*Bioinformatics*, 2023, **39(12)**, btad691
https://doi.org/10.1093/bioinformatics/btad691
Advance Access Publication Date: 22 November 2023
**Applications Note**





Genetics and population analysis

# ChromaX: a fast and scalable breeding program simulator

**Omar G. Younis [1,*], Matteo Turchetta[1,*], Daniel Ariza Suarez[2], Steven Yates[2], Bruno Studer[2], Ioannis N. Athanasiadis[3], Andreas Krause[1], Joachim M. Buhmann[1], Luca Corinzia [1,*]**

[1]Department of Computer Science, ETH Zurich, Zürich, 8092, Switzerland
[2]Department of Environmental Systems Science, Molecular Plant Breeding, ETH Zurich, Switzerland
[3]Department of Social Sciences, Wageningen University & Research, Wageningen, 6708, The Netherlands

*Corresponding authors. Department of Computer Science, ETH Zurich, OAT Y, Andreasstrasse 5, Zürich, 8092, Switzerland. E-mails: omargallalaly.younis@inf.ethz.ch (O.G.Y.), luca.corinzia@inf.ethz.ch (L.C.), and matteo.turchetta@inf.ethz.ch (M.T.)

Associate Editor: Russell Schwartz

## Abstract

**Summary:** ChromaX is a Python library that enables the simulation of genetic recombination, genomic estimated breeding value calculations, and selection processes. By utilizing GPU processing, it can perform these simulations up to two orders of magnitude faster than existing tools with standard hardware. This offers breeders and scientists new opportunities to simulate genetic gain and optimize breeding schemes.

**Availability and implementation:** The documentation is available at https://chromax.readthedocs.io. The code is available at https://github.com/kora-labs/chromax.

## 1 Introduction

Livestock and plant breeding is crucial to sustainable agriculture (Schön and Simianer 2015) and to develop new breeds more suited for specific environments or market demands (Qaim 2020). Recently, the availability of genomic data and advanced statistical methods have revolutionized breeding programs (Kim *et al.* 2020). Notably, genomic selection allows a breeder to predict the performance of an individual based on genetic makeup, avoiding expensive phenotyping (Meuwissen *et al.* 2001, Crossa *et al.* 2017). These new methods unlock various design possibilities for breeding schemes, making it harder to optimize them. Moreover, a single breeding cycle can take many years, involving many design choices during the process. Thus, there is a growing interest in using simulations to optimize breeding programs. The existing tools for simulating crosses are implemented in R (Broman *et al.* 2003, Mohammadi *et al.* 2015, Gaynor *et al.* 2020, Pook *et al.* 2020) or Julia (Chen *et al.* 2022). While they provide an extensive set of features, they are not capable of exploiting parallelism in high-performance computers that may be necessary for large and complex breeding schemes. For example, simulating a full-diallel cross of ten individuals with ten offspring results in 450 offspring, while a similar diallel of 20 individuals generates 1900 offspring. With this rapid scaling, simulating a full diallel in a breeding program with thousands of individuals may be unfeasible; hence developing tools that can speed up simulations is desirable. The most attractive language for this purpose is Python. Python is one of the most used programming languages for numerical computing and data science, with many libraries available for optimization and machine learning (Pedregosa *et al.* 2011, Bradbury *et al.* 2018, Paszke *et al.* 2019, Harris *et al.* 2020, Virtanen *et al.* 2020). Some of them allow exploiting parallelization capabilities of specialized hardware devices, such as Graphics Processing units (GPUs) and Tensor Processing Units (TPUs). Thus, developing a parallelizable Python tool for simulating crosses can increase the throughput of simulations and opens new opportunities to improve breeding efficiency. In this paper, we introduce ChromaX, a fast and scalable Python library that enables the stochastic simulation of the most common features in a breeding program like genetic recombination, fixation of genomes by doubled haploid induction, and selection. It can further calculate the Genomic Estimated Breeding Value (GEBV), and simulate genotype-by-environment interactions. ChromaX has been designed and implemented with scalability in mind and exploits recent advances in that direction from the Python language.

## 2 Software description

ChromaX is based on the high-performance numerical computing library JAX (Bradbury *et al.* 2018). Using JAX, ChromaX functions are compiled in XLA (Accelerated Linear Algebra) (Sabne 2020), a compiler for linear algebra that accelerates function execution according to the domain and hardware available. This allows ChromaX to run seamlessly on various devices, such as CPUs, GPUs, and TPUs exploiting the parallelization offered by a variety of high-performance computing devices. ChromaX is available with the open-source license 3-Clause BSD. Source code is available on GitHub and is also distributed via the Python package installer "pip" as *chromax*. In the following sections, we

**Received:** 16 June 2023; **Revised:** 8 October 2023; **Editorial Decision:** 7 November 2023

describe the core functions available with ChromaX; for a complete list see the documentation.

### 2.1 Simulator initialization

The simulator is initialized by providing a genetic linkage map supplied as a Pandas DataFrame (McKinney *et al.* 2010) or a path to a spreadsheet. In the genetic linkage map, each row represents a marker and columns contain the chromosome identifier, the position of the marker in centimorgans, and a column for each trait containing linear marker effects. Instead of marker positions, it can include a column indicating the probability of recombination occurring after the marker and before the next. The argument *trait_names* can be used to specify a list of trait names; every element must match a column name in the genetic linkage map. A heritability value for each trait can also be specified.

### 2.2 Population data

ChromaX represents the genome of an individual as a NumPy array of shape $(m, d)$, where $m$ is the total number of markers and $d$ is the ploidy. Thus, a population of $n$ individuals is represented as an array of shape $(n, m, d)$. Please note that this encoding requires that input genetic data is phased, which can be achieved using a program such as Beagle (Browning *et al.* 2021). The simulator allows the user to load population data from a file and save it at any point in the breeding program simulation.

### 2.3 Genetic recombination

ChromaX simulates the genetic recombinations that take place during meiosis to create new haplotypes. For simplicity, we assume diploid species and the Poisson model for crossover interference (McPeek and Speed 1995). The genetic recombination function receives as input an array of genetic markers of the $k$ parent pairs. The function performs crosses between pairs of parents and produces $k$ offspring. The use of a single genetic recombination function for all biparental crosses allows the function to be parallelized across several dimensions, namely the number of crosses $k$, the pair of parents, and the ploidy number. Generalizations to autopolyploid species and to other models for crossover interference are left for future developments.

### 2.4 Differentiable genetic recombination

In ChromaX, we further develop a novel genetic recombination function that generalizes the one that takes place in biparental crosses. This function computes the parents of a cross by taking the weighted average of a population. With a weight vector of dimension $(k, n, 2)$, it performs $k$ crosses on a population of size $n$. Using the JAX *grad* functionality, the user can obtain the analytical gradient with respect to parents' weights. This provides a continuous relaxation of the genetic recombination that can be used to optimize the crossing by gradient-based methods (Polak 2012).

### 2.5 Doubled haploid lines

ChromaX can simulate the fixation of genomes by doubled haploid induction. The user can specify the number of offspring per individual $dh$; ChromaX generates a line from each individual of the population. With an input population of shape $(n, m, d)$, the generated population will be of shape $(n, dh, m, d)$. Like the genetic recombination function, the parallelization occurs over the number of lines $n$, ploidy $d$, and the number of individuals per line $dh$.

### 2.6 Traits

ChromaX computes the GEBV for additive traits of a population using the marker effects available in the genetic linkage map or drawn from a standard probability distribution (e.g. normal distribution). The marker effects are represented as an array of shape $(t, m)$, where $t$ is the number of traits and $m$ is the number of markers. ChromaX computes the genomic value by performing a tensor contraction of the marker effect with the input population array of markers of shape $(n, m, d)$. The arrays are multiplied along the $m$-axis and then sum reduced along the $m$ and $d$ axes. The result is an array of shape $(n, t)$ containing the GEBV of the population for each trait. This operation is well-suited for GPUs due to their ability to efficiently perform multi-dimensional multiplication and sum reduction.

ChromaX can further model phenotypes that have a genotype-by-environment interaction component, as described in Faux *et al.* (2016). In genotype-by-environment interaction, an environment is simulated as a random variable drawn from a normal distribution; this value multiplies a random additive trait that we fix at the beginning of the simulation with the variance determined by the heritability.

### 2.7 Selection

ChromaX enables the user to select the best individuals from a population based on a user-defined score. The score function accepts population data as input and returns an array of $n$ scores, where $n$ is the number of individuals in the population. Examples of scoring functions included in the software are breeding values (Lande and Thompson 1990), optimal haploid value (Daetwyler *et al.* 2015), and phenotype.

## 3 Performance

The performance of ChromaX is compared to AlphaSimR (Gaynor *et al.* 2020), a popular breeding program simulator. We perform this comparison on two hardware settings representing typical simulation conditions: a computing cluster CPU (*AMD EPYC 7742 64-Core*), and a commonly available GPU (*Quadro RTX 6000*). Table 1 shows the computation times when simulating 1k, 10k, and 100k biparental crosses from a population described by 1000 markers across ten chromosomes. We do not benchmark AlphaSimR on GPU as it cannot run on this hardware. On the CPU, the computation time scales linearly with the number of crosses for both simulators and ChromaX is around three times faster than AlphaSimR for the tested sizes. In contrast, on GPU the

**Table 1.** User CPU time in milliseconds as a function of the numbers of crosses for AlphaSimR and ChromaX on different hardware settings (CPU, CPU computer cluster AMD EPYC 7742 64-Core; GPU, Quadro RTX 6000).[a]

| | Number of crosses | | |
|---|---|---|---|
| | 1k | 10k | 100k |
| AlphaSimR CPU | 150 ± 5 | 1352 ± 9 | 14 821 ± 149 |
| ChromaX CPU | 47 ± 1 | 473 ± 9 | 5149 ± 94 |
| ChromaX GPU | 3 ± 0 | 6 ± 1 | 29 ± 2 |

[a] Reported are the mean ± standard deviation over 100 simulations.

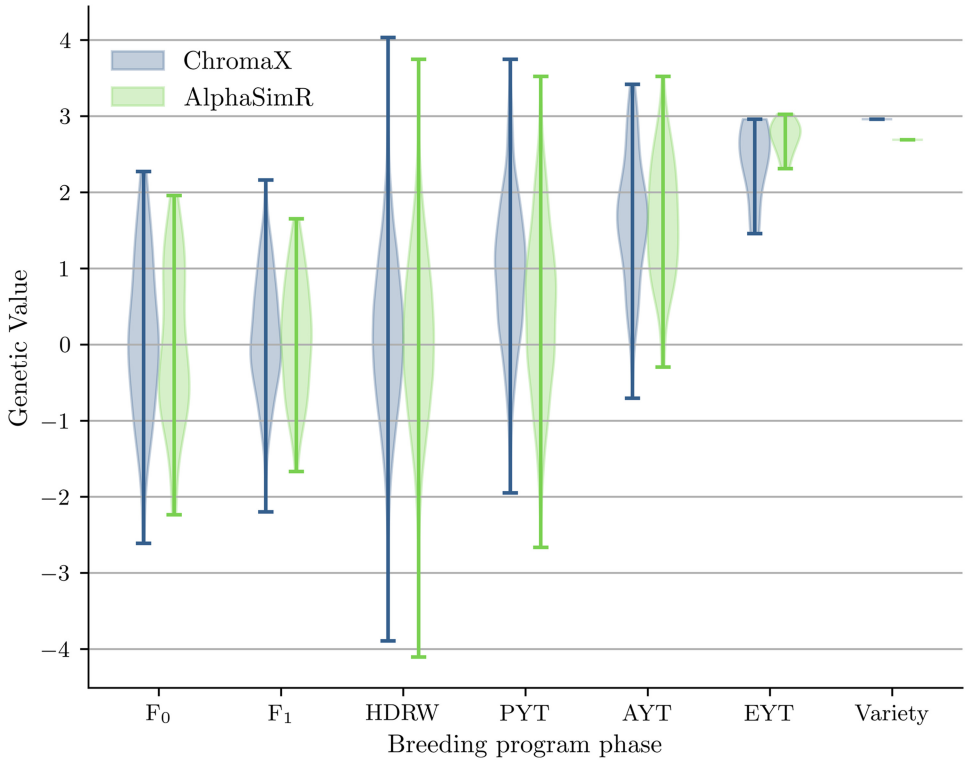


**Figure 1.** Comparision between the genetic value simulated by ChromaX and AlphaSimR for the same breeding schema. The $F_0$ population contains 50 lines and $F_1$ is created by 200 random biparental crosses. From each line in $F_1$, 100 doubled haploids are obtained and evaluated in head rows (HDRW) using visual selection (low accuracy). Then plants are evaluated with increasing accuracy while reducing the population size. PYT, Preliminary Yield Trial; AYT, Advanced Yield Trial; EYT, Elite Yield Trial

computation time scales sub-linearly (10× the computation time for 100× more crosses), and ChromaX can be hundreds of times faster than AlphaSimR.

As a further comparison, we implement the breeding program described in Gaynor *et al.* (2020) with both simulators and compare the results. This breeding program, typical for an inbred species, assumes an initial diallel $F_0$ characterized by 100 QTL per chromosome and 21 chromosomes. $F_1$ is obtained by performing random biparental crosses from $F_0$. Then, we obtain homozygous lines using the doubled haploid technique. To obtain the final cultivars, we simulate visual selection on head rows and preliminary, advanced, and elite yield trials, where individuals are evaluated with increasing accuracy for smaller population sizes. Figure 1 shows the evolution of the population breeding values during the program. While some differences are expected due to the stochasticity of the process, the values are similar for both simulators. Table 2 reports the simulation times when the population sizes at the different generations are multiplied by various scaling factors. The results are similar to Table 1 with ChromaX achieving around 500× speed up on GPU compared to AlphaSimR. Finally, in Table 3, we present data regarding the peak memory usage on the device, measured in megabytes. Notably, ChromaX exhibits higher memory consumption, attributed to its functional design and the utilization of vectorization techniques. We aim to tackle this issue in forthcoming releases.

## 4 Discussion

Our benchmarking experiments show that ChromaX can utilize modern hardware and parallelism to run simulations many times faster than AlphaSimR, in some cases by orders of magnitude. Crucially, this will pave the way for the systematic exploration and optimization of complex designs in modern breeding programs.

**Table 2.** User CPU time in seconds for simulating the inbred schema from Gaynor *et al.* (2020) as a function of the population size on different hardware.[a]

| | Scaling factor | | |
|---|---|---|---|
| | 1× | 5× | 20× |
| AlphaSimR CPU | 3.37 ± 0.18 | 26.59 ± 0.34 | 95.71 ± 0.44 |
| ChromaX CPU | 1.28 ± 0.02 | 6.36 ± 0.06 | 24.08 ± 0.31 |
| ChromaX GPU | 0.02 ± 0.00 | 0.05 ± 0.00 | 0.15 ± 0.00 |

[a] Mean values ± standard deviation over 100 simulations are reported.

## 5 Limitations

ChromaX can simulate standard breeding programs but has limitations. First, ChromaX requires a genetic linkage map with marker effects and genetic data of the populations to simulate breeding cycles. To circumvent these requirements, other programs can simulate these data *in silico* by making some assumptions about the genetic features of the species. Second, ChromaX models additive traits and genotype-by-environment effects to simulate population phenotypes. ChromaX does not yet implement other biological effects, such as dominance or epistasis. Finally, ChromaX is designed for breeding programs of self- or open-pollinated species; other systems, such as hybrid breeding from distant heterotic

**Table 3.** Device peak memory usage in gigabytes for simulating the inbred schema from Gaynor *et al.* (2020) as a function of the population size on different hardware.[a]

| | Scaling factor | | |
|---|---|---|---|
| | 1× | 5× | 20× |
| AlphaSimR CPU | 0.14 | 0.40 | 2.14 |
| ChromaX CPU | 0.21 | 0.64 | 14.97 |
| ChromaX GPU | 1.38 | 8.79 | 24.71 |

[a] We report the mean across 10 runs. We do not report the standard deviation as it is below the reported resolution for every entry of the table.

groups, or monoecious/dioecious reproductive systems are not supported. Addressing these limitations will enable widespread use in a wider community.

## Acknowledgements

The authors thank the anonymous reviewers for their valuable suggestions.

## Conflict of interest

None declared.

## Funding

This work was supported by the Wageningen University and Research research theme "Data Driven Discoveries in a Changing Climate", reference 531/AQ5541.

## Data availability

No new data were generated or analysed in support of this research.

## References

Bradbury J, Frostig R, Hawkins P *et al. JAX: Composable Transformations of Python+NumPy Programs*. 2018.

Broman KW, Wu H, Sen Ś *et al.* R/qtl: QTL mapping in experimental crosses. *Bioinformatics* 2003;**19**:889–90.

Browning BL, Tian X, Zhou Y *et al.* Fast two-stage phasing of large-scale sequence data. *Am J Hum Genet* 2021;**108**:1880–90.

Chen CJ, Garrick D, Fernando R *et al.* XSim version 2: simulation of modern breeding programs. *G3 Genes Genomes Genet* 2022;**12**: jkac032.

Crossa J, Pérez-Rodríguez P, Cuevas J *et al.* Genomic selection in plant breeding: methods, models, and perspectives. *Trends Plant Sci* 2017; **22**:961–75.

Daetwyler HD, Hayden MJ, Spangenberg GC *et al.* Selection on optimal haploid value increases genetic gain and preserves more genetic diversity relative to genomic selection. *Genetics* 2015;**200**:1341–8.

Faux A-M, Gorjanc G, Gaynor RC *et al.* Alphasim: software for breeding program simulation. *Plant Genome* 2016;**9**:0013.

Gaynor RC, Gorjanc G, Hickey JM. AlphaSimR: an R package for breeding program simulations. *G3 Genes Genomes Genet* 2020;**11**: jkaa017.

Harris CR, Millman KJ, van der Walt SJ *et al.* Array programming with NumPy. *Nature* 2020;**585**:357–62.

Kim KD, Kang Y, Kim C. Application of genomic big data in plant breeding: past, present, and future. *Plants* 2020;**9**:1454.

Lande R, Thompson R. Efficiency of marker-assisted selection in the improvement of quantitative traits. *Genetics* 1990;**124**:743–56.

McKinney W *et al.* 2010. Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–6.

McPeek MS, Speed TP. Modeling interference in genetic recombination. *Genetics* 1995;**139**:1031–44.

Meuwissen TH, Hayes BJ, Goddard ME. Prediction of total genetic value using genome-wide dense marker maps. *Genetics* 2001;**157**: 1819–29.

Mohammadi M, Tiede T, Smith KP. Popvar: a genome-wide procedure for predicting genetic variance and correlated response in biparental breeding populations. *Crop Sci* 2015;**55**:2068–77.

Paszke A, Gross S, Massa F *et al.* Pytorch: an imperative style, high-performance deep learning library. *Adv Neural Inf Process Syst* 2019;**32**:8024–35.

Pedregosa F, Varoquaux G, Gramfort A *et al.* Scikit-learn: machine learning in python. *J Mach Learn Res* 2011;**12**:2825–30.

Polak E. *Optimization: Algorithms and Consistent Approximations*, Vol. **124**. New York: Springer Science & Business Media, 2012.

Pook T, Schlather M, Simianer H. MoBPS – modular breeding program simulator. *G3 (Bethesda)* 2020;**10**:1915–8.

Qaim M. Role of new plant breeding technologies for food security and sustainable agricultural development. *Applied Eco Perspectives Pol* 2020;**42**:129–50.

Sabne A. *XLA: Compiling Machine Learning for Peak Performance*. 2020.

Schön CC, Simianer H. Resemblance between two relatives – animal and plant breeding. *J Anim Breed Genet* 2015;**132**:1–2.

Virtanen P, Gommers R, Oliphant TE *et al.*; SciPy 1.0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat Methods* 2020;**17**:352.