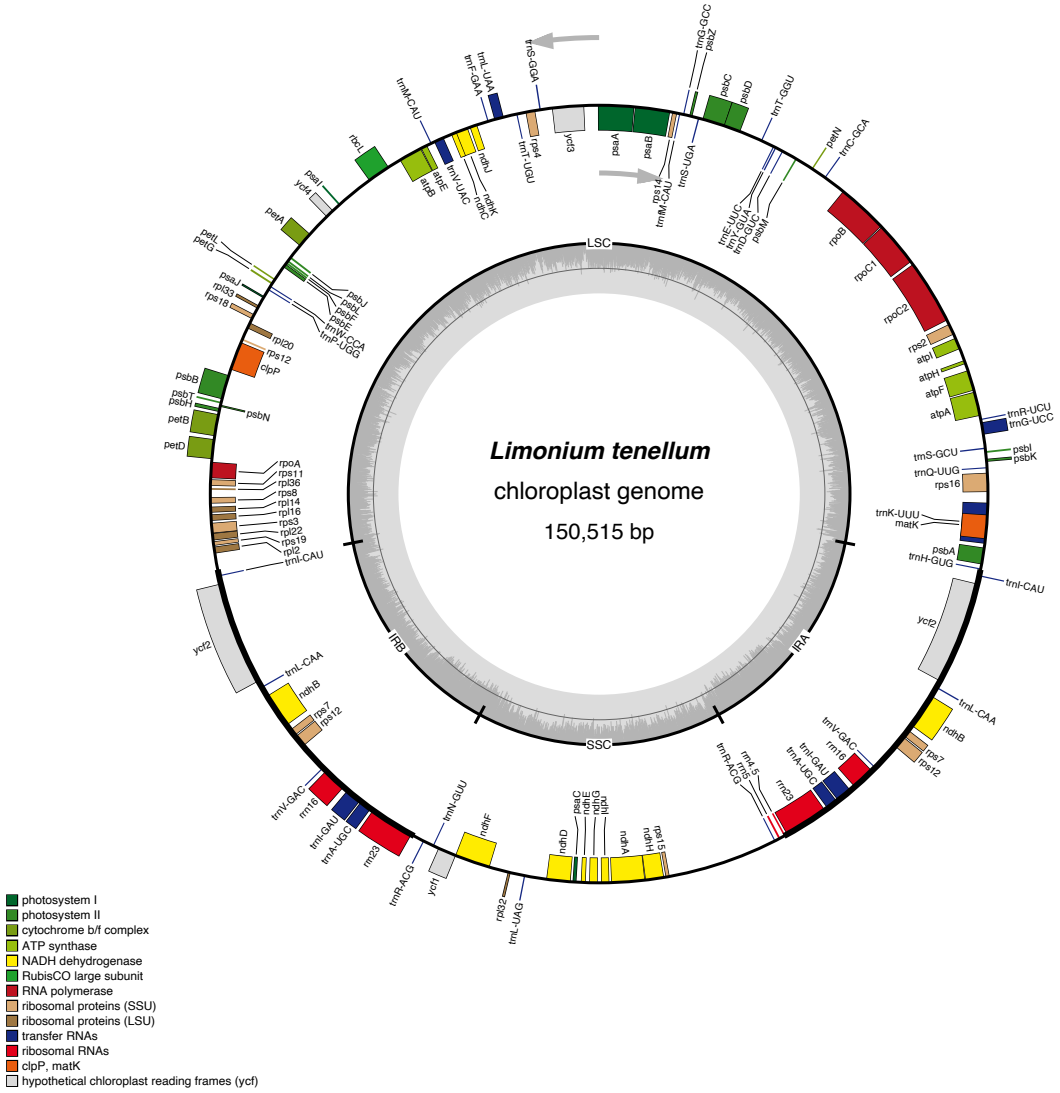
Supplementary Table 2: List of species included in the phylogenomic study and IR boundary study.



Supplementary Figure 2: (A) Nucleotide diversity and (B) Haplotype diversity for the introns of 9 *Limonium* species



Supplementary Figure 3: *L. melitense* and *L. zeraphae* (A) genic and (B) intergenic nucleotide diversity



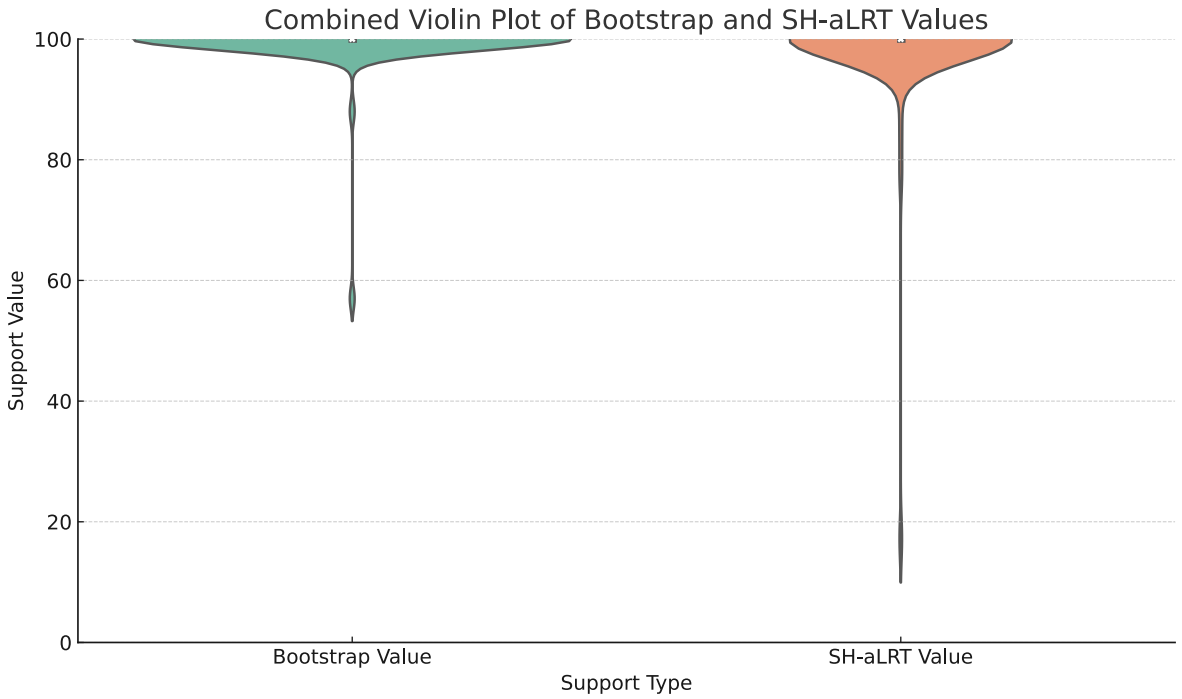
Supplementary Figure 4: Plastome map of *L. tenellum* showing SSC region

Supplementary table 3: Simple sequence repeats (SSRs) in the plastomes of 9 *Limonium* species											
		L. melitense	*L. zeraphae*	*L. bicolor*	*L. franchetii*	*L. aureum*	*L. otolepis*	*L. sinense*	*L. tenellum*	*L. tetragonum*	
SSRs	Mono	34	33	32	33	37	39	43	33	34	
	Di	5	5	7	7	7	7	11	7	6	
	Tri	1	1	0	0	0	0	0	0	0	
	Tetra	0	0	0	0	0	0	0	0	0	
	Penta	0	0	0	0	0	0	0	0	0	
	Hexa	0	0	0	0	0	0	0	0	0	
Total SSRs		**40**	**39**	**39**	**40**	**44**	**46**	**54**	**40**	**40**	**382**

Supplementary Table 4: Number of direct repeats, separated into size ranges, in the plastomes of 9 *Limonium* species.									
	L. melitense	*L. zeraphae*	*L. bicolor*	*L. franchetii*	*L. aureum*	*L. otolepis*	*L. sinense*	*L. tenellum*	*L. tetragonum*
20-29	65	68	77	91	83	69	165	71	81
30-39	12	11	17	18	17	18	22	14	18
40-59	6	7	3	7	9	7	32	11	9
60-79				0	2	2	16	1	
80-99				1			6		
100-200							35		
200-300							4		
300-400							6		
>400							7		
Total	**83**	**86**	**97**	**117**	**111**	**96**	**293**	**97**	**108**

Supplementary Table 5: Number of palindromic repeats, separated into size ranges, in the plastomes of 9 *Limonium* species.									
	L. melitense	*L. zeraphae*	*L. bicolor*	*L. franchetii*	*L. aureum*	*L. otolepis*	*L. sinense*	*L. tenellum*	*L. tetragonum*
20-29	59	55	69	88	76	64	116	90	71
30-39	14	14	22	22	21	21	23	26	22
40-59	4	4	7	6	6	7	11	12	6
60-79	0	0				2	6	0	
80-99	0	0					0	1	
100-200	0	0					14	1	
200-300	1	1					2		
300-400							0		
>400							3		
Total	**78**	**74**	**98**	**116**	**103**	**94**	**175**	**130**	**99**

Supplementary Table 6: Number of supermaximal repeats, separated into size ranges, in the plastomes of 9 *Limonium* species									
	L. melitense	*L. zeraphae*	*L. bicolor*	*L. franchetii*	*L. aureum*	*L. otolepis*	*L. sinense*	*L. tenellum*	*L. tetragonum*
20-29	50	50	39	46	44	41	54	34	43
30-39	8	7	11	12	11	8	16	8	12
40-59	4	6	6	5	6	3	16	8	6
60-79			1	0	1	2	11	1	
80-99				1			3		
100-200							35		
200-300							4		
300-400							6		
>400							7		
Total	**62**	**63**	**57**	**64**	**62**	**54**	**152**	**51**	**61**



Supplementary Figure 5: Violin plots of the bootstrap and SH-aLRT values for the phylogenomic tree