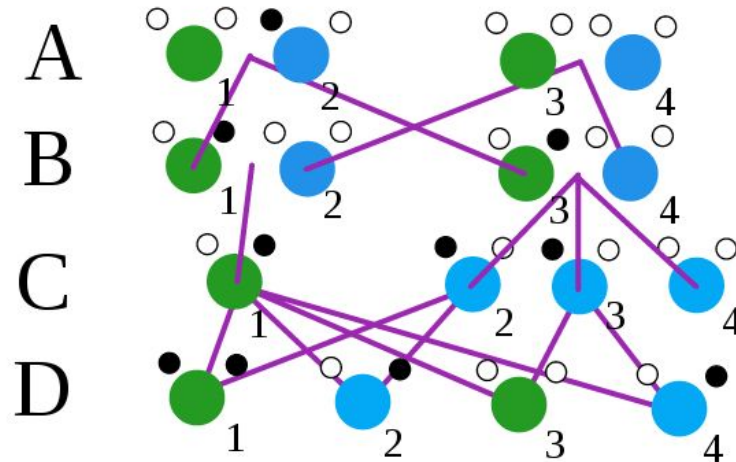
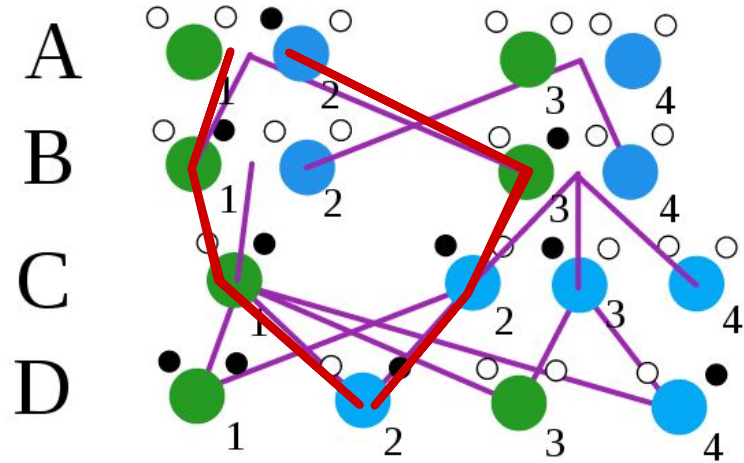


The effect of **natural selection** on **relatedness** in randomly mating populations.



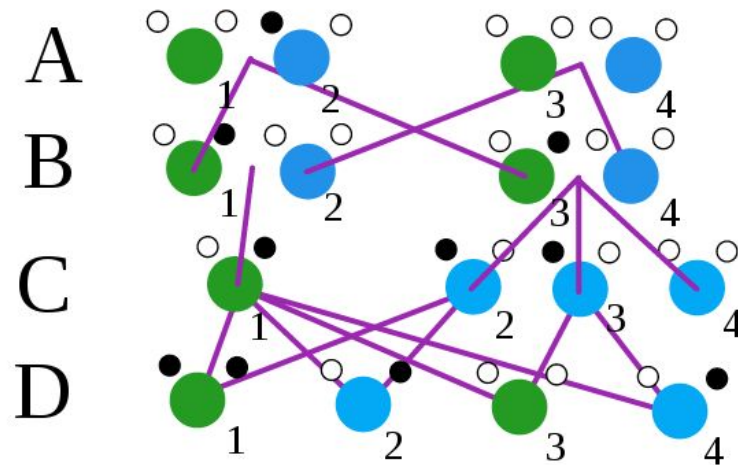
Family tree not a tree...



It's a DAG!

Measuring Relatedness

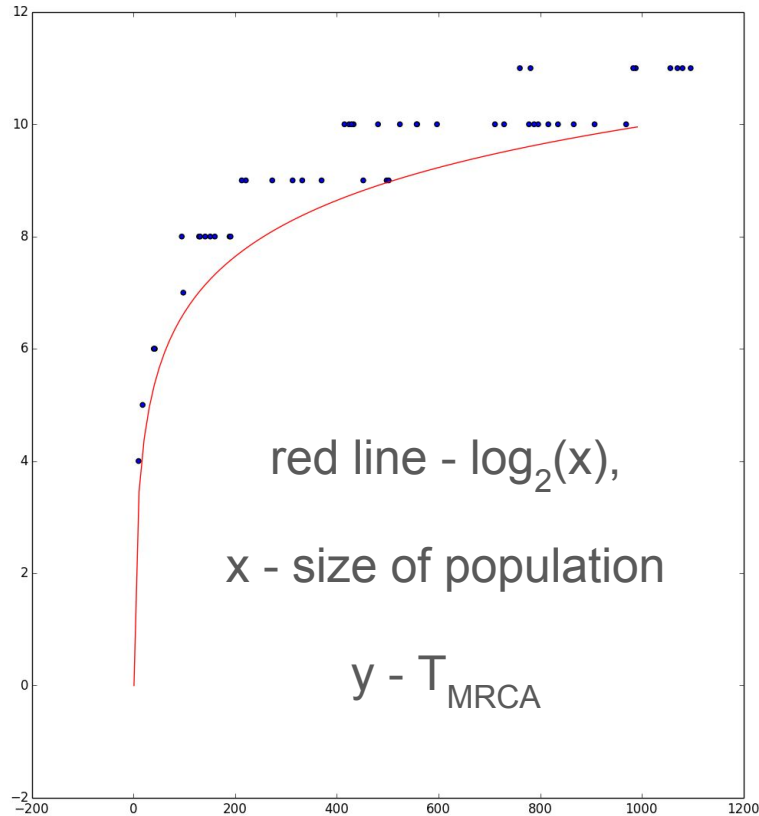
- based on population graph:
 - kth-generation pedigree collapse
 - T_{MRCA}
- based on individual's biology:
 - trait-based
 - Simple Tandem Repeats (STR)
 - Single Nucleotide Polymorphism (SNP)



What we know about T_{MRCA}

Rohde, Olsen and Chang* estimate T_{MRCA} for entire humanity to be ~ 100 generations.

In my simulations T_{MRCA} is no greater than ~ 10 (due to population size)



$T_{MRCA} \sim \log_2(n)^*$,
 n is the size of population,
 valid for randomly mating populations

Formula becomes more complicated if
 the population is not randomly mating*

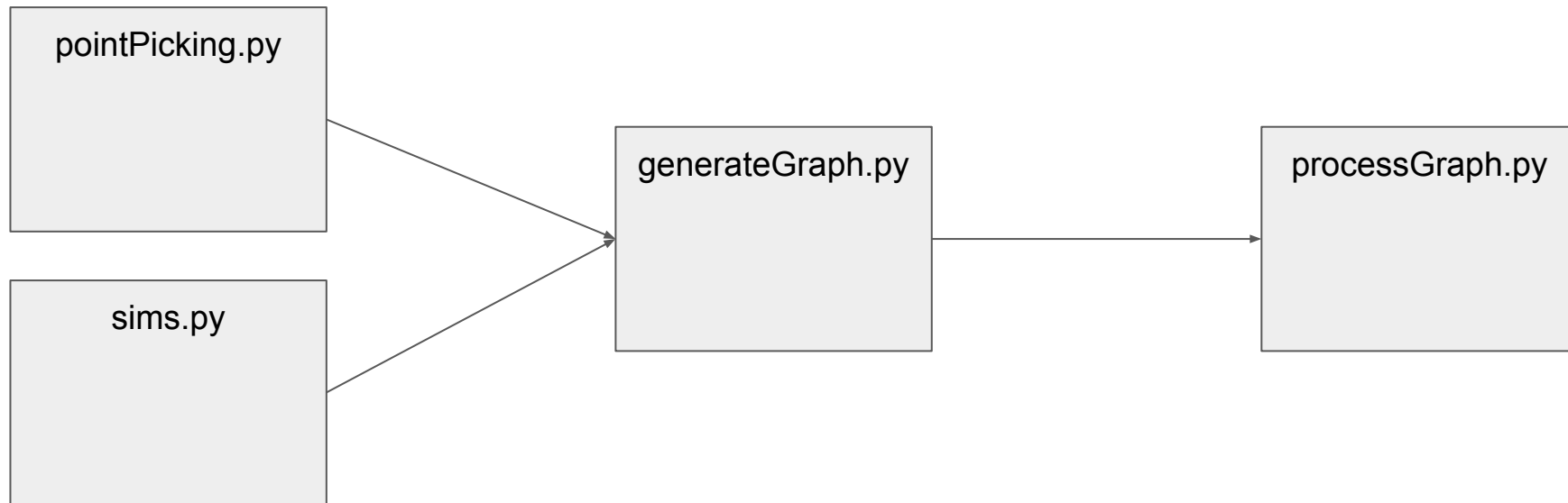
* Rohde, Douglas L. T., Steve Olson, and Joseph T. Chang (Sept. 2004).
 "Modelling the recent common ancestry of all living humans". In: Nature
 431.7008, pp. 562–566. ISSN : 0028-0836. DOI : 10 . 1038 /nature02842

The effects of natural selection were not researched before



But it seemed that they should have big effect. Somebody with a really useful trait should achieve reproductive success and go on to become a MRCA.

The framework



location
genotype
parent-child relation

mating
breeding
inheritance
mutation

T_{MRCA}
feature ratio
statistical analysis

- moderately complex codebase ~ 1600 lines of code

```
[picrin@command naturalSelection]$ find -name "*.py" | grep -v "venv" | xargs wc -l
22 ./presentResults3.py
72 ./naturalSelection/testModule/testSims.py
39 ./naturalSelection/testModule/testPointPicking.py
78 ./naturalSelection/testModule/testProcessGraph.py
143 ./naturalSelection/testModule/testGenerateGraph.py
18 ./naturalSelection/testModule/__init__.py
3 ./naturalSelection/utils.py
106 ./naturalSelection/sims.py
63 ./naturalSelection/pointPicking.py
62 ./naturalSelection/formulaDerivation.py
4 ./naturalSelection/__init__.py
279 ./naturalSelection/generateGraph.py
190 ./naturalSelection/processGraph.py
50 ./runSimulation.py
37 ./fitnessBreederSelectiveSweep.py
24 ./randomBreeding.py
26 ./printGraph.py
1 ./values.py
14 ./test.py
293 ./runExperiments.py
64 ./processSuccessful.py
22 ./presentResults.py
10 ./presentResults2.py
1620 total
[picrin@command naturalSelection]$ scrot
```

- most complex part is the MRCA algorithm (also most extensively tested)
- 25 test cases, ~75% test coverage.

5 sigma testing

```
def testFitnessBreeder(self):
    """
    Tests that the results of fitness breeder are within 5 sigma around mean.
    The feature boosts absolute fitness by the factor of 2.0
    The distribution is assumed to be binomial with the following probabilities:
    A = p(both parents have feature) = 2/3 * 1/2 = 1/3
    B = p(both parents have no feature) = 1/3 * 1/2 = 1/6
    C = p(exactly one parent has the feature) = 2/3 * 1/2 + 1/3 * 1/2 = 1/2

    A + B + C = 1, as expected.

    D = p(at least one parent has no feature) = C + B = 2/3
    E = p(at least one parent has a feature) = C + A = 5/6

    we then use standard deviation result for binomial distribution, say, repeated 100 times.
    The formula is sqrt(n(1-p)p).
    """
    totalSize = 100
    size = int(totalSize/2)

    D = 2.0/3
    E = 5.0/6
    D_sigma = sigmaBinomial(100, D, 5)
    E_sigma = sigmaBinomial(100, E, 5)
    self.assertAlmostEquals(D_sigma, 23.57, delta=0.1)
    self.assertAlmostEquals(E_sigma, 18.6, delta=0.1)
    unsuccessfulSims = [createRandomlyPositionedSim() for i in range(size)]
    successfulSims = [createRandomlyPositionedSim() for i in range(size)]
    successfulSimsIDs = [sim["uid"] for sim in successfulSims]
    unsuccessfulSimsIDs = [sim["uid"] for sim in unsuccessfulSims]
    for sim in successfulSims:
        makeHomozygous(sim)
    for sim in unsuccessfulSims:
        cancelFeature(sim)
    sims = successfulSims + unsuccessfulSims
    for sim in sims:
        makeDominant(sim)
    nextGeneration = fitnessBreeder(sims, 2.0, len(sims))
    successfulParents = 0
    unsuccessfulParents = 0
    self.assertTrue(len(nextGeneration) == totalSize)
    for sim in nextGeneration:
        if sim["parentA"]["uid"] in successfulSimsIDs or sim["parentB"]["uid"] in successfulSimsIDs:
            successfulParents += 1
        if sim["parentA"]["uid"] in unsuccessfulSimsIDs or sim["parentB"]["uid"] in unsuccessfulSimsIDs:
            unsuccessfulParents += 1
    self.assertTrue(-E_sigma <= successfulParents - E*totalSize <= E_sigma)
    self.assertTrue(-D_sigma <= unsuccessfulParents - D*totalSize <= D_sigma)
```

```
4 5 6 7 8 9 picir@command:~/programming/naturalSelection
testing with python2
random state dumped to .randomstate
testAdjustFitness (naturalSelection.testModule.testSims.TestSims) ... ok
testHasFeature (naturalSelection.testModule.testSims.TestSims) ... ok
testIterativeHomebody (naturalSelection.testModule.testSims.TestSims)
Impressive, I can get 10^-14 accuracy of the point staying ... ok
testRandomGeneCopy (naturalSelection.testModule.testSims.TestSims) ... ok
testSimsBreed (naturalSelection.testModule.testSims.TestSims) ... ok
testBuckets (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testDistance (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testMove (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testBreeder (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testCondWriter (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testFitnessBreeder (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testFitnessCDF (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testMater (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testMater2 (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testParentChildFirstPart (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testAlleleProportion (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testFeatureProportion (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testRandomMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testRepeatedMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testSatQuickMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testTracers (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testUnsatQuickMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testValidGenerationNumbers (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testValidJson (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
.....
Ran 25 tests in 0.202s

OK
testing with python3
random state dumped to .randomstate
testAdjustFitness (naturalSelection.testModule.testSims.TestSims) ... ok
testHasFeature (naturalSelection.testModule.testSims.TestSims) ... ok
testIterativeHomebody (naturalSelection.testModule.testSims.TestSims)
Impressive, I can get 10^-14 accuracy of the point staying ... ok
testRandomGeneCopy (naturalSelection.testModule.testSims.TestSims) ... ok
testSimsBreed (naturalSelection.testModule.testSims.TestSims) ... ok
testBuckets (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testDistance (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testMove (naturalSelection.testModule.testPointPicking.TestPointPicking) ... ok
testBreeder (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testCondWriter (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testFitnessBreeder (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testFitnessCDF (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testMater (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testMater2 (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testParentChildFirstPart (naturalSelection.testModule.testGenerateGraph.TestGenerateGraph) ... ok
testAlleleProportion (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testFeatureProportion (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testRandomMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testRepeatedMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testSatQuickMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testTracers (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testUnsatQuickMRCA (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testValidGenerationNumbers (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
testValidJson (naturalSelection.testModule.testProcessGraph.TestProcessGraph) ... ok
.....
Ran 25 tests in 0.294s

OK
[picir@command naturalSelection]$ scrot -d 2
```

Parameter choice

- population kept constant throughout the simulation (e.g. $n = 100$)
- discrete generations
- probability of becoming the first parent proportional to relative advantage, computed from absolute advantage (e.g. 1.05):

$$p(s, S) = \frac{f(s)}{\sum_{s' \in S} f(s')}$$

where S is the set of all sims in the previous generation (including s), and $f(s)$, the sim's absolute fitness, is defined by

$$f(s) = \begin{cases} 1.05, & \text{if sim } s \text{ has the trait} \\ 1, & \text{if sim } s \text{ does not have the trait} \end{cases}$$

- case - control study
- closely follows the approach of Peng and Amos *

* Peng, Bo and Christopher I. Amos (2010). "Forward-time simulation of realistic samples for genome-wide association studies". In: BMC Bioinformatics 11, p. 442. ISSN : 1471-2105

Simulating a concrete population easy

```
1 2 3 4 5 6 7 8 9 picrin@command:~/programming/naturalSelection
from naturalSelection import *
import time
import sys
initialSize = 100
generations = 700
count = 0

def SSDWFBS(size, generations):
    """
    stands for Selective Sweep Dominant Wanderer Fitness Breeder Simulation.
    """
    def firstGeneration():
        return simsFrame(populationSize = size, mutator = firstMutator)
    def nextGeneration(nextFrame):
        return nextFrame(migrator=wandererMigrator, breeder=AdvantageFitnessBreeder, mutator = dominantSingleAlleleMutator)
    return generatePopulationPure(generations, firstGeneration, nextGeneration)

def AdvantageFitnessBreeder(sims):
    return fitnessBreeder(sims, 1.05, len(sims))

def firstMutator(allSims):
    for sim in allSims:
        sim["genotype"]["isDominant"] = True
        dominantSingleAlleleMutator(allSims)
        sim["genotype"]["hasCopy1"] = True

def nullMutator(allSims):
    pass

filename = "fitnessBreederResults/checkMRCASmall"
with open(filename, "w") as result:
    writePopulation(result, SSDWFBS(initialSize, generations))

with open(filename, "r") as result:
    simulation = loadGraph(result)
    print(quickMRCAs(simulation, -1, 2**64))
```

Unfortunately the results are inconclusive

T_{MRCA}		
	sample	control
average	6.415	6.37
median	6	6
mode	7	7
min	5	5
max	7	7

Table 3.1: statistics for Selective Sweep in Randomly Mating Population experiment

	Mann-Whitney	Kolomogorov-Smirnov
p-values	0.258987	0.999987

Table 3.2: p-values of null-hypothesis for Selective Sweep in Randomly Mating Population experiment

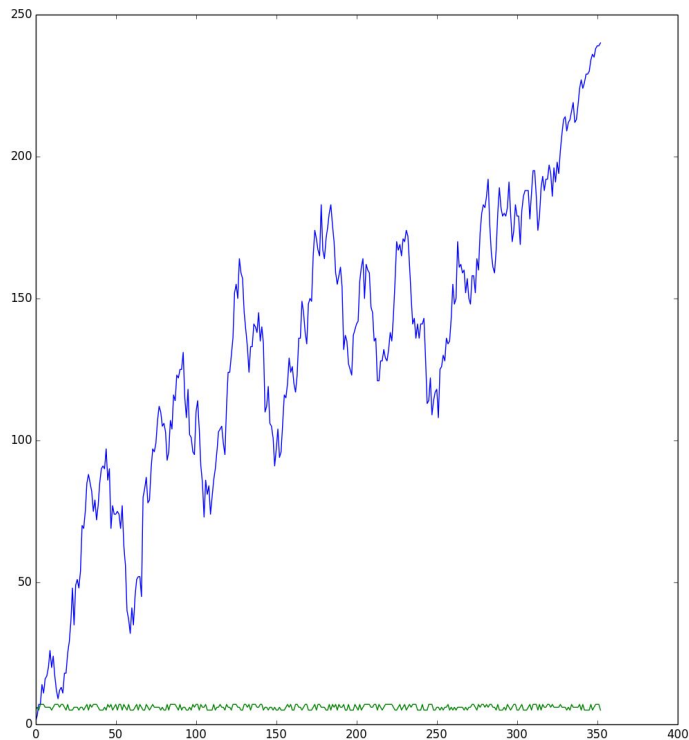
Why?

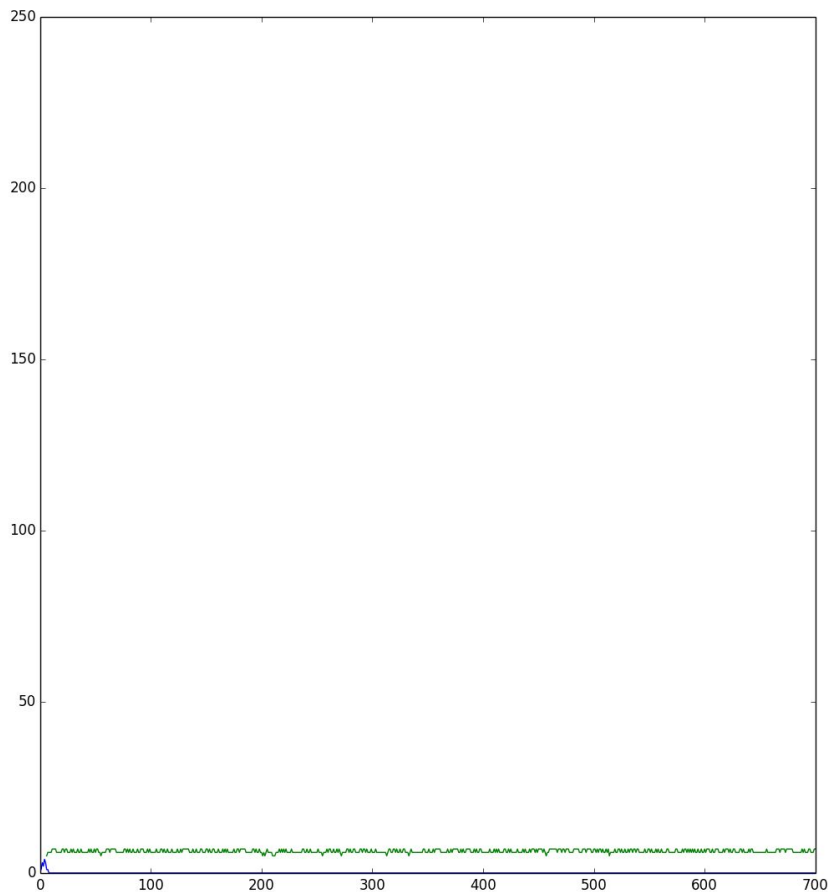
Natural selection is sloooooow.

x-axis: generation

green - T_{MRCA}

blue - allele proportion.





And it rarely works. Only about 4% of simulations end in advantageous feature spreading to the entire population.

x-axis: generation

green - T_{MRCA}

blue - allele proportion.

Future work

- Run the experiments for populations with more structure -> bigger T_{MRCA}
- Is natural selection enough? Consider sexual selection.