PyBrOpS: a Python package for breeding program simulation and optimization for multi-objective breeding

Robert Z. Shrote and Addie M. Thompson

Abstract

Plant breeding is a complex endeavor that is almost always multi-objective in nature. In recent years, stochastic breeding simulations have been used by breeders to assess the merits of alternative breeding strategies and assist in decision making. In addition to simulations, visualization of a Pareto frontier among multiple competing breeding objectives can assist breeders in decision making. This paper introduces Python Breeding Optimizer and Simulator (PyBrOpS), a Python package capable of performing multi-objective optimization of breeding objectives and stochastic simulations of breeding pipelines. PyBrOpS is unique among other simulation platforms in that it can perform multi-objective optimizations and incorporate these results into breeding simulations. PyBrOpS is built to be highly modular and has a script-based philosophy, making it highly extensible and customizable. In this paper, we describe some of the main features of PyBrOpS and demonstrate its ability to map Pareto frontiers for breeding possibilities and perform multi-objective selection in a simulated breeding pipeline.

Keywords: Stochastic Simulation; Python; Breeding Programs; Multi-Objective Optimization; Pareto Frontier

Introduction

The advent of modern computer processors has made many computationally intensive breeding-related tasks possible in recent years, including the use of simulation. There are four broad use-cases for computer simulation in breeding: evaluating breeding pipelines, assessing the efficacy of new genetic analysis techniques, studying gene networks and genotype by environment interaction, and modeling crop growth and development (Li et al., 2012). Among these four use-cases, the simulation of breeding

pipelines is perhaps the most practical application in a breeding program. Breeding pipeline simulations allow for breeders to approximate genetic gain and diversity metrics for a proposed breeding strategy. By repeating simulations, breeders may compare the efficacy of multiple proposed breeding strategies and assist in decision making.

In most circumstances, breeding is multi-objective in nature. In plant breeding, characteristics such as yield, standability, seed and fruit composition, resistance to biotic and abiotic stresses, biomass quality, adaptation to mechanization, and photoperiod response are potential breeding objectives (Fehr, 1991). Additionally, maintaining genetic diversity for long-term genetic gain is important (Akdemir et al., 2019; Allier et al., 2019a). The path to meeting multiple breeding objectives is not straightforward, unfortunately. Historically, tandem and index selection methods have been proposed as strategies to meet multiple breeding goals. In more recent years, numerical optimization and simulation techniques have been proposed as improvements to these methods.

This paper presents Python Breeding Optimizer and Simulator (PyBrOpS), a Python package capable of performing numerical optimization for problems in breeding and simulating breeding pipelines. Several breeding simulation frameworks have been developed in years prior including AlphaSim (Faux et al., 2016), AlphaSimR (Gaynor et al., 2021), DeltaGen (Jahufer and Luo, 2018), ADAM-Plant (Liu et al., 2019), MoBPS (Pook et al., 2020), QuLinePlus (Hoyos-Villegas et al., 2019), and XSimVersion 2 (Chen et al., 2022). PyBrOpS is unique among these packages in that it is explicitly designed to simulate and map Pareto frontiers for multiple breeding objectives.

Methods

Representation of a breeding program within PyBrOpS

Breeding programs are complex entities that can vary greatly based on a program's resource availability, breeding goals, and the species being bred. To properly constitute or simulate any breeding program, it is necessary to develop a formal description of the breeding program that is comprehensive, unambiguous, and reproducible (Simianer et al., 2021). In PyBrOpS, breeding programs are represented using Operators and Protocols. An Operator represents a process in a breeding program that takes an entire breeding program state, transforms it, and returns a modified breeding program state (Figure 1a). Operators are meant to be broad in functionality to encompass the wide diversity of breeding programs. They can be assembled into a Universal Breeding Algorithm from which a breeding program can be simulated (Figure 2). A Protocol represents a process that takes several inputs and returns a single output (Figure 1b). Protocols are meant to be used to construct the processes defined by Operators (Figure 2).

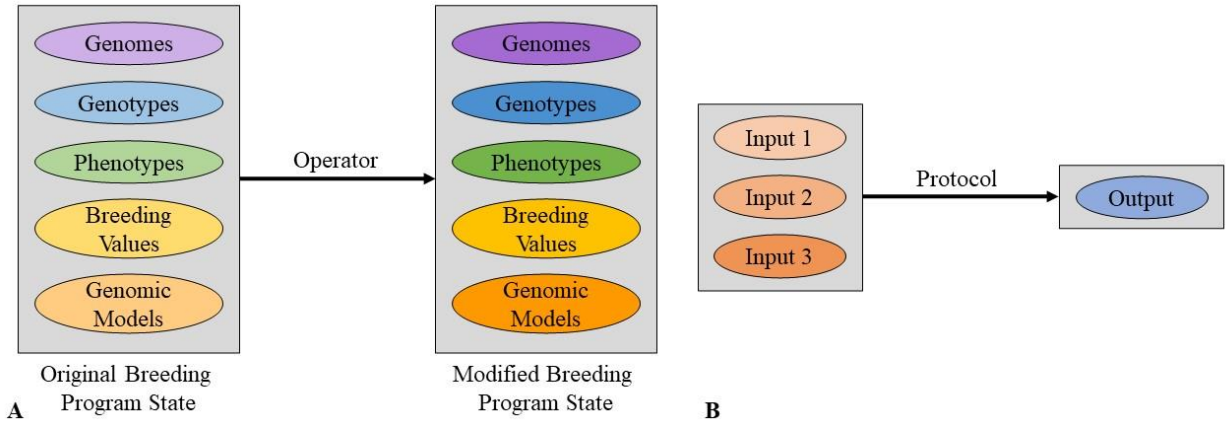

Figure 1: a) Breeding operators take entire breeding program states, which may consist of genomic, genotypic, phenotypic, breeding value, genomic model, and/or other types of data, and return a modified breeding program state. b) Breeding Protocols take data inputs and return a single output. The precise data types for each input and the output depends on the type of Breeding Protocol being used.

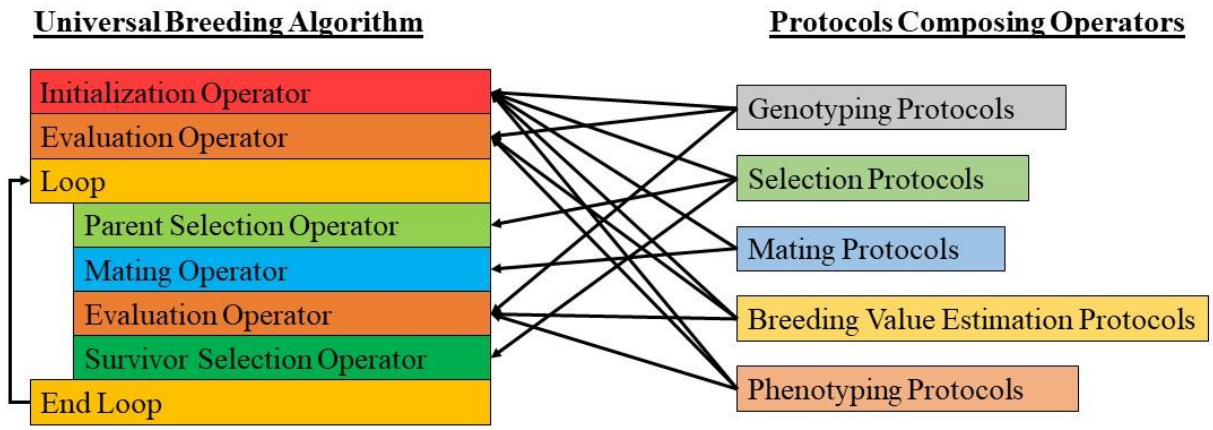

Figure 2: Left: Breeding Operators can be assembled into a Universal Breeding Algorithm. Outputs from each Operator are used as inputs for the next Operator to simulate a breeding program. Right: Breeding Protocols are modular can be used to construct Operators and define their functionality.

In PyBrOpS, there are five defined Operators: Initialization, Parental Selection, Mating, Evaluation, and Survivor Selection (Figure 2). Briefly, each Operator's purpose is described. The purpose of the Initialization Operator is to give a breeding program or breeding program simulation a starting point from which to proceed. This may be as simple as loading genotypic and phenotypic data into PyBrOpS from a real breeding program or it may define a set of burn-in generations preceding a breeding program simulation as is common in many research simulations (Sonesson et al., 2012; Liu et al., 2015; Müller et al., 2018; Akdemir et al., 2019; Allier et al., 2019a). The Evaluation Operator of a breeding program performs all tasks related to the measurement of the desirability of individuals in the breeding program. This encompasses the genotyping of individuals, the phenotyping of individuals in field trials for one or more traits using experimental design, the development and deployment of genomic prediction models, and the modeling and estimation of breeding values. The purpose of the Parental Selection operator is to utilize information provided by the Evaluation Operator to select from a list of candidates a set of parents and cross configuration mating those parents together to generate offspring. The purpose of the Reproduction Operator is to mate parents using the cross configuration selected by the Parental Selection

Operator and generate progenies according to its instruction. In breeding programs, this operator is interpreted as the process of meiosis and mutation. Finally, the purpose of the Survivor Selection Operator is to use the information provided by the Evaluation Operator to select a set of individuals to serve as parental candidates for the next breeding cycle. This operator is similar to the Parent Selection Operator, and its main purpose is to quickly narrow down the set of breeding candidates and merge breeding subpopulations within the program together. An example of a Survivor Selection Operator might be within-family truncating selection with selected family members being merged into a single population.

PyBrOpS defines five Protocols: Breeding Value Estimation, Genotyping, Mating, Phenotyping, and Selection. Briefly, each Protocol's purpose is described. The purpose of the Breeding Value Estimation Protocol is to provide routines for estimating breeding values from phenotypic and genotypic data. Mixed model analysis is nascent in Python, but users can interface with R via the package rpy2 (https://github.com/rpy2/rpy2) to utilize packages such as lme4 (Bates et al., 2015), or other mixed model analysis packages. The purpose of the Genotyping Protocol is to simulate genotyping of individuals in a breeding program. The Mating Protocol module simulates mating. PyBrOpS has protocols implemented for self-crosses, two-way crosses, three-way crosses, four-way crosses, and doubled haploid generation from two-, three-, and four-way crosses. The purpose of the Phenotyping Protocol is responsible for simulating phenotypes resulting from field trials. Finally, the purpose of the Selection Protocol is to provide methods for selecting individuals.

PyBrOpS implementation

PyBrOpS is implemented in pure Python and uses SOLID design principles in its architecture (Martin, 2018). SOLID design principles are a set of best practices that provide a framework for making software packages that are maintainable, comprehendible, modular, and extensible (Martin, 2018). Following SOLID principles, all objects in PyBrOpS implement an object interface – a contract defining standard

object method members and behaviors. This means that for almost all modules in PyBrOpS, if the user desires to provide additional functionality, the user need only implement a new class deriving from the interface.

PyBrOpS functionality

Genome and genotype representation

PyBrOpS represents and stores genomes and genotypes in Genotype Matrix objects, which serve as wrappers around dense, 8-bit integer NumPy arrays (Oliphant, 2006). Genomes are stored in three dimensional arrays, while genotypes are stored in two dimensional arrays. PyBrOpS has the capability to load external genomic and genotypic data from VCF files using cyvcf2 (Pedersen and Quinlan, 2017). If the user desires to implement a custom representation of a genome or genotype matrix, the user may implement a new class derived from a shared Genotype Matrix interface defining standard functionality for all Genotype Matrices.

Genetic map and genetic map function representation

PyBrOpS represents genetic maps using Genetic Map objects, which serve as wrappers around NumPy arrays. Genetic Map objects store positions in Morgans and support linear spline construction and interpolation of genetic positions from physical positions. If the user desires additional functionality, the user may create a new class that implements a Genetic Map interface defining standard functionalities for all Genetic Maps.

Associated with genetic maps are genetic map functions. PyBrOpS implements two Genetic Map Function classes representing the Haldane (Haldane, 1919) and Kosambi (Kosambi, 1943) map functions. These two functions can be used to simulate recombination or calculate marker covariance caused by recombination (Allier et al., 2019b).

Simulation of recombination

Recombination is simulated in a two-step process. In the first step, recombination probabilities between successive genomic markers are pre-computed using genetic map distances and a genetic map function and stored in a vector $\mathbf{p_{xo}}$. In the second step, a vector of uniform random numbers, $\mathbf{u}$, is generated such that $u_i \sim \text{Uniform}(0,1)\ \ \forall i$. If $u_i < (p_{xo})_i$, then a crossover occurs between the successive markers. Corresponding genome information is copied to progeny chromosomes.

Genomic model representation

PyBrOpS represents genomic models using Genomic Model classes that provide model fitting and multi-trait prediction functionality. PyBrOpS implements two linear genomic models that follow Fisher's model for quantitative genetics: a strictly additive linear genomic model, and an additive + dominance linear genomic model. The general mathematical representation of these models is as follows. Let n be the number of individuals, t be the number of traits, q be the number of fixed effect predictors, and p be the number of random effect predictors. Let $\mathbf{Y}$ be a $n \times t$ matrix of response variables, $\mathbf{X}$ be a $n \times q$ matrix of fixed effect predictors, $\mathbf{B}$ be a $q \times t$ matrix of fixed effect regression coefficients, $\mathbf{Z}$ be a $n \times p$ matrix of random effect predictors, and $\mathbf{U}$ be a $p \times t$ matrix of random effect regression coefficients. Then the general mathematical representation of a linear genomic model is:

$$\mathbf{Y} = \mathbf{XB} + \mathbf{ZU}$$

To model a strictly additive trait, we can interpret $\mathbf{X}$ as a $n \times 1$ matrix of ones and $\mathbf{B}$ as the intercept for each of t traits. If we assume that elements in $\mathbf{Z}$ represent genotype calls coded as 0, 1, and 2, and elements in $\mathbf{U}$ represent the effects of variants for each of t traits, then the formula reduces to the familiar model for an additive trait:

$$y_{ij} = \mu_j + \sum_{k=1}^{p} u_{ik} z_{ij}$$

for the ith individual, for the jth trait. This simplifies to Fisher's equations assuming that the dominance effect is 0.

To model an additive + dominance trait, we once again interpret $\mathbf{X}$ as a $n \times 1$ matrix of ones, and $\mathbf{B}$ as the intercept for each of t traits. $\mathbf{Z}$ is interpreted as a block matrix composed of a genotype block ($\mathbf{Z_a}$), and a dominance indicator block ($\mathbf{Z_d}$):

$$\mathbf{Z} = [\mathbf{Z_a} \quad \mathbf{Z_d}]$$

Elements in $\mathbf{Z_a}$ represent genotype calls as 0, 1, and 2. Elements in $\mathbf{Z_d}$ represent binary, indicator variables for whether an individual is heterozygous at a particular locus. Specifically, elements of $\mathbf{Z_d}$ are determined as $(z_d)_{ij} = I_{((z_a)_{ij}=1)}\big((z_a)_{ij}\big)$. The $\mathbf{U}$ matrix is interpreted as a block matrix containing additive effects ($\mathbf{U_a}$) and dominance effects ($\mathbf{U_d}$) for each of t traits:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U_a} \\ \mathbf{U_d} \end{bmatrix}$$

Putting all this together, then the formula reduces to the familiar model for an additive trait with dominance effects:

$$y_{ij} = \mu_j + \sum_{k=1}^{p} (u_a)_{ik} (z_a)_{ij} + \sum_{k=1}^{p} (u_d)_{ik} (z_d)_{ij}$$

for the ith individual, for the jth trait. If we assume that the dominance effects are defined as $(u_d)_{ik} = \mathrm{sign}((u_a)_{ik}) \left|\delta_{ij}\right|$ for some δ_{ij}, then this simplifies to Fisher's equations.

Epistatic and nonlinear models are not implemented in PyBrOpS, but using the Genomic Model interfaces, one may implement custom epistatic or nonlinear models.

<u>Multi-objective selection</u>

PyBrOpS implements several Selection Protocols including conventional genomic selection (Meuwissen et al., 2001), conventional phenotypic selection, within family phenotypic selection, optimal contribution selection (Meuwissen, 1997), optimal haploid value selection (Daetwyler et al., 2015), optimal population value selection (Goiffon et al., 2017), expected maximum breeding value selection (Müller et al., 2018), weighted genomic selection (Jannink, 2010), and random selection. Selection Protocols provide several methods useful for making selections in physical breeding programs and in-silico breeding simulations. The first method is the `pareto` method. Given genomes, genotypes, phenotypes, breeding values, and a genomic prediction model, the `pareto` method estimates a set of non-dominated selection decisions lying on the Pareto frontier for a multi-objective problem. The user may utilize a default NSGA-II (Deb et al., 2002) or NSGA-III (Deb and Jain, 2014) algorithm or provide a custom multi-objective optimization algorithm to the Selection Protocols to map a Pareto frontier. The `pareto` method returns a $n \times d$ matrix containing non-dominated points for the n individuals along the frontier for d dimensional space along with a list of corresponding solutions in the decision space for the n individuals along the frontier.

The second method is the `select` method. Given genomes, genotypes, phenotypes, breeding values, and a genomic prediction model, the `select` method calculates a set of selection decisions and cross configurations following the strategy defined within the selection protocol class. In most selection protocol implementations in PyBrOpS, the select routine has two different behavior modes: a single-objective selection mode and a Pareto selection mode. In the single-objective selection mode, it is assumed that a transformation function T has been provided that reduces the objective space to a single objective: $T(\mathbb{R}^d) \mapsto \mathbb{R}$. For some problems, this may be as simple as providing a linear weighting function. From there, a single-objective optimization algorithm is used to identify a solution. In the Pareto selection mode, first the Pareto frontier for a given multi-objective optimization problem is approximated using the `pareto` method. Next, the approximated Pareto frontier set is transformed by a non-dominated set transformation function T^{NDset}. This function accepts objective coordinates for the whole non-

dominated set and assigns a single score to each individual: $T^{NDset}\left(\mathbb{R}^{n\times d}\right) \mapsto \mathbb{R}^{n}$. In some applications, this may be a function that calculates the distance between a preference weight vector and a non-dominated point. Finally, the individual with the best score is selected. The Pareto selection mode may be useful in situations where Pareto frontiers are non-convex. In situations where the Pareto frontier is non-convex, simple linear weighting methods will fail to find points in non-convex portions of the Pareto frontier (Deb, 2001).

Results

To demonstrate the capabilities of PyBrOpS, two use-case scenarios are exhibited. In scenario 1, a Pareto frontier is mapped for Optimal Contribution Selection (Meuwissen, 1997) of two synthetic traits. In scenario 2, Pareto frontiers are mapped and subsequently used to make selections for two synthetic traits using Conventional Genomic Selection (Meuwissen et al., 2001).

Simulation setup

In both scenarios, 2000 randomly selected markers with minor allele frequencies greater than 0.2 from the Wisconsin Diversity Panel 942 (Hirsch et al., 2014; Mazaheri et al., 2019) were used to serve as empirical marker sources. The US NAM maize linkage map (McMullen et al., 2009) was used to linearly interpolate genetic map positions for each of the 2000 selected markers. For each marker, strictly additive effects were assigned for the two synthetic traits following a multivariate normal distribution with negative covariance to simulate two traits that are competing in nature:

$$\mathbf{U} \sim MVN\left(\mathbf{0}, \begin{bmatrix} 1 & -0.4 \\ -0.4 & 1 \end{bmatrix}\right)$$

The intercepts for synthetic traits 1 and 2 were 10 and 25, respectively.

In scenario 1, true breeding values for each individual were calculated using the assigned genomic model. Additionally, kinships between individuals were calculated as described by (Allier et al., 2019a). Using

the breeding values and kinship matrices as inputs, the Pareto frontier was mapped for optimal parental contributions for synthetic trait 1, synthetic trait 2, and mean parental inbreeding with NSGA-III. The results of the optimization can be viewed in Figure 3.

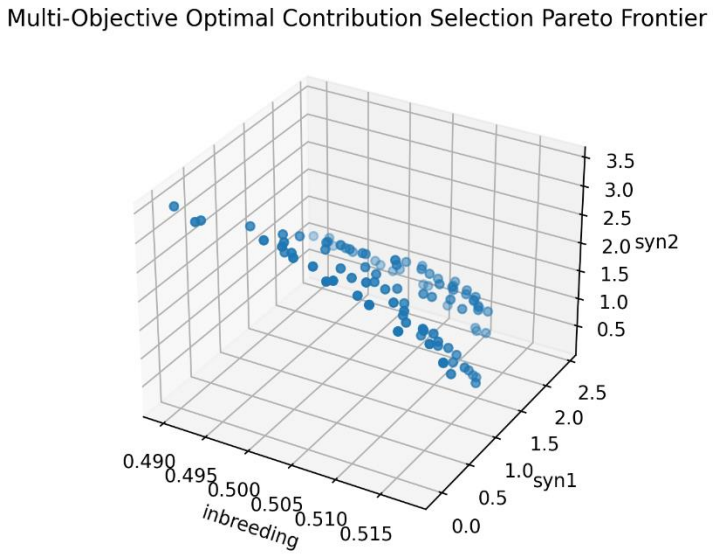

Figure 3: Three-dimensional Pareto frontier for Optimal Contribution Selection in Scenario 1. The x-axis represents mean population inbreeding, the y-axis represents genetic gain for synthetic trait 1, and the z-axis represents genetic gain for synthetic trait 2.

In scenario 2, simulations were performed following protocol similar to what is described in (Allier et al., 2019a). 40 unique, randomly selected genotypes were randomly intermated to serve as founders for the breeding simulation. Narrow sense heritabilities of 0.4 and 0.6 were assigned to synthetic traits 1 and 2, respectively. The simulation was initiated with 10 generations of phenotypic selection for burn-in to mimic linkage disequilibrium patterns that resemble those found in real breeding programs. In the burn-in stage, the top 40 individuals were selected and randomly intermated in 20 biparental crosses to produce 80 progenies per cross. The progenies were evaluated in 4 locations assuming no genotype by

environment interactions. The top 5% (4 individuals) from each family were selected to serve as parental candidates for the next generation. Parental candidates from the previous three generations were considered in the selection of founders for the next generation.

After the burn-in, the true genomic model was used to make selections using conventional genomic selection. Parental selections were made using the Pareto method whereby NSGA-II was used to map Pareto optimal parental selection configurations for both traits. Trait value ranges for the Pareto optimal set were scaled to the range [0,1]. The parental selection configuration closest to a weight of 0.5 for both scaled traits was used to select the 40 founders for the next generation. Crossing and progeny generation protocols were identical to that of the burn-in. The simulation lasted 10 generations.

Throughout the simulation, the Pareto frontier at each selection stage was saved (Figure 4). Additionally, mean population trait genotypic values were saved and are visualized in (Figure 5).

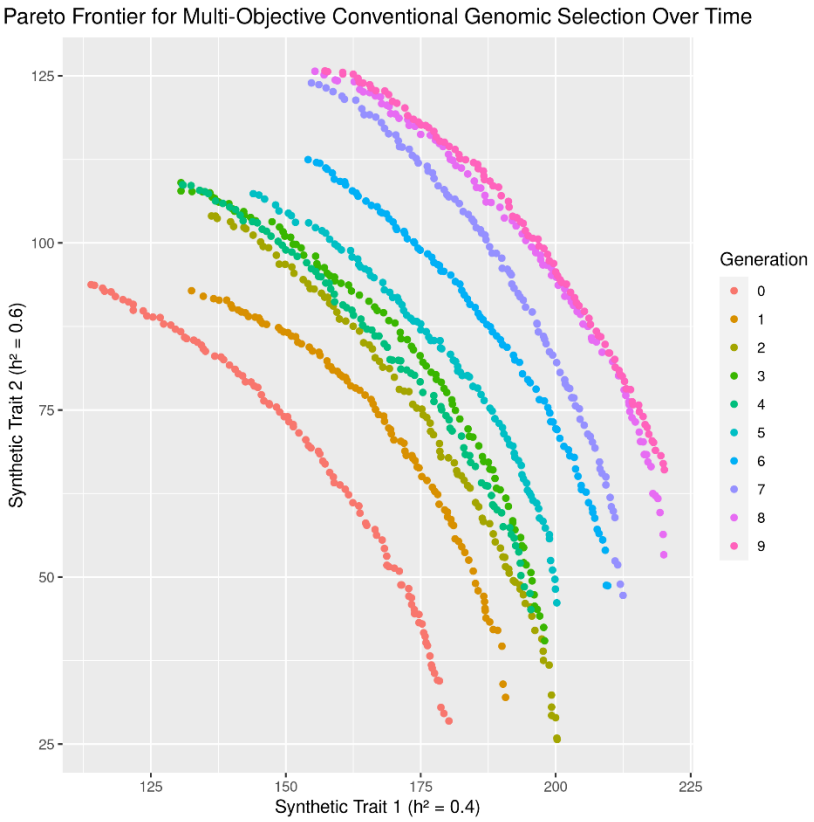

Figure 4: Pareto frontiers mapped by NSGA-II at each generation in Scenario 2 simulations. The Pareto frontier at Generation 0 represents the initial population selection possibilities, while the frontier at Generation 9 represents the selection possibilities before the final mating round in the simulation. Individual points along each Pareto frontier represent potential breeding decisions a breeder could make.

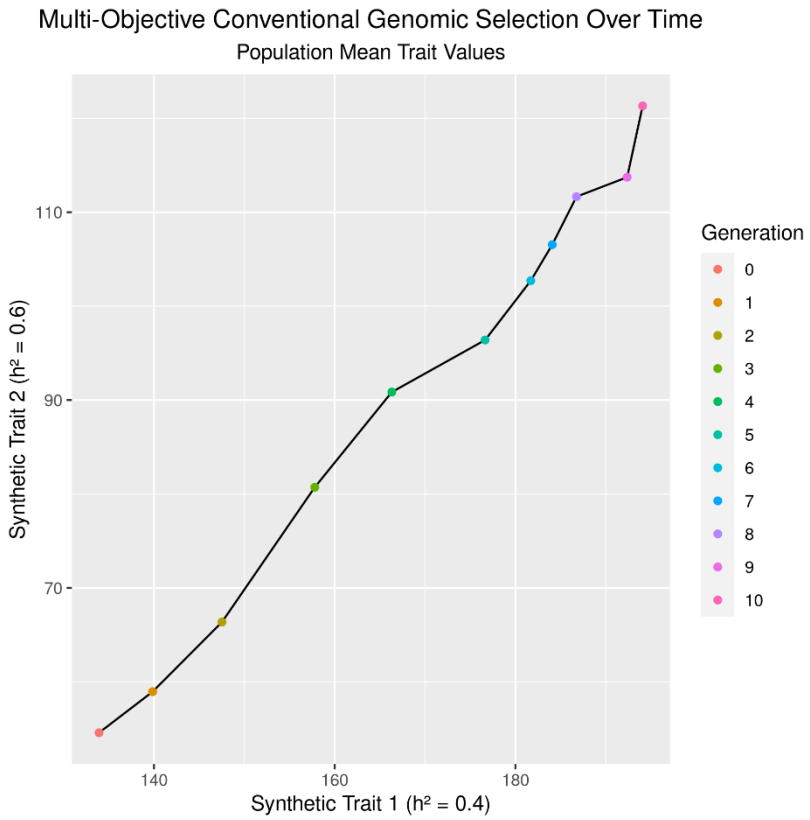

Figure 5: Mean trait values for the breeding population in Scenario 2 simulations.

Conclusion

PyBrOpS is a Python package for simulating breeding programs and performing multi-objective optimizations. PyBrOpS has the capability to simulate multiple quantitative traits, simulate the evolution of entire breeding programs, and map breeding frontier possibilities for these traits. Stochastic simulations serve as a valuable tool in a breeding program and can be used to optimize a pipeline for genetic gain, genetic diversity, economic value, or other program objectives. Additionally, the native ability of PyBrOpS to map breeding frontiers for multiple traits is of particular interest to breeders, decision-makers, and other stakeholders. Mapping breeding frontiers facilitates the visualization and understanding of tradeoffs and can assist in decision making.

Due to the high degree of modularity built into PyBrOpS's architecture, new features are easy to implement. To provide additional functionality, the user may implement an interface relevant to the feature desired to be added. Future versions of PyBrOpS will include additional selection protocols and offer genomic prediction routines native to Python.

Web resources

PyBrOpS is open source and can be found at https://github.com/rzshrote/pybrops/. Source code for simulations can be found within the "examples" directory at the repository above. Feature requests, bug reports, and feature submissions may be made at the GitHub link aforementioned.

References

Akdemir D, Beavis W, Fritsche-Neto R, Singh AK, Isidro-Sánchez J (2019) Multi-objective optimized genomic breeding strategies for sustainable food improvement. Heredity (Edinb) **122**: 672–683

Allier A, Lehermeier C, Charcosset A, Moreau L, Teyssèdre S (2019a) Improving Short- and Long-Term Genetic Gain by Accounting for Within-Family Variance in Optimal Cross-Selection . Frontiers in Genetics **10**: 1006

Allier A, Moreau L, Charcosset A, Teyssèdre S, Lehermeier C (2019b) Usefulness Criterion and Post-selection Parental Contributions in Multi-parental Crosses: Application to Polygenic Trait Introgression. G3: Genes|Genomes|Genetics **9**: 1469 LP – 1479

Bates D, Mächler M, Bolker B, Walker S (2015) Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software; Vol 1, Issue 1 (2015) . doi: 10.18637/jss.v067.i01

Chen CJ, Garrick D, Fernando R, Karaman E, Stricker C, Keehan M, Cheng H (2022) XSim version 2: simulation of modern breeding programs. G3 Genes|Genomes|Genetics **12**: jkac032

Daetwyler HD, Hayden MJ, Spangenberg GC, Hayes BJ (2015) Selection on Optimal Haploid Value Increases Genetic Gain and Preserves More Genetic Diversity Relative to Genomic Selection. Genetics **200**: 1341 LP – 1348

Deb K (2001) Multi-objective optimization using evolutionary algorithms, 1st ed. John Wiley & Sons, Ltd, Chichester, West Sussex, England

Deb K, Jain H (2014) An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. IEEE Transactions on Evolutionary Computation **18**: 577–601

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**: 182–197

Faux A-M, Gorjanc G, Gaynor RC, Battagin M, Edwards SM, Wilson DL, Hearne SJ, Gonen S, Hickey JM (2016) AlphaSim: Software for Breeding Program Simulation. Plant Genome **9**: plantgenome2016.02.0013

Fehr WR (1991) Principles of Cultivar Development, First Edition. MacMillian Publishing Company

Gaynor RC, Gorjanc G, Hickey JM (2021) AlphaSimR: an R package for breeding program simulations. G3 Genes|Genomes|Genetics. doi: 10.1093/g3journal/jkaa017

Goiffon M, Kusmec A, Wang L, Hu G, Schnable PS (2017) Improving Response in Genomic Selection with a Population-Based Selection Strategy: Optimal Population Value Selection. Genetics **206**: 1675 LP – 1682

Haldane JBS (1919) The combination of linkage values, and the calculation of distances between. J Genet **8**: 299

Hirsch CN, Foerster JM, Johnson JM, Sekhon RS, Muttoni G, Vaillancourt B, Peñagaricano F, Lindquist E, Pedraza MA, Barry K, et al (2014) Insights into the maize pan-genome and pan-transcriptome. Plant Cell **26**: 121–135

Hoyos-Villegas V, Arief VN, Yang W-H, Sun M, DeLacy IH, Barrett BA, Jahufer Z, Basford KE (2019) QuLinePlus: extending plant breeding strategy and genetic model simulation to cross-pollinated populations—case studies in forage breeding. Heredity (Edinb) **122**: 684–695

Jahufer MZZ, Luo D (2018) DeltaGen: A Comprehensive Decision Support Tool for Plant Breeders. Crop Sci **58**: 1118–1131

Jannink J-L (2010) Dynamics of long-term genomic selection. Genetics Selection Evolution **42**: 35

Kosambi DD (1943) The estimation of map distances from recombination values. Ann Eugen **12**: 172–175

Li X, Zhu C, Wang J, Yu J (2012) Chapter six - Computer Simulation in Plant Breeding. *In* DL Sparks, ed, Advances in Agronomy. Academic Press, pp 219–264

Liu H, Meuwissen THE, Sørensen AC, Berg P (2015) Upweighting rare favourable alleles increases long-term genetic gain in genomic selection programs. Genetics Selection Evolution **47**: 19

Liu H, Tessema BB, Jensen J, Cericola F, Andersen JR, Sørensen AC (2019) ADAM-Plant: A Software for Stochastic Simulations of Plant Breeding From Molecular to Phenotypic Level and From Simple Selection to Complex Speed Breeding Programs . Frontiers in Plant Science **9**: 1926

Martin RC (2018) Clean Architecture. Pearson Education, Inc.

Mazaheri M, Heckwolf M, Vaillancourt B, Gage JL, Burdo B, Heckwolf S, Barry K, Lipzen A, Ribeiro CB, Kono TJY, et al (2019) Genome-wide association analysis of stalk biomass and anatomical traits in maize. BMC Plant Biol **19**: 45

McMullen MD, Kresovich S, Villeda HS, Bradbury P, Li H, Sun Q, Flint-Garcia S, Thornsberry J, Acharya C, Bottoms C, et al (2009) Genetic properties of the maize nested association mapping population. Science (1979) **325**: 737–740

Meuwissen THE (1997) Maximizing the response of selection with a predefined rate of inbreeding. J Anim Sci **75**: 934–940

Meuwissen THE, Hayes BJ, Goddard ME (2001) Prediction of Total Genetic Value Using Genome-Wide Dense Marker Maps. Genetics **157**: 1819–1829

Müller D, Schopp P, Melchinger AE (2018) Selection on Expected Maximum Haploid Breeding Values Can Increase Genetic Gain in Recurrent Genomic Selection. G3: Genes|Genomes|Genetics **8**: 1173 LP – 1181

Oliphant TE (2006) Guide to NumPy. Trelgol Publishing

Pedersen BS, Quinlan AR (2017) cyvcf2: fast, flexible variant analysis with Python. Bioinformatics **33**: 1867–1869

Pook T, Schlather M, Simianer H (2020) MoBPS - Modular Breeding Program Simulator. G3: Genes|Genomes|Genetics **10**: 1915 LP – 1918

Simianer H, Büttgen L, Ganesan A, Ha NT, Pook T (2021) A unifying concept of animal breeding programmes. Journal of Animal Breeding and Genetics **138**: 137–150

Sonesson AK, Woolliams JA, Meuwissen THE (2012) Genomic selection requires genomic control of inbreeding. Genetics Selection Evolution **44**: 27

373