

Text S1. Msprime code used to simulate each demography.

Skov simulation:

```
#!/usr/bin/python
```

```
import msprime as msp
```

```
import time
```

```
import numpy as np
```

```
import sys
```

```
import math
```

```
import tskit
```

```
_, chrom = sys.argv
```

```
# Generation time, mutation rate
```

```
gen_time = 29.0
```

```
mu = 1.25e-8
```

```
# Population sizes from the YAML model
```

```
N_Anc = 23275
```

```
N_Nea1 = 3400
```

```
N_Nea2 = 3400
```

```
N_Asia_early = 749
```

```
N_Asia_late = 7264
```

```
N_Asia_bottleneck=1305
```

```
N_Asia_present = 5054
```

```
N_YRI = 27122
```

```
N_ancestral_Neanderthals = 13249
```

```

# Time points (in generations)

T_anc_split = 2484

T_nea_start = 22652

T_admixture = 1896

T_Asia_bottleneck = 1600.0


T_Neanderthals = 130036 / gen_time # This gives ~4484 generations


# Calculate growth rate for Asia population
# From N_Asia_late to N_Asia_present over T_Asia_growth generations


# Number of samples

n_YRI = 500

n_Asia = 200

n_Nea1 = 2

n_Nea2 = 2

t_Nea = 2330.0


samples = [

    msp.SampleSet(n_Nea1, population='Neanderthal1', time=t_Nea),
    msp.SampleSet(n_Nea2, population='Neanderthal2', time=t_Nea),
    msp.SampleSet(n_YRI, population='YRI'),
    msp.SampleSet(n_Asia, population='Asia'),
]


demography = msp.Demography()


# Add ALL populations first

```

```
demography.add_population(name='YRI', initial_size=N_YRI)
demography.add_population(name='Asia', initial_size=N_Asia_present)
demography.add_population(name='Neanderthal1', initial_size=N_Nea1)
demography.add_population(name='Neanderthal2', initial_size=N_Nea2)
demography.add_population(name='Anc', initial_size=N_Anc)
demography.add_population(name='Neanderthal_anc',
initial_size=N_ancestral_Neanderthals)
demography.add_population(name='Common_anc', initial_size=N_Anc)
```

```
demography.add_population_parameters_change(
    time=0, initial_size=N_Asia_present, growth_rate=0,
    population='Asia')
```

```
demography.add_population_parameters_change(
    time=T_Asia_bottleneck-100, initial_size=N_Asia_bottleneck, growth_rate=0,
    population='Asia')
```

Neanderthal admixture into Asia (from Neanderthal1)

```
demography.add_population_parameters_change(
    time=T_Asia_bottleneck, initial_size=N_Asia_late, growth_rate=0,
    population='Asia')
```

Neanderthal admixture into Asia (from Neanderthal1)

```
demography.add_mass_migration(
    time=T_admixture, source='Asia', dest='Neanderthal1', proportion=0.0225)
```

```
demography.add_population_parameters_change(
    time=T_anc_split-100, initial_size=N_Asia_early, growth_rate=0,
```

```

population='Asia')

# YRI and Asia split from Anc
demography.add_population_split(
    time=T_anc_split, derived=['YRI', 'Asia'], ancestral='Anc')

demography.add_population_split(
    time=T_Neanderthals, derived=['Neanderthal1', 'Neanderthal2'],
    ancestral='Neanderthal_anc')

# Neanderthal ancestor splits from Anc
demography.add_population_split(
    time=T_nea_start, derived=['Neanderthal_anc', 'Anc'], ancestral='Common_anc')

demography.sort_events()

map_file = 'genetic_map_GRCh37_chr{}.txt'.format(chrom)
recomb_map=msp.RateMap.read_hapmap(map_file)

start = time.time()
ts = msp.sim_ancestry(
    samples = samples,
    demography = demography,
    recombination_rate = recomb_map,
    record_migrations=True,
    random_seed = 123
)

```

```

ts_mutated=msp.sim_mutations(ts, rate=mu)

end = time.time()

print("Finish simulating chromosome {}. Time: {}".format(chrom, end - start))

'''
Tag introgressed segments
'''

seq_len = ts.get_sequence_length()
start = time.time()
ME_ids = ts.get_samples(1)
de_seg = {i: [] for i in ME_ids}
ar_seg = {i: [] for i in ME_ids}
for mr in ts.migrations():
    if mr.source == 1 and mr.dest == 2:
        for tree in ts.trees(leaf_lists=True):
            if mr.left > tree.get_interval()[0]:
                continue
            if mr.right <= tree.get_interval()[0]:
                break
            for l in tree.leaves(mr.node):
                if l in ME_ids:
                    # print(l)
                    de_seg[l].append(tree.get_interval())

def combine_segs(segs, get_segs = False):
    merged = np.empty([0, 2])

```

```

if len(segs) == 0:
    if get_segs:
        return([])
    else:
        return(0)
sorted_segs = segs[np.argsort(segs[:, 0]), :]
for higher in sorted_segs:
    if len(merged) == 0:
        merged = np.vstack([merged, higher])
    else:
        lower = merged[-1, :]
        if higher[0] <= lower[1]:
            upper_bound = max(lower[1], higher[1])
            merged[-1, :] = (lower[0], upper_bound)
        else:
            merged = np.vstack([merged, higher])
if get_segs:
    return(merged)
else:
    return(np.sum(merged[:, 1] - merged[:, 0])/seq_len)

```

```

true_de_prop = [combine_segs(np.array(de_seg[i])) for i in sorted(de_seg.keys())]
true_de_segs = [combine_segs(np.array(de_seg[i]), True) for i in sorted(de_seg.keys())]
print("Denisovan ancestry: ", true_de_prop)

```

```

for i, s in enumerate(true_de_segs):
    np.savetxt('truth/SkovSimNeanderthal_chr{}_hap{}_de.txt'.format(chrom, i), s, fmt =
"%0.4f", delimiter = "\t")

```

```
ts_mutated.dump('res/SkovSimNeanderthal_chr{}.tree'.format(chrom))
```

```
out='obs/SkovSimNeanderthal_chr{}.vcf'.format(chrom)
```

```
with open(out, "w") as vcf_file:
```

```
    ts_mutated.write_vcf(vcf_file, contig_id=chrom)
```

Gower simulation:

```
#!/usr/bin/python
```

```
import msprime as msp
```

```
import time
```

```
import numpy as np
```

```
import sys
```

```
import math
```

```
import tskit
```

```
_, chrom = sys.argv
```

```
# Generation time, mutation rate
```

```
gen_time = 29.0
```

```
mu = 1.25e-8
```

```
# Population sizes
```

```
N_Anc = 18500
```

```
N_Nea1 = 3400
```

N_Nea2 = 3400

N_CEU_early = 1080

N_CEU_late = 1450

N_CEU_present = 13377

N_YRI = 27000

N_ancestral_Neanderthals = 13249

Time points (in generations)

T_anc_split = 2265

T_nea_start = 18965

T_admixture = 1896

T_ceu_growth = 1100.0

T_Neanderthals = 130036 / gen_time # This gives ~4484 generations

From N_CEU_late to N_CEU_present over T_ceu_growth generations

growth_rate_ceu = math.log(N_CEU_present / N_CEU_late) / T_ceu_growth

Number of samples

n_YRI = 500

n_CEU = 200

n_Nea1 = 2

n_Nea2 = 2

t_Nea = 2330.0

samples = [

msp.SampleSet(n_Nea1, population='Neanderthal1', time=t_Nea),

msp.SampleSet(n_Nea2, population='Neanderthal2', time=t_Nea),

msp.SampleSet(n_YRI, population='YRI'),


```
msp.SampleSet(n_CEU, population='CEU'),  
]
```

```
demography = msp.Demography()
```

```
demography.add_population(name='YRI', initial_size=N_YRI)
```

```
demography.add_population(name='CEU', initial_size=N_CEU_present)
```

```
demography.add_population(name='Neanderthal1', initial_size=N_Nea1)
```

```
demography.add_population(name='Neanderthal2', initial_size=N_Nea2)
```

```
demography.add_population(name='Anc', initial_size=N_Anc)
```

```
demography.add_population(name='Neanderthal_anc',  
initial_size=N_ancestral_Neanderthals)
```

```
demography.add_population(name='Common_anc', initial_size=N_Anc)
```

```
demography.add_population_parameters_change(  
    time=0, initial_size=N_CEU_present, growth_rate=growth_rate_ceu,  
    population='CEU')
```

```
demography.add_population_parameters_change(  
    time=T_ceu_growth, initial_size=N_CEU_late, growth_rate=0,  
    population='CEU')
```

```
# Neanderthal admixture into CEU (from Neanderthal1)
```

```
demography.add_population_parameters_change(  
    time=T_admixture, initial_size=N_CEU_early, growth_rate=0,  
    population='CEU')
```

```

# Neanderthal admixture into CEU (from Neanderthal1)
demography.add_mass_migration(
    time=T_admixture, source='CEU', dest='Neanderthal1', proportion=0.0225)

# YRI and CEU split from Anc
demography.add_population_split(
    time=T_anc_split, derived=['YRI', 'CEU'], ancestral='Anc')

demography.add_population_split(
    time=T_Neanderthals, derived=['Neanderthal1', 'Neanderthal2'],
    ancestral='Neanderthal_anc')

# Neanderthal ancestor splits from Anc
demography.add_population_split(
    time=T_nea_start, derived=['Neanderthal_anc', 'Anc'], ancestral='Common_anc')

demography.sort_events()

map_file = 'genetic_map_GRCh37_chr{}.txt'.format(chrom)
recomb_map=msp.RateMap.read_hapmap(map_file)

start = time.time()
ts = msp.sim_ancestry(
    samples = samples,
    demography = demography,
    recombination_rate = recomb_map,
    record_migrations=True,
    random_seed = 123

```

```

)

ts_mutated=msp.sim_mutations(ts, rate=mu)

end = time.time()

print("Finish simulating chromosome {}. Time: {}".format(chrom, end - start))

'''

Tag introgressed segments

'''

seq_len = ts.get_sequence_length()

start = time.time()

ME_ids = ts.get_samples(1)

de_seg = {i: [] for i in ME_ids}

ar_seg = {i: [] for i in ME_ids}

for mr in ts.migrations():

    if mr.source == 1 and mr.dest == 2:

        for tree in ts.trees(leaf_lists=True):

            if mr.left > tree.get_interval()[0]:

                continue

            if mr.right <= tree.get_interval()[0]:

                break

            for l in tree.leaves(mr.node):

                if l in ME_ids:

                    # print(l)

                    de_seg[l].append(tree.get_interval())

def combine_segs(segs, get_segs = False):

```

```

merged = np.empty([0, 2])
if len(segs) == 0:
    if get_segs:
        return([])
    else:
        return(0)
sorted_segs = segs[np.argsort(segs[:, 0]), :]
for higher in sorted_segs:
    if len(merged) == 0:
        merged = np.vstack([merged, higher])
    else:
        lower = merged[-1, :]
        if higher[0] <= lower[1]:
            upper_bound = max(lower[1], higher[1])
            merged[-1, :] = (lower[0], upper_bound)
        else:
            merged = np.vstack([merged, higher])
if get_segs:
    return(merged)
else:
    return(np.sum(merged[:, 1] - merged[:, 0])/seq_len)

```

```

true_de_prop = [combine_segs(np.array(de_seg[i])) for i in sorted(de_seg.keys())]
true_de_segs = [combine_segs(np.array(de_seg[i]), True) for i in sorted(de_seg.keys())]
print("Denisovan ancestry: ", true_de_prop)

```

```

for i, s in enumerate(true_de_segs):

```

```
np.savetxt('truth/GowerSimNeanderthal_chr{}_hap{}_de.txt'.format(chrom, i), s, fmt =
"%0.4f", delimiter = "\t")
```

```
ts_mutated.dump('res/GowerSimNeanderthal_chr{}.tree'.format(chrom))
```

```
out='obs/GowerSimNeanderthal_chr{}.vcf'.format(chrom)
```

```
with open(out, "w") as vcf_file:
```

```
    ts_mutated.write_vcf(vcf_file, contig_id=chrom)
```

stdpopsim Papuan simulation:

```
#!/usr/bin/python
```

```
import msprime as msp
```

```
import time
```

```
import numpy as np
```

```
import sys
```

```
import math
```

```
import tskit
```

```
import pandas as pd
```

```
_, chrom = sys.argv
```

```
# Generation time, mutation rate
```

```
gen_time = 29.0
```

```
mu = 1.25e-8
```

Population sizes

N_YRI = 27122

N_Asia_present = 8834 # updated present-day size

N_Asia_bottleneck = 243 # updated bottleneck size

N_Anc = 23275

N_Nea1 = 3400

N_Nea2 = 3400

N_ancestral_Neanderthals = 13249

Denisovan parameters

N_Den1 = 13249

N_Den2 = 5083

N_Den_Nea_anc = 13249 # ancestral size for Denisovan+Neanderthal

N_Den_anc = 100 # ancestral size for Den1/Den2 before split

Time points (in generations)

T_anc_split = 2484

T_nea_start = 22652

T_admixture_nea = 1896

T_Asia_present = 0

T_Neanderthals = 130036 / gen_time # ~4484 generations

Denisovan times

T_den1_den2_split = 282750 / gen_time # ~9750 generations

T_den_nea_split = 437610 / gen_time # ~15090 generations

T_denisovan_admixture = 45700 / gen_time # ~1576 generations

Number of samples

```
n_YRI = 500
```

```
n_Asia = 200
```

```
n_Nea1 = 2
```

```
n_Nea2 = 2
```

```
n_Den1 = 2
```

```
n_Den2 = 2
```

```
t_Nea = 2330.0
```

```
t_Den = 2330.0
```

```
samples = [
```

```
    msp.SampleSet(n_Nea1, population='Neanderthal1', time=t_Nea),
```

```
    msp.SampleSet(n_Nea2, population='Neanderthal2', time=t_Nea),
```

```
    msp.SampleSet(n_Den1, population='Denisovan1', time=t_Den),
```

```
    msp.SampleSet(n_Den2, population='Denisovan2', time=t_Den),
```

```
    msp.SampleSet(n_YRI, population='YRI'),
```

```
    msp.SampleSet(n_Asia, population='Asia'),
```

```
]
```

```
demography = msp.Demography()
```

```
# Add populations
```

```
demography.add_population(name='YRI', initial_size=N_YRI)
```

```
demography.add_population(name='Asia', initial_size=N_Asia_present)
```

```
demography.add_population(name='Neanderthal1', initial_size=N_Nea1)
```

```
demography.add_population(name='Neanderthal2', initial_size=N_Nea2)
```

```
demography.add_population(name='Denisovan1', initial_size=N_Den1)
```

```
demography.add_population(name='Denisovan2', initial_size=N_Den2)
```

```
demography.add_population(name='Anc', initial_size=N_Anc)
```

```

demography.add_population(name='Neanderthal_anc',
initial_size=N_ancestral_Neanderthals)

demography.add_population(name='Den_Nea_anc', initial_size=N_Den_Nea_anc)

demography.add_population(name='Den_anc', initial_size=N_Den_anc)

demography.add_population(name='Common_anc', initial_size=N_Anc)


# Asia population size changes

demography.add_population_parameters_change(
    time=T_Asia_present, initial_size=N_Asia_present, growth_rate=0, population='Asia')

demography.add_population_parameters_change(
    time=T_anc_split-100, initial_size=N_Asia_bottleneck, growth_rate=0,
population='Asia')


# Denisovan admixture into Asia (from Denisovan1)

demography.add_mass_migration(
    time=T_denisovan_admixture, source='Asia', dest='Denisovan1', proportion=0.04)


# Neanderthal admixture into Asia (from Neanderthal1)

demography.add_mass_migration(
    time=T_admixture_nea, source='Asia', dest='Neanderthal1', proportion=0.0225)


# YRI and Asia split from Anc

demography.add_population_split(
    time=T_anc_split, derived=['YRI', 'Asia'], ancestral='Anc')


# Neanderthal populations split

demography.add_population_split(
    time=T_Neanderthals, derived=['Neanderthal1', 'Neanderthal2'],
ancestral='Neanderthal_anc')

```



```

# Denisovan populations split
demography.add_population_split(
    time=T_den1_den2_split, derived=['Denisovan1', 'Denisovan2'], ancestral='Den_anc')

# Denisovan and Neanderthal ancestor split
demography.add_population_split(
    time=T_den_nea_split, derived=['Neanderthal_anc', 'Den_anc'],
    ancestral='Den_Nea_anc')

# Den+Nea ancestor splits from Anc
demography.add_population_split(
    time=T_nea_start, derived=['Den_Nea_anc', 'Anc'], ancestral='Common_anc')

demography.sort_events()

map_file = 'genetic_map_GRCh37_chr{}.txt'.format(chrom)
recomb_map=msp.RateMap.read_hapmap(map_file)

start = time.time()
ts = msp.sim_ancestry(
    samples = samples,
    demography = demography,
    recombination_rate = recomb_map,
    record_migrations=True,
    random_seed = 123
)

```

```

ts_mutated=msp.sim_mutations(ts, rate=mu)

end = time.time()

print("Finish simulating chromosome {}. Time: {}".format(chrom, end - start))

'''

Tag introgressed segments
'''

seq_len = ts.get_sequence_length()

start = time.time()

# Get sample IDs for Asians
import numpy as np
import pandas as pd
import time

# Get sample IDs for Asians
ASIAN_ids = ts.get_samples(1)

# Prepare dictionaries for segment storage
nea_seg = {i: [] for i in ASIAN_ids}
den_seg = {i: [] for i in ASIAN_ids}

# Loop over migrations to record introgressed segments
for mr in ts.migrations():

    # Neanderthal1 into Asia
    if mr.source == 1 and mr.dest == 2:

        for tree in ts.trees(leaf_lists=True):

```

```

    if mr.left > tree.get_interval()[0]:
        continue
    if mr.right <= tree.get_interval()[0]:
        break
    for l in tree.leaves(mr.node):
        if l in ASIAN_ids:
            nea_seg[l].append(tree.get_interval())

# Denisovan1 into Asia
if mr.source == 1 and mr.dest == 4:
    for tree in ts.trees(leaf_lists=True):
        if mr.left > tree.get_interval()[0]:
            continue
        if mr.right <= tree.get_interval()[0]:
            break
        for l in tree.leaves(mr.node):
            if l in ASIAN_ids:
                den_seg[l].append(tree.get_interval())

# Function to merge overlapping segments and compute proportions
def combine_segs(segs, get_segs=False):
    merged = np.empty([0, 2])
    if len(segs) == 0:
        return [] if get_segs else 0
    sorted_segs = segs[np.argsort(segs[:, 0]), :]
    for higher in sorted_segs:
        if len(merged) == 0:
            merged = np.vstack([merged, higher])
    else:

```

```

    lower = merged[-1, :]
    if higher[0] <= lower[1]:
        upper_bound = max(lower[1], higher[1])
        merged[-1, :] = (lower[0], upper_bound)
    else:
        merged = np.vstack([merged, higher])
    if get_segs:
        return merged
    else:
        return np.sum(merged[:, 1] - merged[:, 0]) / seq_len

# Calculate proportions and get segments
true_nea_prop = [combine_segs(np.array(nea_seg[i])) for i in sorted(nea_seg.keys())]
true_den_prop = [combine_segs(np.array(den_seg[i])) for i in sorted(den_seg.keys())]
true_nea_segs = [combine_segs(np.array(nea_seg[i]), True) for i in
sorted(nea_seg.keys())]
true_den_segs = [combine_segs(np.array(den_seg[i]), True) for i in
sorted(den_seg.keys())]

print("Neanderthal ancestry: ", true_nea_prop)
print("Denisovan ancestry: ", true_den_prop)

# Combine all segments into a single DataFrame
all_segments = []

# Add Neanderthal segments
for hap_idx, segments in enumerate(true_nea_segs):
    for seg in segments:
        all_segments.append({

```

```
        'hap': hap_idx,
        'archaic': 'Neanderthal',
        'start': seg[0],
        'end': seg[1],
        'length': seg[1] - seg[0]
    })
```

```
# Add Denisovan segments
```

```
for hap_idx, segments in enumerate(true_den_segs):
```

```
    for seg in segments:
```

```
        all_segments.append({
            'hap': hap_idx,
            'archaic': 'Denisovan',
            'start': seg[0],
            'end': seg[1],
            'length': seg[1] - seg[0]
        })
```

```
# Create DataFrame and save to file
```

```
df_segments = pd.DataFrame(all_segments)
```

```
df_segments = df_segments.sort_values(['hap', 'start']) # Sort by haplotype then
position
```

```
# Save to single file
```

```
df_segments.to_csv('truth/Asia_archaic_segments_chr{}.txt'.format(chrom),
                   sep='\t', index=False, float_format='%.4f')
```

```
print(f"Saved {len(df_segments)} segments to combined file")
```

```
print("First few rows:")
```

```
print(df_segments.head())
```

```
# Save tree sequence and VCF as before
```

```
ts_mutated.dump('res/Asia_denisovan_Neanderthal_chr{}.tree'.format(chrom))
```

```
out = 'obs/Asia_denisovan_Neanderthal_chr{}.vcf'.format(chrom)
```

```
with open(out, "w") as vcf_file:
```

```
    ts_mutated.write_vcf(vcf_file, contig_id=chrom)
```