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 for 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

1. 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, breeding pipeline simulation is perhaps the most practical application in a breeding program. Breeding pipeline simulations allow for breeders to estimate genetic gain and diversity metrics for a proposed breeding strategy without having to spend time and resources empirically testing them. By repeating simulations, breeders may compare the expected performance of multiple proposed breeding strategies, aiding in decision-making processes.

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 (Gorjanc et al., 2018; Akdemir et al., 2019; Allier et al., 2019a). The path to meeting multiple breeding objectives is not straightforward, unfortunately. Historically, tandem selection, independent culling, and index selection methods have been proposed as strategies to meet multiple breeding goals (Bernardo, 2020). Among these three historical selection strategies, index selection methods are generally regarded as being more effective (Bernardo, 2020).

In more recent years, multi-objective numerical optimization and simulation techniques have been proposed as improvements to traditional methods (Akdemir and Sánchez, 2016; De Beukelaer et al., 2017; Akdemir et al., 2019; Allier et al., 2019a; Moeinizade et al., 2020; Amini et al., 2022). For example, Akdemir and Sánchez (2016) combined an inbreeding objective with a mating risk objective related to the mean and variance of progenies using a weighted sum. De Beukelaer et al. (2017) combined a genetic gain objective with a rare allele management objective using a weighted sum. Akdemir et al. (2019) proposed a non-dominated selection strategy and multi-objective versions of optimal contribution selection (Meuwissen, 1997) and genomic mating (Akdemir and Sánchez, 2016). Allier et al. (2019a) combined a cross usefulness objective with an inbreeding objective cognizant of biased parental genome contributions using the ε-constraint method. Moeinizade et al. (2020) proposed a simulation-based look-

ahead-selection strategy for multiple traits. Finally, Amini et al. (2022) proposed a multi-trait, L-shaped index selection strategy to address the shortcomings of traditional index selection techniques. Among these cited selection strategies, most transform multiple objectives into a single objective using weight or constraint parameters determined *a priori* and are consequently solved as single-objective optimization problems. In Akdemir and Sánchez (2016) and Akdemir et al. (2019), the authors utilized a truly multi-objective optimization strategy. They performed multiple optimizations over a range of objective weights and identified efficient frontiers for their objectives.

Multi-objective optimization techniques offer two main benefits over traditional selection techniques. First, multi-objective optimization techniques eliminate or reduce the difficulty of assigning weights to breeding objectives by identifying a set of Pareto optimal or near-Pareto optimal, non-dominated solutions from which to choose. A solution is Pareto optimal if there exists no other solution in the selection search space that can improve upon one objective without making another objective worse. A solution is non-dominated in the context of a set of solutions if there exists no other solution in the set that can improve upon one objective without making another objective worse. With multi-objective optimization, breeders may visualize identified solutions, examine the tradeoffs between various objectives, and choose from the available points without having to assign weights as one would in an index selection. If desired, pseudo-weights, which are normalized to the range [0,1] and indicate the relative importance of an objective among solutions lying along the Pareto frontier, may also be assigned to identified solutions and serve as a more convenient way to make selection decisions (Deb, 2001). Second, multi-objective optimization techniques allow for the identification of Pareto-optimal solutions that do not lie on the convex hull of the selection search space. A search space is considered convex if a line connecting any two points within the search space lies entirely within the search space. A non-convex search space is one where this is not true for all points. As Amini et al., 2022 emphasized, in discrete selection problems where the Pareto frontier is non-convex, it is mathematically impossible to identify Pareto optimal solutions that do not lie along the convex hull using a linear selection index. This is problematic, as there may exist superb solutions in non-convex regions of the Pareto frontier. A good multi-objective optimization algorithm will be able to identify multiple solutions lying in non-convex Pareto frontier regions from which to choose.

It is common to use evolutionary algorithms to solve multi-objective optimization problems. Evolutionary algorithms are particularly suitable to solving multi-objective optimization problems because they are population based, which allows them to find multiple Pareto optimal or near-Pareto optimal solutions in a single run (Coello Coello et al., 2007). Additionally, evolutionary algorithms are less susceptible to getting trapped in local optima and make few assumptions about the shape and continuity of the Pareto frontier (Coello Coello et al., 2007). In contrast, classical multi-objective optimization techniques such as the weighted sum method, the ε-constraint method, weighted metric methods, value function methods, goal programming methods, and min-max optimization can only find a single solution from a single algorithm run, and some make assumptions about the shape and continuity of the Pareto frontier or require problem knowledge such as a set of suitable weights or target values (Deb, 2001; Coello Coello et al., 2007). Because of their advantages, many multi-objective evolutionary algorithms have been developed. Notable multi-objective evolutionary algorithms include PAES (Knowles and Corne, 1999), SPEA (Zitzler and Thiele, 1999), PESA (Corne et al., 2000), SPEA2 (Zitzler et al., 2001), NSGA-II (Deb et al., 2002), MOEA/D (Zhang and Li, 2007), SMS-MOEA (Beume et al., 2007), and NSGA-III (Deb and Jain, 2014).

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 multiple traits and estimate Pareto frontiers for multiple breeding objectives.

2. Methods

2.1. 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. A Protocol represents a process in a breeding program that takes several inputs and returns a single output (Figure 1B). Protocols are meant to be narrow and specific in their functionality and used to construct the processes defined by Operators (Figure 2). Operators, once defined, can be assembled into a Universal Breeding Algorithm from which a breeding program can be simulated (Figure 2).

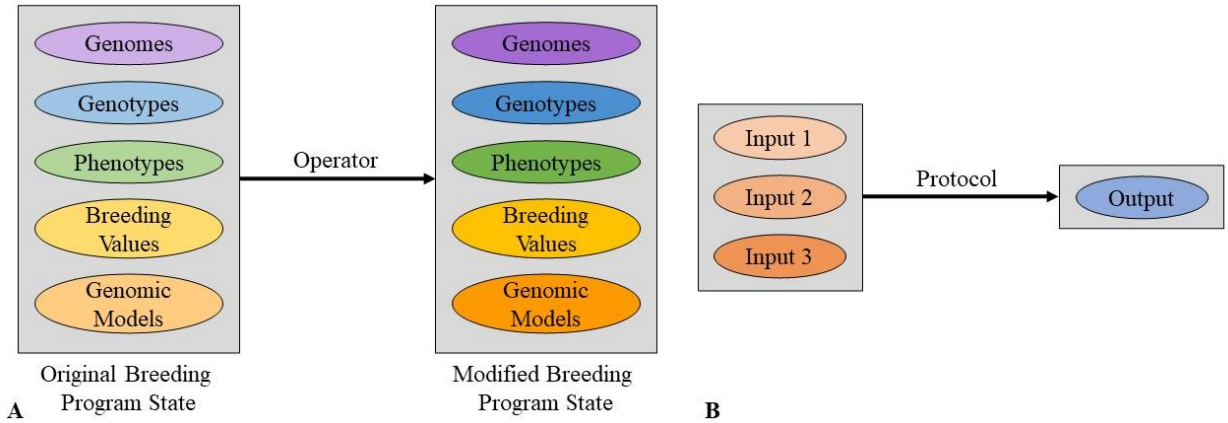

Figure 1: A) Breeding operators take entire breeding program states, which may consist of genomic, genotypic, phenotypic, breeding value, genomic model, and/or other types of data, and return a modified breeding program state. B) Breeding Protocols take data inputs and return a single output. The precise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

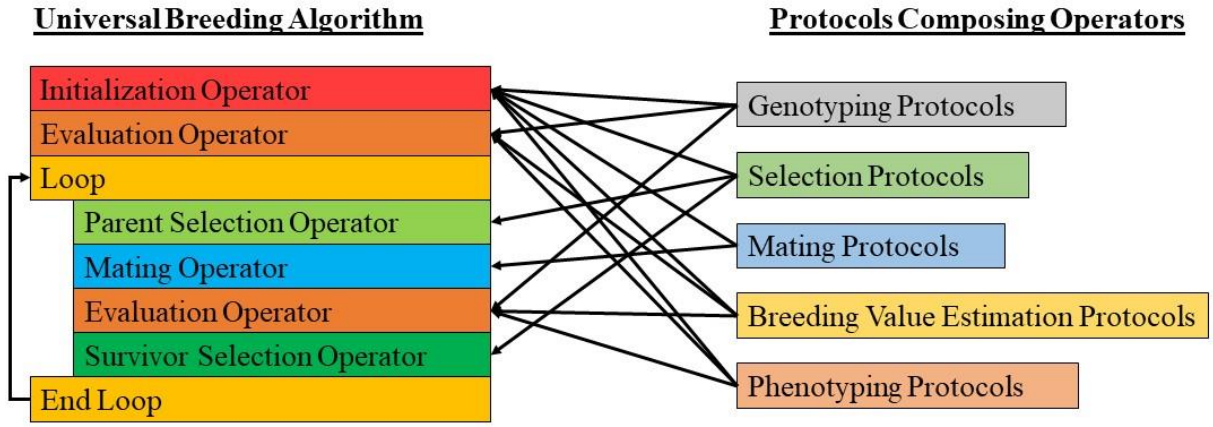

Figure 2: Left: Breeding Operators can be assembled into a Universal Breeding Algorithm. Outputs from each Operator are used as inputs for the next Operator to simulate a breeding program. Right: Breeding Protocols are modular and can be used to construct Operators and define their functionality.

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 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 a corresponding cross configuration mating those parents together to generate offspring. The purpose of the Mating Operator is to mate parents using the cross configuration selected by the Parental Selection Operator, simulate recombination, and generate progenies according to its instruction. 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 like the Parent Selection Operator, but its main purpose is to quickly narrow down the set of breeding candidates to a manageable quantity and merge breeding subpopulations within the program. An example of a Survivor Selection Operator might perform within-family truncating selection followed by merging of selected family members 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 may utilize the statsmodels package (Seabold and Perktold, 2010) to fit linear mixed models and define custom Breeding Value Estimation Protocols. The purpose of the Genotyping Protocol is to take genomic information and simulate genotyping of individuals in a breeding program. The Mating Protocol module simulates recombination and 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. Random or uncontrolled mating, as may be prevalent in some species, can be simulated manually using implemented modules. The purpose of the Phenotyping Protocol is to simulate phenotypes resulting from field trials given genomic information. PyBrOpS implements a basic multi-environment protocol without genotype by environment effects. Finally, the purpose of the Selection Protocol is to provide methods for selecting individuals based on their genetic and/or phenotypic merits.

2.2. 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 or novel functionality, the user need only implement a new class deriving from the interface. Objects implementing the same interface may be used interchangeably since routines depend on the interface abstraction and not any given object implementation.

2.3. PyBrOpS functionality

2.3.1. *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) and create Genotype Matrices from raw NumPy arrays. Furthermore, genome and genotype data may be imported from or exported to a binary HDF5 format for convenient creation of breakpoints in a breeding simulation. 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.

2.3.2. *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. PyBrOpS can load and save genetic map positions from and to CSV-like formats. If the user desires additional functionality, the user may create a new class that implements the Genetic Map interface defining standard functionalities for all Genetic Map classes.

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).

2.3.3. *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 the ith random number is less than the ith recombination probability, $u_i < (p_{xo})_i$, then a crossover occurs between the successive markers. Corresponding genome information is copied to progeny chromosomes.

2.3.4. *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:

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

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 an $n \times p_a$ genotype block ($\mathbf{Z_a}$), and an $n \times p_d$ 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 $(z_d)_{ik} = I_{((z_a)_{ik}=1)}((z_a)_{ik})$. The $\mathbf{U}$ matrix is interpreted as a block matrix containing $p_a \times t$ additive effects ($\mathbf{U_a}$) and $p_d \times t$ 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:

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

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

Models may be imported from and exported to Pandas DataFrames, CSV-like formats, and HDF5 files, allowing users to use models fit with popular R packages like rrBLUP (Endelman, 2011) or BGLR (Pérez and de los Campos, 2014). Epistatic and nonlinear models are not implemented in PyBrOpS, but using the Genomic Model interfaces, one may implement custom epistatic or nonlinear models.

2.3.5. ***Breeding value representation***

PyBrOpS represents breeding values with Breeding Value Matrix objects. Breeding Value Matrix objects are wrappers around NumPy arrays and store relevant information such as names of the traits and individuals whose breeding values are stored in the matrix. PyBrOpS can load and save breeding values from Pandas DataFrames, CSV-like formats, and HDF5.

2.3.6. ***Single- and multi-objective selection***

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 four methods useful for making selections in physical breeding programs and in-silico breeding simulations.

The first method is the `problem` method. The `problem` method constructs a Selection Problem object from provided genomes, genotypes, phenotypes, breeding values, and/or a genomic prediction model. Selection Problems define an optimization problem and encapsulate all required data and routines required to conduct optimizations for the defined problem. Selection Problems define the following optimization problem:

$$\min_{\mathbf{x}} T_{obj}\big(L(\mathbf{x})\big)$$

Such that:

$$T_{ineq}\big(L(\mathbf{x})\big) \leq 0$$
$$T_{eq}\big(L(\mathbf{x})\big) = 0$$
$$\mathbf{x} \in \Omega$$

Here, $\mathbf{x}$ is a decision vector that represents a selection decision in some selection decision space Ω. The meaning of selection decision depends on the context. If $\mathbf{x}$ is a subset, it represents the subset of individuals or crosses chosen for selection. If $\mathbf{x}$ is binary valued, it represents whether individuals or crosses are selected (1) or not (0). If $\mathbf{x}$ is integer-valued, it represents an integer-valued contribution of an individual or cross to selection. Finally, if $\mathbf{x}$ is real-valued, it represents a real-valued contribution of an individual or cross to selection. Next, $L(\mathbf{x})$ is a latent function that maps a selection decision to an l-dimensional latent vector: $L(\mathbf{x}) \mapsto \mathbb{R}^l$. The latent function is unique to the Selection Problem and its dimension is determined by the Selection Protocol constructing the Selection Problem. The latent vector contains all objective and constraint evaluation information for a given selection strategy. Finally, T_{obj}, T_{ineq}, and T_{eq} are transformation functions provided by the user to transform the latent vector into objectives, inequality constraints, and equality constraints, respectively. By providing different transformation functions, the user may convert various elements of the latent vector into constraints, or even new objectives. Latent vector transformation functions are transmitted from the Selection Protocol object to the Selection Problem object when the former constructs the latter. Selection Problem objects have methods to evaluate candidate solutions, allowing the user to manually evaluate the desirability of a proposed solution. All Selection Problem objects are also compatible with PyMOO (Blank and Deb, 2020) and can be solved with PyMOO infrastructure if desired.

The second method is the `sosolve` method. The `sosolve` method, given genomes, genotypes, phenotypes, breeding values, and/or a genomic prediction model, first creates a Selection Problem and then, assuming that the Selection Problem is single-objective in nature, solves it using a single-objective optimization algorithm. A generic genetic algorithm is the default optimization algorithm, but the user may utilize pre-built hillclimber algorithms or provide a custom single-objective optimization algorithm. Similarly, the third method, `mosolve`, creates a Selection Problem and solves the Selection Problem with a multi-objective optimization algorithm, assuming that the Selection Problem is multi-objective in nature. A generic NSGA-II (Deb et al., 2002) algorithm is the default multi-objective optimization algorithm, but the user may also utilize NSGA-III (Deb and Jain, 2014) or provide a custom multi-objective optimization algorithm to solve the problem. Both the `sosolve` and `mosolve` methods return a Selection Solution object, which encapsulates the solution or solutions to the optimization problem.

The final method is the `select` method. Given genomes, genotypes, phenotypes, breeding values, and/or a genomic prediction model, the `select` method constructs a Selection Problem, and solves the problem using an appropriate single- or multi-objective problem. If the problem is single-objective, it constructs and returns a Selection Configuration object from the singular solution obtained from the optimization. If the problem is multi-objective, then the solution is a set of non-dominated solutions. Since selection necessitates that a single decision is made, solutions in the non-dominated set are transformed by a non-dominated set transformation function T_{ND}. This function accepts objective coordinates for the whole non-dominated set and assigns a single score to each solution: $T_{ND}\big(\mathbb{R}^{n\times d}\big) \mapsto \mathbb{R}^n$. In some applications, this may be a function that calculates the distance between a preference weight vector and a non-dominated point. The solution with the best score is used to construct a Selection Configuration object. Selecting points from a non-dominated set of solutions may be useful in situations where the Pareto frontier is 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). In both single- and multi-objective selection scenarios, the returned Selection Configuration object encapsulates all information necessary to identify the individuals or crosses that should be made. This information can be used to drive simulations or inform the breeder.

3. 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).

3.1. <u>Example Scenario 1: Mapping a Pareto frontier for Optimal Contribution Selection</u>

In the first use-case scenario, 2000 markers were randomly selected with minor allele frequencies greater than 0.2 from the Wisconsin Maize Diversity Panel 942 (Hirsch et al., 2014; Mazaheri et al., 2019) 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 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 set to 10 and 25, respectively.

A multi-objective optimization of a subset variant of optimal contribution selection (Meuwissen, 1997) was conducted and can be summarized as follows:

$$\max_{\mathbf{x}} \frac{1}{J} \mathbf{bv_1}' \mathbf{x}$$
$$\max_{\mathbf{x}} \frac{1}{J} \mathbf{bv_2}' \mathbf{x}$$
$$\min_{\mathbf{x}} \frac{1}{J^2} \mathbf{x}' \mathbf{K} \mathbf{x}$$

Such that:

$$\mathbf{1}' \mathbf{x} = J$$
$$\mathbf{x} \in \{0,1\}^n$$

Here, $\mathbf{x}$ represents a subset of J individuals selected from the diversity panel. $\mathbf{bv_1}$ and $\mathbf{bv_2}$ represent the true breeding values of panel members for synthetic traits 1 and 2, respectively, using the assigned genomic model. $\mathbf{K}$ represents the kinship relationships between individuals using methods described by Allier et al., 2019a. The first two objectives can be interpreted as maximizing the mean breeding value for synthetic traits 1 and 2 in the selected subset. The third objective can be interpreted as minimizing the average kinship between individuals in the selected subset.

An NSGA-III (Deb and Jain, 2014) algorithm with a population size of 100 and 91 reference points was evolved for 1500 generations to approximate the Pareto frontier for the problem formulated above and identify a set of relatively evenly distributed non-dominated points for each of the three objectives (Figure 3). While appearing to be smooth and convex from a distance, the estimated Pareto frontier contains regions that are locally non-convex. These local non-convexities are likely due to the discrete nature of the optimization problem but could also be attributable to premature convergence to the Pareto frontier.

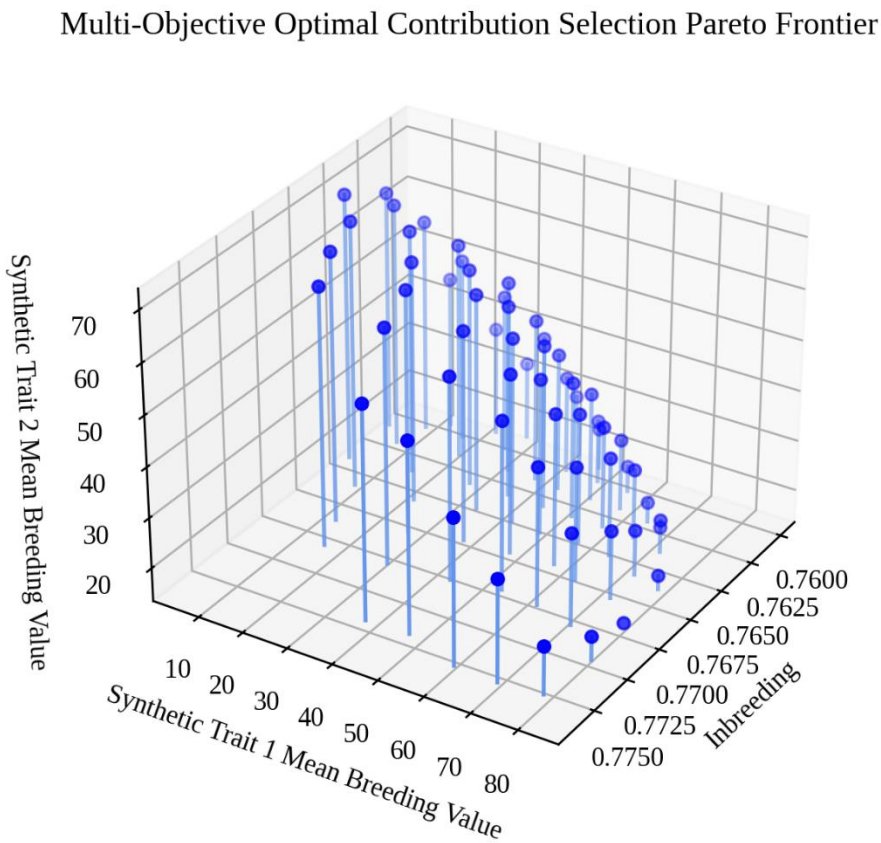

Figure 3: Three-dimensional Pareto frontier for Optimal Contribution Selection in Scenario 1, created using Matplotlib (Hunter, 2007) in Python. The x-axis represents mean inbreeding of the selected subset, the y-axis represents the mean breeding value (genetic gain) of the selected subset for synthetic trait 1, and the z-axis represents mean breeding value (genetic gain) of the selected subset for synthetic trait 2.

3.2. <u>Example Scenario 2: Multi-objective breeding program simulation</u>

In the second use-case scenario, 1000 markers with minor allele frequencies greater than 0.2 were randomly sampled from the Wisconsin Maize Diversity Panel 942 (Hirsch et al., 2014; Mazaheri et al., 2019), their genetic map distances linearly interpolated using the US NAM maize linkage map (McMullen et al., 2009), assigned strictly additive effects for two synthetic traits following a multivariate normal distribution, $MVN\left(\mathbf{0}, \begin{bmatrix} 1 & -0.4 \\ -0.4 & 1 \end{bmatrix}\right)$, and assigned trait intercepts of 10 and 25 for synthetic traits 1 and 2, respectively. Breeding program simulations were performed following a protocol like what is described in (Allier et al., 2019a). We wrote code to perform a burn-in in PyBrOpS. Briefly, 40 unique, randomly selected genotypes were randomly intermated for 10 generations to create a founder population 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 by approximating the Pareto frontier using NSGA-II (Deb et al., 2002), scaling trait value ranges of the identified Pareto optimal set to the range [0,1], and selecting the solution closest to a weight of 0.5 (equal preference) for both scaled traits. The selected solution contained information on which 40 parents were to be selected as 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 point was saved and mean population trait breeding values were recorded. The Pareto frontiers and mean population breeding values throughout time were plotted using Matplotlib (Hunter, 2007) and are visualized in Figures 4A and 4B, respectively.

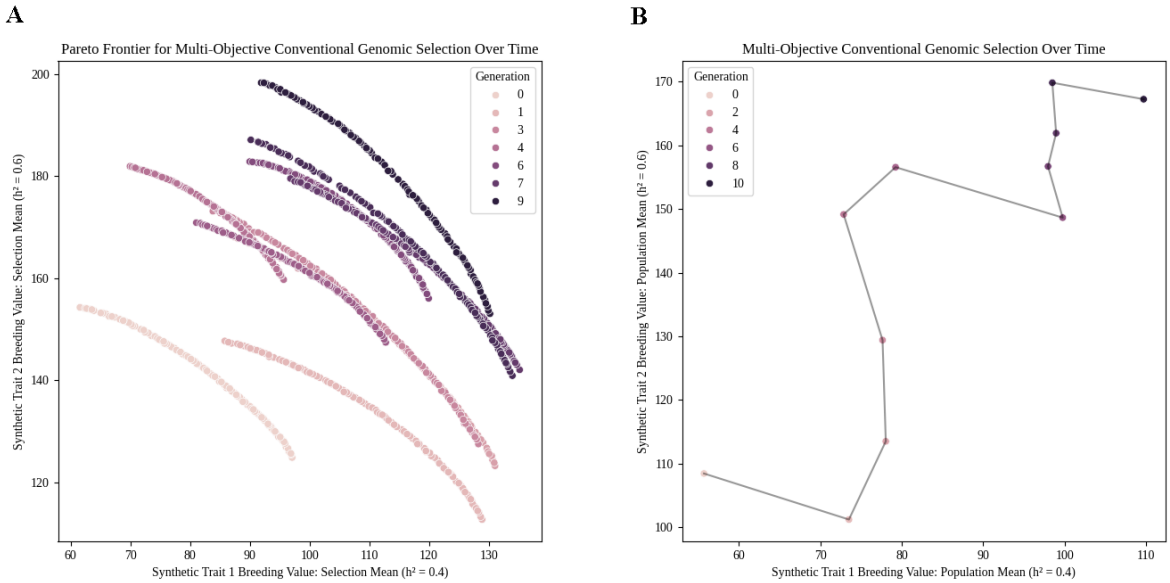

Figure 4: A) Pareto frontiers mapped by NSGA-II and B) population mean breeding values at each generation in Scenario 2 simulations. A) The Pareto frontier at Generation 0 represents the initial population selection possibilities, while the frontier at Generation 9 represents the selection possibilities before the final mating round in the simulation. Individual points along each Pareto frontier represent potential subsets of individuals a breeder could select. B) Mean trait breeding values for the breeding population in Scenario 2 simulations.

4. 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.

5. Web resources

PyBrOpS is open source and can be found on GitHub at https://github.com/rzshrote/pybrops/. Complete documentation, examples from this paper, and examples for other multi-objective optimization and simulation scenarios can be found on the associated GitHub Pages website at https://rzshrote.github.io/pybrops/. Feature requests, bug reports, and feature submissions may be made at the aforementioned GitHub repository link.

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

Akdemir D, Sánchez JI (2016) Efficient Breeding by Genomic Mating. Front Genet **7**: 210

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

Amini F, Hu G, Wang L, Wu R (2022) The L-shaped selection algorithm for multitrait genomic selection. Genetics **221**: iyac069

Bernardo R (2020) Breeding for Quantitative Traits in Plants. Stemma Press, Woodbury, Minnesota

De Beukelaer H, Badke Y, Fack V, De Meyer G (2017) Moving Beyond Managing Realized Genomic Relationship in Long-Term Genomic Selection. Genetics **206**: 1127 LP – 1138

Beume N, Naujoks B, Emmerich M (2007) SMS-EMOA: Multiobjective selection based on dominated hypervolume. Eur J Oper Res **181**: 1653–1669

Blank J, Deb K (2020) Pymoo: Multi-Objective Optimization in Python. IEEE Access **8**: 89497–89509

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

Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd ed. Springer, New York, NY

Corne DW, Knowles JD, Oates MJ (2000) The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization. *In* M Schoenauer, K Deb, G Rudolph, X Yao, E Lutton, JJ Merelo, H-P Schwefel, eds, Parallel Problem Solving from Nature PPSN VI. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 839–848

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

Endelman JB (2011) Ridge Regression and Other Kernels for Genomic Selection with R Package rrBLUP. Plant Genome. doi: https://doi.org/10.3835/plantgenome2011.08.0024

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

Gorjanc G, Gaynor RC, Hickey JM (2018) Optimal cross selection for long-term genetic gain in two-part programs with rapid recurrent genomic selection. Theoretical and Applied Genetics **131**: 1953–1966

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

Hunter JD (2007) Matplotlib: A 2D Graphics Environment. Comput Sci Eng **9**: 90–95

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

Knowles J, Corne D (1999) The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). pp 98-105 Vol. 1

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

Moeinizade S, Kusmec A, Hu G, Wang L, Schnable PS (2020) Multi-trait Genomic Selection Methods for Crop Improvement. Genetics **215**: 931–945

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

Pérez P, de los Campos G (2014) Genome-Wide Regression and Prediction with the BGLR Statistical Package. Genetics **198**: 483 LP – 495

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

Seabold S, Perktold J (2010) Statsmodels: Econometric and Statistical Modeling with Python. *In* S van der Walt, J Millman, eds, Proceedings of the 9th Python in Science Conference. Austin, Texas, pp 92–96

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

Zhang Q, Li H (2007) MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation **11**: 712–731

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength pareto evolutionary algorithm. Zurich, Switzerland

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation **3**: 257–271