- How to get started:
    - you were able link to your to a local dir on your machine, i.e. store your data on the local machine
        - your container is still available (docker ps -a):

```
docker start -i <containerID>
```

        - your container got removed (—rm option when started or `docker rm <containerID>`):

```
docker run -it -p 8888:8888 -v ~:/home/jovyan/work qtlrocks/jwas-docker
```

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

Welcome to XSim.

XSim is a fast and user-friendly software tool to simulate sequence data and complicated pedigree structures

Quick-start Julia

```
> using XSim
> XSim.init(numChr,numLoci,chrLength,geneFreq,mapPos,qtlMarker,qtlEffects,mutRate)
> popSizeFounder = 2
> sires = sampleFounders(popSizeFounder)
> sires1,dams1,gen1 = sampleRan(popSize, ngen, sires, dams);
```

Quick-start C++

Library, demo1, demo2, demo3, data

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- simulates genome storing only base genome and recombination events (—> efficient)
  - dropping down allele origin not allele states
  - ignoring mutation, a chromosome in an individual is unambiguously defined by 2 vectors:
    - vector of allele origins
    - vector of crossover positions
- additional vector necessary to store mutation sites

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

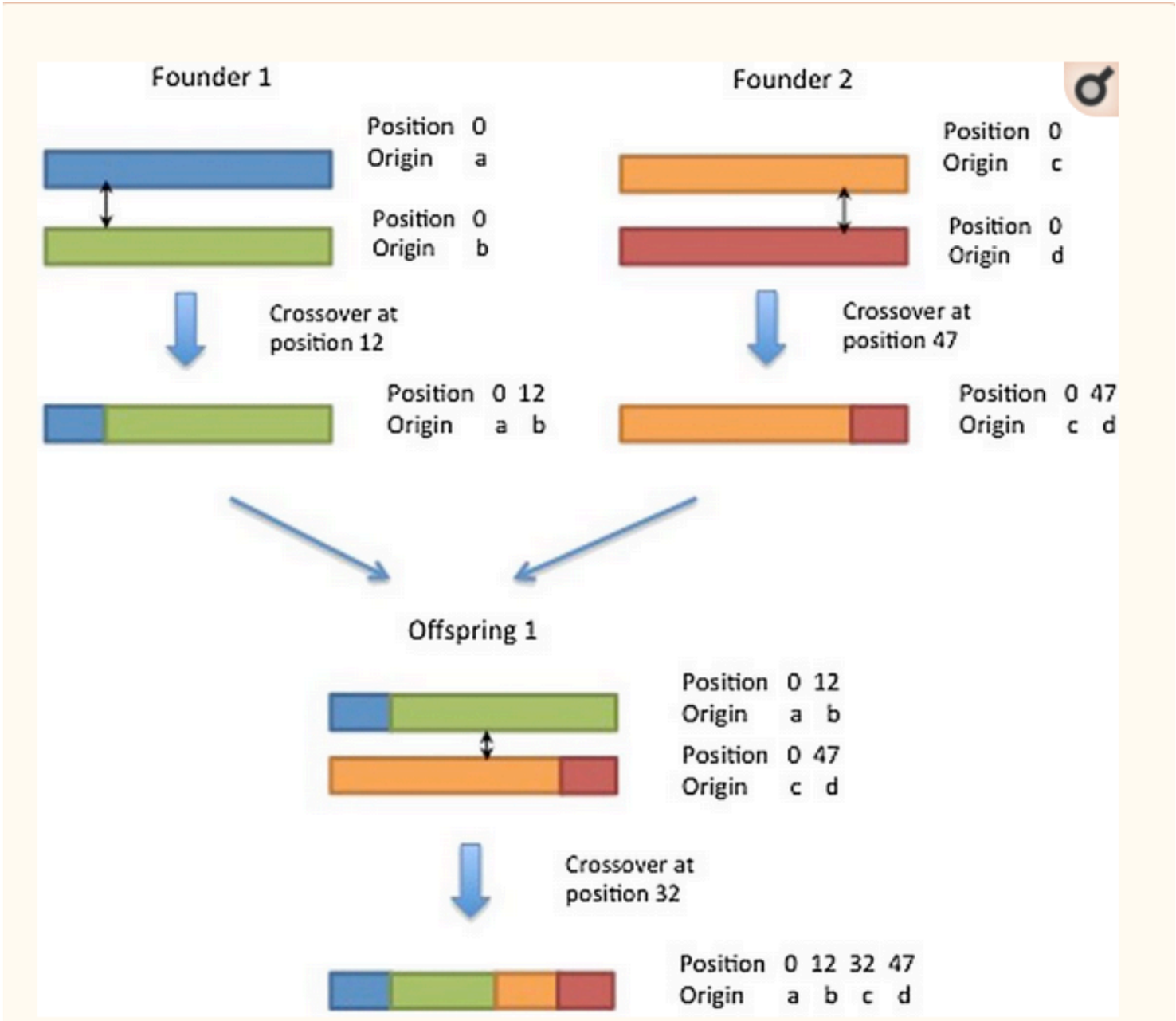


Figure 1

An example to illustrate the simulation strategy (crossover sites indicated by ↕).

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4502375/

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- using real genotypes for base population or simulating based on allele frequencies
- arbitrary pedigree structures and mating designs
- based on the concept of ‘mating cohorts’
- user-defined parameters for allele frequency, map position, number of loci chromosome length & number, (map function), mutation rate.

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

1dArrays
2dArrays

- first building block

build_genome(*nChromosome, chromosomelength, nLoci, gene_frequency, map_position, qtl_index=[], qtl_effect=[], mutation_rate, genotypeErrorRate=0, myCommon=common*)

core method

build_genome(*nChromosome, chromosomelength, nLoci_each_chrom, qtl_each_chrom, mutation_rate*)

build_genome(*nChromosome, chromosome_length, nLoci, gene_frequency, map_position, mutation_rate, qtl_index=[], qtl_effect=[], genotypeErrorRate=0, myCommon=common*)

- creates “objects” myLocusInfo, myChromosomeInfo and myGenomeInfo

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

1dArrays
2dArrays

- first building block

build_genome(*nChromosome, chromosomelength, nLoci, gene_frequency, map_position, qtl_index=[], qtl_effect=[], mutation_rate, genotypeErrorRate=0, myCommon=common*)

core method

- XSim.G is created, an object of type ‘GenomeInfo’
  - GenomeInfo has
    - chr::Array{ChromosomeInfo,1}
    - numChrom::Int64
    - mutRate::Float64
    - genotypeErrorRate::Float64
    - qtl_index::Array{Int64,1}
    - qtl_effects::Array{Float64,1}

- ‘ChromosomeInfo’ is an object which has
  - chrLength::Float64
  - numLoci::Int64
  - mapPos::Array{Float64,1}
  - loci::Array{LocusInfo,1}

- ‘LocusInfo’ is an object which has
  - map_pos::Float64
  - allele_freq::Array
  - QTL::Bool
  - QTL_effect::Float64

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

build_genome(*nChromosome, chromosomelength, nLoci_each_chrom, qtl_each_chrom, mutation_rate*)

- each chromosome has length '*chromosomelength' with 'nLoci_each_chrom'* lequally space loci located on each oft them
- gene frequency is 0.5 for all loci
- '*qtl_each_chrom'* QTL are sampled randomly per chromosome
- QTL-effects are randomly sampled from a normal(0,1) distribution
- then the core function is called

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

1dArrays
2dArrays

build_genome(*chromosome_length, nLoci, gene_frequency,*
map_position, mutation_rate, qtl_index=[], qtl_effect=[],
genotypeErrorRate=0, myCommon=common)

- single chromosome has length '*chromosomelength' with 'nLoci'* equally spaced loci located on each oft them
- gene frequency is NOT 0.5 for all loci
- Array map_position specifies the location of each locus, positions are the same across chromosomes
- arrays of QTL and corresponding effects are specified, again the same for all chromosomes
- then the core function is called

let’s try

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- second building block: Animals in cohorts

```
mutable struct Cohort

    animalCohort::Array{Animal,1}

    npMatrix::Array{Int64,2}

end
```

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- second building block: Animals in cohorts

```
mutable struct Animal
    genomePat::Array{Chromosome,1}
    genomeMat::Array{Chromosome,1}
    breedComp::Array{Float64,1}
    myID::Int64
    sireID::Int64
    damID::Int64
    phenVal::Float64
    genVal::Float64
    ebv::Float64
end
```

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- second building block: Animals in cohorts

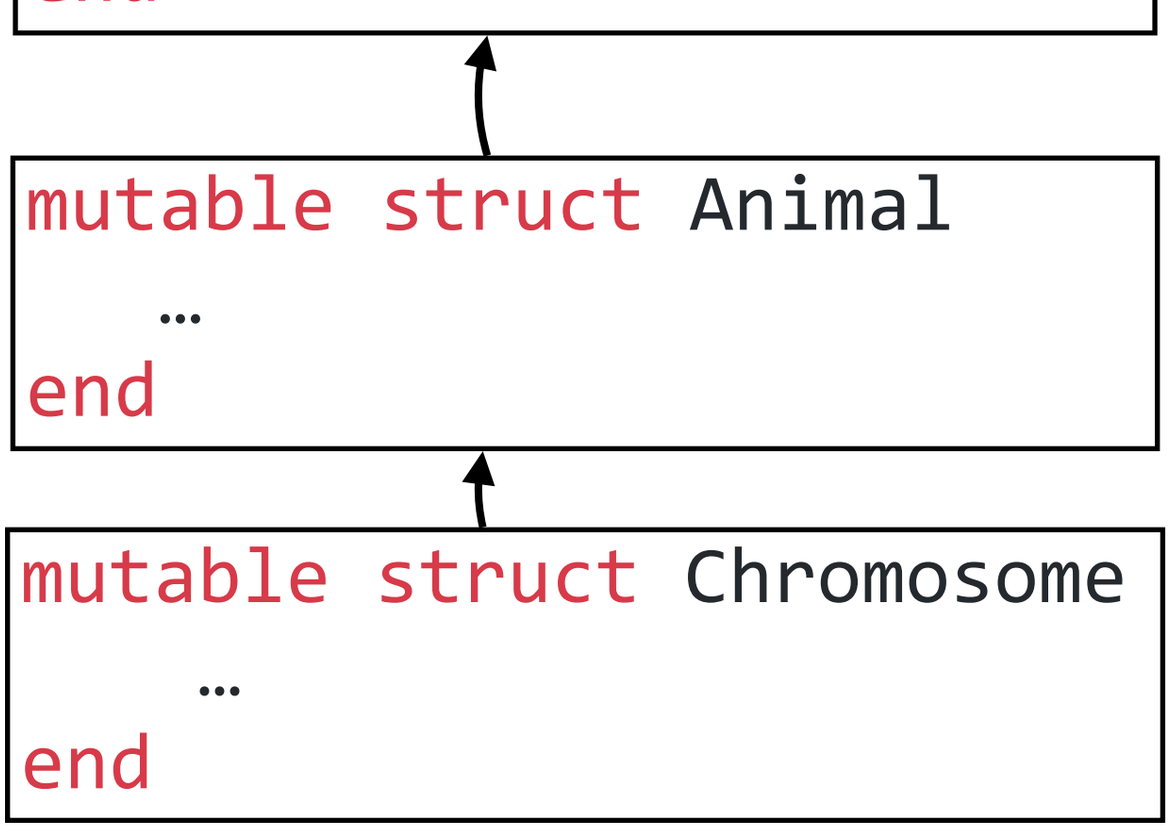
```
mutable struct Chromosome

    haplotype::Array{Int64,1}

    ori::Array{Int64,1}

    pos::Array{Float64,1}

end
```

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- second building block: Animals in cohorts

```
mutable struct Cohort
    …
end
```



```
mutable struct Animal
    …
end
```



```
mutable struct Chromosome
    …
end
```

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- sampling founders from phased genotype data

  sampleFounders*(popSizeFounder, "fileName")*

  - *creates a cohort of Founders (array of individuals)*
  - *labels each individuals maternal and paternal chromosome with 1… 2*numInd*nchrom*
  - *reads each individuals two chromosomes denoted as "m" and "p" into maternal and paternal chromosome of each individual*
  - *returns a cohort of base individuals*
  - *can split that cohort into male and female Individuals to start mating*

Introduction to XSim

(Hao Cheng, Rohan Fernando, Dorian Garrick)

- sampling children from individuals (with phased genotype data)

sampleRan*(popSize, nGen, sires, dams;gen=1,fileName=“",printFlag=true)*

sampleSel*(popSize, nSires, nDams, nGen, varRes=common.varRes)*

sampleSel*(popSize, nSires, nDams, nGen, maleParents, femaleParents, varRes=common.varRes; gen=1,*
fileName="", direction=1)

sampleBLUPSel*(popSize, nSires, nDams, nGen, maleParents, femaleParents, varRes=common.varRes, varGen=1;*
gen=1, fileName="XSim", direction=1)

- *creates a cohort of Founders (array of individuals)*
- *labels each individuals maternal and paternal chromosome with 1…2*numInd*nchrom*
- *reads the two chromosomes denoted as “m” and “p” into maternal and paternal chromosome of each individual*
- *returns a cohort of base individuals*
- *can split that cohort into male and female Individuals to start mating*

let’s try