Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.elsevier.com/locate/iswa

A reinforcement Learning approach to resource allocation in genomic selection

Saba Moeinizade*, Guiping Hu, Lizhi Wang

Industrial and Manufacturing Systems Engineering Department, Iowa State University, Ames, IA 50011 USA

A R T I C L E I N F O

Article history:
Received 31 July 2021
Revised 25 February 2022
Accepted 3 April 2022
Available online 5 April 2022

Keywords:
Reinforcement learning
Genomic selection
Plant breeding
Resource allocation
Markov decision process
Function approximation

A B S T R A C T

Genomic selection (GS) is a technique that plant breeders use to select individuals to mate and produce new generations of species. Allocation of resources is a key factor in GS. At each selection cycle, breeders are facing the choice of budget allocation to make crosses and produce the next generation of breeding parents. Inspired by recent advances in reinforcement learning for AI problems, we develop a reinforcement learning-based algorithm to automatically learn to allocate limited resources across different generations of breeding. We mathematically formulate the problem in the framework of Markov Decision Process (MDP) by defining state and action spaces. To avoid the explosion of the state space, an integer linear program is proposed that quantifies the trade-off between resources and time. Finally, we propose a value function approximation method to estimate the action-value function and then develop a greedy policy improvement technique to find the optimal resources. We demonstrate the effectiveness of the proposed method in enhancing genetic gain using a case study with realistic data.

1. Introduction

Over the past decades breeding methods have evolved from traditional phenotype-based selection to marker-assisted selection methods. Genomic selection (GS), which was initially proposed by Meuwissen et al. (2001), is a special form of marker assisted selection that estimates the effects of genome-wide markers in a training population consisting of genotyped and phenotyped individuals. Different statistical and machine learning models are proposed to develop prediction models based on the genotypic and phenotypic data of the training population (Chen et al., 2014; Crossa et al., 2017; Dong et al., 2016; Li et al., 2015; Liu et al., 2018; Liu and Wang, 2017; Mahendran et al., 2020; Montesinos-López et al., 2018; Neves et al., 2012; Pryce et al., 2011). Then, the prediction model is used to derive the genomic estimated breeding values (GEBVs) for all individuals of the breeding population (BP) from their genomic profile by calculating the sum of the estimated marker effects. Given the genotype information and the estimated marker effects of individuals in a breeding population, there are different decisions that should be made within each breeding cycle. These decisions include selection, mating, and resource allocation which must be made in every generation with the objective of continuously improving individuals subject to deadline constraints (Moeinizade, 2018; 2021; Moeinizade et al., 2021; 2020a; 2020b).

Recently, Moeinizade et al. (2019) presented the look-ahead selection (LAS) method by applying operations research techniques to optimize selection and mating strategies. A new time-dependent technique was invented to anticipate the consequences of selection and mating decisions through several generations, which was achieved by quantitatively taking into account recombination frequencies. Recombination, the main source of uncertainty in reproductive biology, is the phenomenon that occurs during meiosis and creates different combinations of alleles in the resulting gametes (Lobo and Shaw, 2008). In Moeinizade et al. (2019), we conducted a case study using realistic maize data and compared LAS with other published selection methods. Simulation results suggested the superiority of LAS to other selection methods. However, the LAS method was unable to optimize resource allocation decisions, e.g., how should the budget be distributed over time? Should it be spent evenly or should more investment be made in earlier generations before genetic diversity deteriorates? how many crosses should be made and how many progeny should be produced? These resource allocation decisions should be optimized systematically, given the cost of making a cross and genotyping progeny, under budget and deadline constraints, considering the uncertainty in recombination in each generation.

* Corresponding author.
E-mail address: sabamoeini.sm@gmail.com (S. Moeinizade).

https://doi.org/10.1016/j.iswa.2022.200076

In this study, we develop a reinforcement learning-based algorithm to automatically learn to allocate resources across different generations of breeding. The proposed new method integrates the LAS approach in a reinforcement learning framework. The LAS method is capable of anticipating the consequences of the selection and mating decisions under uncertain recombination events efficiently and accurately, whereas the reinforcement learning framework is capable of making a trade-off between cost and time which is necessary to make resource allocation decisions.

Reinforcement learning (RL) is one of the most important research directions of machine learning, which has been widely used in different fields like social sciences, natural sciences, and engineering and has significantly impacted the development of Artificial Intelligence (AI) over the last years (Dayan and Niv, 2008). Sutton and Barto (2018) define Reinforcement learning as learning what to do -how to map situations to actions- so as to maximize a numerical reward signal. There is a growing body of literature applying reinforcement learning methods to existing operations research (OR) problems (Hubbs et al., 2020; Li, 2017; Mazyavkina et al., 2021). Powell (2007) discusses RL techniques from an operations research perspective and presenets several solution methodologies for solving large-scale problems in supply chain, inventory management, resource management, etc. Bertsekas (2012) discusses large-scale dynamic programming based on approximations which intertwines with the current field known as reinforcement learning. Gambella et al. (2021) surveys the underlying optimization problems in machine learning by presenting mathematical models for regression, classification and deep learning. Furthermore, reinforcement learning methods have been used to solve combinatorial optimization problems such as Traveling Salesman Problem (TSP), KnapSack, and Vehicle Routing Problem (VRP), to name a few (Bello et al., 2016; Khalil et al., 2017; Kool et al., 2018; Mazyavkina et al., 2021; Nowak et al., 2018).

The main characters of RL are the agent and the environment. The environment represents the outside world to the agent and the agent interacts with the environment by taking actions and receiving a reward signal. The goal of the agent is to maximize the cumulative reward, named return. To do that, the agent should learn the optimal policy which is an optimal strategy to behave in the environment. RL problems can be formulated mathematically in the framework of Markovian Decision Processes (MDPs) by defining states, actions, transition probabilities, and rewards (Szepesvári, 2010). The transition and reward functions in MDPs are called the model of environment. A known MDP can be solved by dynamic programming which relies on simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner (Bellman, 1966). However, we often do not have the transition and the rewards of the MDP. This class of problems with unknown MDPs are called model-free. While model-based methods rely on planning as their primary component, model-free methods rely on learning (Sutton and Barto, 2018). Model-free methods can be applied to both prediction and control problems. In model-free prediction, the goal is to estimate the value function of an unknown MDP where as model-free control aims at optimizing the value function. The value function represents how good it is for an agent to be in a given state.

In recent years, different solution methods have been proposed to solve model-free RL problems (Hausknecht and Stone, 2015; Heess et al., 2017; Mnih et al., 2013; Schulman et al., 2015a; 2015b; 2017; Tucker et al., 2018; Van Hasselt et al., 2015; Wang et al., 2016). These solution methods include two main types of algorithms, value-based and policy-based. Value-based algorithms iteratively update the value of a state to finally learn an optimal policy. Policy-based algorithms learn a parameterized policy that can select actions without consulting a value function.

Q-learning, a value-based RL algorithm, is one of the most popular solution methods in reinforcement learning. This algorithm uses Q-values (an estimation of how good it is to take an action at a given state) to iteratively improve the behaviour of the learning agent (Watkins and Dayan, 1992). However, for large-scale problems with an enormous number of state-action pairs, it is difficult to explicitly store all the Q-values. To overcome this challenge, function approximation methods are used where value function is represented by mapping a state description to a value (Arulkumaran et al., 2017; Gosavi, 2009; Kaelbling et al., 1996). Many implementations of RL in real-world problems have used neural networks as function approximators (Hausknecht and Stone, 2015; Mnih et al., 2013; Van Hasselt et al., 2015; Wang et al., 2016). One of the examples is the achievement of AlphaGo in 2016, where a deep Q-network was implemented and trained to predict total reward (Silver et al., 2016). Other approximation methods including kernel methods, nearest-neighbor algorithms, and decision trees can be used to estimate the Q-values (Chapman and Kaelbling, 1991; Friedman et al., 2001; Howe and Pyeatt, 1998). Policy gradient algorithms learn in a more robust way by approximating policy and updating it according to the gradient of expected reward with respect to the policy parameters (Sutton et al., 1999) without the need to construct a value function.

In this study, we mathematically formulate the resource allocation in genomic selection problem in the framework of MDP by defining state and action spaces and then propose a value-based algorithm with function approximation and introduce a backward greedy policy approach with respect to the estimated values (i.e., the policy that selects the action with highest estimated value in each state). The idea of the backward approach is to learn the optimal action in a backward manner starting from the final generation to the first generation given that the optimal strategy in the final generation is allocating all remaining resources. One of the major contributions of this study is the definition of state space to avoid formidable dimensions given the genotypic information. We tackled this challenge by proposing an integer linear program which captures the important information of the population by quantifying the trade-off between resources and time.

In the remainder of this paper, we formulate the resource allocation problem in an RL framework, discuss the solution methods and finally present a case study to compare our proposed allocation strategy with even allocation using computer simulation.

2. Methods

In this section, we first define the genomic selection resource allocation problem and then formulate the proposed problem mathematically in the context of Markov Decision Process (MDP), where reinforcement learning algorithms can be used. Finally, we provide a solution method to solve the proposed MDP and find the optimal policy.

2.1. Problem definition

A classical plant breeding process starts with an initial population and iteratively goes through the selection and reproduction steps until getting the final population. In addition to the selection decisions, in each generation, the breeder should decide how to allocate resources (i.e., the number of crosses to be made and the number of progeny to be produced from each cross). The focus of this study is optimizing the resource allocation strategy in a breeding program.

Let $G_t \in \mathbb{B}^{L\times M\times N}$ represent the genotype of the population at generation t, where L is the total number of alleles, M indicates the ploidy of the plant ($M = 2$ for diploid species) and N is the total number of individuals in the population. For all l, let β_l denote the

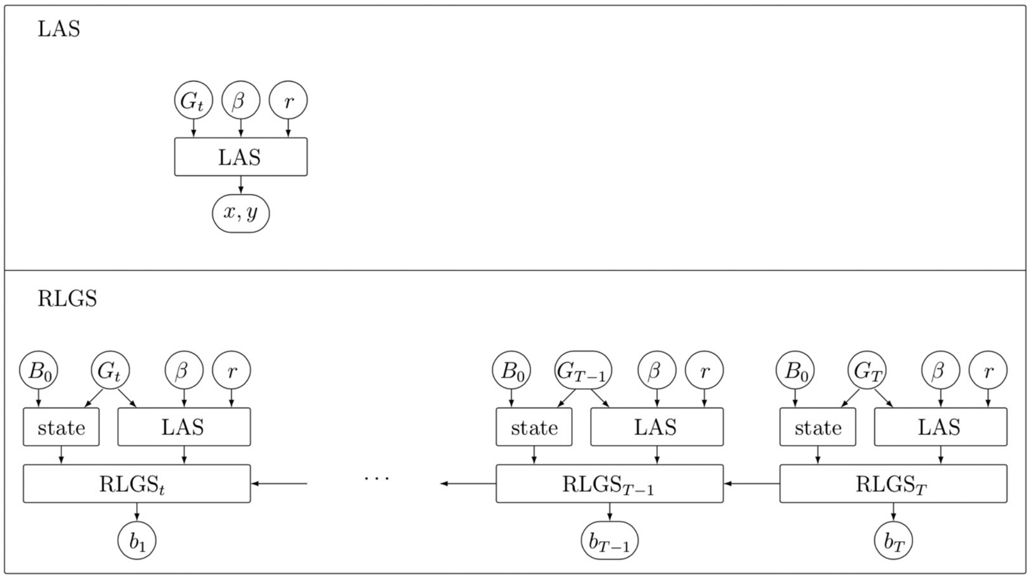


Fig. 1. The architectural diagram for look-ahead selection (LAS) and the proposed reinforcement learning for genomic selection (RLGS) algorithms.

additive effect of allele l, which is assumed to have been reasonably estimated. Given β and G, the look-ahead selection (LAS) algorithm can optimize the selection and mating steps with a time-dependent approach (Moeinizade et al., 2019) by maximizing the expected GEBV of the best offspring in the terminal generation (T) where GEBV of an individual can be calculated as the sum of all marker effects across the entire genome.

Let the cost of producing one progeny be one unit of budget. Then, spending b units of resources in the current generation will produce b progeny. Given a fixed amount of total budget of B_0 units of resources over T generations, the goal is to find the optimal budget or population size for each generation, $(b_1, b_2, \dots, b_T)$, in order to maximize the performance of individuals in the final generation. Similar with selection and mating, resource allocation decisions should be made in a dynamic manner after observing the genotype of progeny from previous generations, while considering the total budget constraint over T generations: $\sum_{t=1}^{T} b_t \le B_0$.

Fig. 1 demonstrates the architecture of the proposed reinforcement learning algorithm for genomic selection (RLGS) and highlights the differences between RLGS and the previously proposed look-ahead selection (LAS) algorithm. The inputs to the LAS algorithm are the genotype of the population at generation t (G_t), allele effects (β), and recombination frequency (r). The outputs are the selection (x) and mating (y) decisions. The RLGS algorithm uses LAS to optimize the selection and mating decisions and focuses on optimizing the resource allocation decisions. As presented in Fig. 1, the RLGS algorithm takes B_0, G_t, β and r as inputs and outputs the optimal resources for each generation in a backward manner.

2.2. Problem formulation

Here, we present the MDP formulation for the genomic selection resource allocation problem. An MDP process is described by a finite set of states (S), a finite set of actions (A), transition probabilities (T), and a reward function (R). Due to the stochastic nature of the environment, in this problem, we cannot derive the transition probabilities and the reward is delayed until the terminal generation. Hence, we use learning to understand the behavior of the environment by simulating different scenarios of resource allocation (Section 2.3). In this section, we define the state and action spaces for the MDP.

We mathematically formulate the problem in the framework of Markov Decision Process (MDP) by defining state and action spaces. To avoid the explosion of the state space, an integer linear program is proposed that quantifies the trade-off between resources and time. Finally, we propose a value function approximation method to estimate the action-value function and then develop a greedy policy improvement technique to find the optimal resources.

2.2.1. State space

To capture the full information in each generation, the population genotype would be necessary to define the state space, but it would make the resulting model unsolvable. For example, for a small population of 200 individuals and only 10,000 pairs of genes, the dimension of the state space would be $3^{2,000,000}$ with each pair of genes taking three possible combinations of two variants of alleles (AA, aa, or Aa). To avoid formidable dimensions, we need to simplify the state space by presenting a compact definition that captures the important information by considering the current genetic value of the population and quantifying the trade-off between time and resources.

At generation t, we define the state by $\left(g_t^{\max}, C_t, B_{t-1}\right)$, where $g_t^{\max}$ is the highest GEBV of the N individuals at generation t calculated as follows:

$$g_t^{\max} = \max_{n\in\{1,2,\dots,N\}}\left(\sum_{l=1}^{L}\sum_{m=1}^{2} G_t^{l,m,n}\beta^l\right) \tag{1}$$

In this state space definition, $C_t \in \mathbb{R}^{K\times M}$ measures the specific combining ability, and B_{t-1} is the available budget to be spent in generations t to the end. Specifically, $C_t^{k,m}$ is the highest possible GEBV of a gamete that could be assembled from G_t using at most m individuals with recombination events that are more likely than p^k, where $p \in (0, 1)$ is an adjustable parameter, depending on the sensitivity of recombination frequency and available resources. The value $C_t^{k,m}$ measures the potential of the genotype G_t to create a gamete with the highest possible GEBV subject to resource and time constraints. The first dimension k reflects the constraint of probabilistic recombinations afforded by remaining resources, and the second dimension m indicates the number of founding parents that the gamete needs to collect alleles from, which would require $\lfloor \log_2 m \rfloor$ generations of breeding. Value $C_t^{k,m}$ can be obtained using

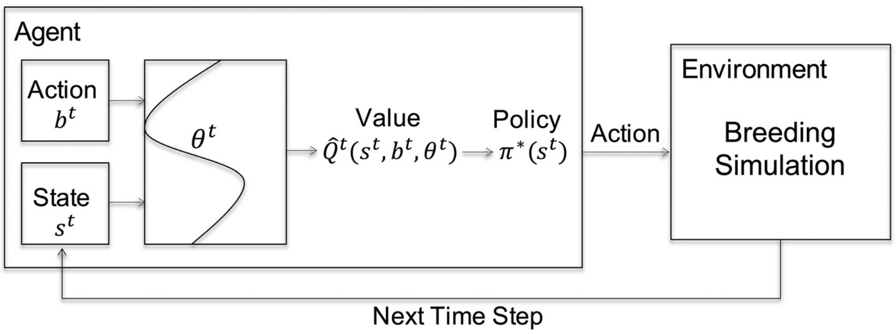


Fig. 2. The reinforcement learning system representation. The environment is the breeding simulation which provides the next state to the agent. At each time step, the value of the current state is evaluated using a pretrained nonlinear function for a given action. Finally, the policy function determines the optimal action which is passed to the simulation.

the following integer linear program.

$$\max_{x,y,z} C_t^{k,m} = \sum_i \sum_c \sum_j \beta_i G_t^{i,c,j} x_{i,c,j} \tag{2}$$

$$\text{s.t.} \sum_j \left(x_{i,1,j} + x_{i,2,j}\right) = 1 \ \forall i \tag{3}$$

$$\sum_i \left(x_{i,1,j} + x_{i,2,j}\right) \le L y_j \ \forall j \tag{4}$$

$$x_{i,c,j} - x_{i+1,c,j} \le z_i \ \forall i, c, j \tag{5}$$

$$\sum_j y_j \le m \tag{6}$$

$$\prod_i \left(\frac{r_i}{1-r_i}\right)^{z_i} \ge p^k \tag{7}$$

$$x, y, z \text{ binary} \tag{8}$$

Here, $x_{i,c,j}$ is a binary variable that indicates whether allele (i, c, j) is selected $\left(x_{i,c,j} = 1\right)$ or not $\left(x_{i,c,j} = 0\right)$ to assemble the gamete, y_j is a binary variable that indicates whether individual j is used $\left(y_j = 1\right)$ or not $\left(y_j = 0\right)$, and z_i is a binary variable that indicates whether there is a recombination between loci i and $i+1$ $(z_i = 1)$ or not $(z_i = 0)$.

The objective value (2) is the maximum possible GEBV of a gamete that can be assembled from the current population. Constraint (3) ensures selection of one chromosome for each locus in an individual to assemble the gamete. Constraint (4) requires that no alleles from unselected individuals can be used to assemble the gamete. Constraint (5) detects whether a recombination is necessary between loci i and $i+1$. Constraint (6) limits the selection of at most m parents among all individuals. Finally, constraint (7) requires that the likelihood of necessary recombinations be no less than p^k, which equivalently limits the amount of resources needed to afford such recombination events. Take for example, for very small r_i values, the value of $\left(\frac{r_i}{1-r_i}\right)$ becomes very small, hence we need more resources (larger k) to make that recombination happen.

2.2.2. Action space

The decision maker should take an action in each generation and decide the amount of resources and the selection strategy for that generation. In Moeinizade et al. (2019), we demonstrated the effectiveness of look-ahead selection (LAS) against conventional selection methods. Here, we focus on optimizing the resource allocation and use LAS algorithm to determine the selection strategy.

A common way of obtaining approximate solutions for continuous action spaces is to discretize the action space. In a discrete action space, the agent decides which distinct actions to perform from a finite action set. In this study, we discretize the action space. Specifically, we assume allocating b amount of resources in one generation is equal to producing b total number of progeny in that generation (making one progeny costs one unit of budget). Hence, action is a discrete value representing the number of progeny in the population.

We define the action space as time dependent set of $(b_1, b_2, \ldots, b_T)$ values such that $\sum_{t=1}^{T} b_t = B_0$ where b_t is the amount of resources to spend in generation t, T is the total number of generations, and B_0 is the amount of total budget.

2.3. Proposed solution technique

Fig. 2 represents the reinforcement learning system. As shown in this figure, there are two main components in the system: agent and environment. The goal of the agent is to find the optimal policy (i.e., the optimal action to take at time t) where policy is defined as a function mapping states to the actions ($\pi : S \to A$). The environment is the breeding simulation which provides the next state to the agent. The agent evaluates the value of performing action b in state s using a pretrained value function and then acts greedily to find the best policy. The agent then decides to take the optimal action at time $t+1$ and this process continuous until reaching the deadline.

Suppose we have an MDP defined as state-action pairs and given some policy π. First, we predict the value function by constructing the action-value function $\hat{Q}(s, b, \theta)$ to represent the objective value for a given state-action pair. Then, we can predict the value of a state given all possible actions and find the optimal policy, $\pi^*(s)$, that maximizes the action-value function. Section 2.3.1 describes the value function approximation technique and Section 2.3.2 elaborates on policy improvement.

2.3.1. Value function approximation

The value function demonstrates how good each state and/or action is by calculating the expected cumulative reward in long-term. In this problem, the immediate rewards are considered to be zero and the objective is to maximize the genetic gain in final generation. Hence, the value here represents the GEBV of the best offspring in the final generation, where GEBV of individual n is calculated as the sum of effects across the entire genome $(\text{GEBV}(n) = \sum_{l=1}^{L} \sum_{m=1}^{2} G^{l,m,n} \beta^l)$.

The simplest way of representing a value function is by the use of a lookup table, with the values of each state-action pair stored. However, when the state-action spaces are large, storing and retrieving values become a problem, as it takes up large amounts of computational resources. To solve this problem, function approximators can also be used instead of a lookup table for representing value functions, thereby limiting the memory being used and speeding up the learning process. Therefore, to estimate the value function, $\hat{Q}(s, b, \theta)$, efficiently, we should use a function approximation method (e.g., nonlinear regression, support vector machine, decision tree based models and neural network).

The parameters (θ) need to be learned for each time period, t, separately. We employ a backward approach by optimizing resources from the final generation to the first generation. Given that the objective is to maximize the maximum GEBV in the target generation, ($g_T^{\max}$), the optimal strategy in the final generation is to allocate all the remaining budget ($b_T^* = B_{T-1}$, where B_{T-1} is the remaining resources for the final generation).

To find optimal budget, b_t^*, for earlier generations $t \in \{1, \ldots, T-1\}$, we take advantage of simulation to learn how different budget allocation scenarios impact the final performance by generating learning data as described in Algorithm 1. This algorithm

Algorithm 1 Learning data generation.

```
Start with initial population $G_0$ and total budget $B_0$
for $t := 1$ to $\tau - 1$ do
    $b_t = \texttt{Action}(t, T, B_{t-1}, A)$
    $[S] = \texttt{Select}(G_{t-1}, r, n, b_t)$
    $[G_t] = \texttt{Reproduce}(G_{t-1}, S, r)$
end for
for $b_\tau \in A$ do
    for $t := \tau$ to $T$ do
        if $t = \tau$ then
            Record $(g_\tau^{\max}, C_\tau, B_\tau)$ and $b_\tau$
        else
            if $t = T$ then
                $b_t = B_{T-1}$
            else
                $b_t = \texttt{argmax}_{b \in A} \hat{Q}_t(s, b, \theta)$
            end if
        end if
        $[S] = \texttt{Select}(G_{t-1}, r, n, b_t)$
        $[G_t] = \texttt{Reproduce}(G_{t-1}, S, r)$
        Record $g_T^{\max}$
    end for
end for
```

presents data collection process for a given generation, τ, which goes backwards from $T-1$ to 1. For generation τ, we record state-action pairs, $(g_\tau^{\max}, C_\tau, B_\tau)$ and b_τ, and the objective value, $g_T^{\max}$. Then, we estimate the value function to map state-action pairs to the objective value. We first define the three functions used in Algorithm 1 for data generation and then discuss the value function approximation technique.

Definition 2.1. Selection function is defined as: $[S] = \texttt{Select}(G_{t-1}, r, n, b_t)$. The input parameters are the population genotype at generation $t-1$, $G_{t-1} \in \mathbb{B}^{L \times 2 \times N}$, recombination frequency vector, $r \in [0, 0.5]^{N-1}$, the number of crosses n, and amount of resources for generation t, b_t. Note that the resources correspond to the progeny population size. The output $S = \begin{bmatrix} s_{1,1} & s_{1,2} & b_t^1 \\ s_{2,1} & s_{2,2} & b_t^2 \\ & \cdots & \\ s_{n,1} & s_{n,2} & b_t^n \end{bmatrix}$ contains the indices of selected parents in the breeding population and the number of progeny produced from each cross (here, row $\begin{bmatrix} s_{i,1} & s_{i,2} & b_t^i \end{bmatrix}$ means that the i^{th} cross in generation t is made using parents $s_{i,1}$ and $s_{i,2}$, which produces b_t^i progeny. The numbers of progeny satisfy the budget constraint that $\sum_{i=1}^n b_t^i = b_t$).

Definition 2.2. The reproduction function is defined as follows: $[G_t] = \texttt{Reproduce}(G_{t-1}, S, r)$. The input parameters are the population genotype at generation $t-1$, $G_{t-1} \in \mathbb{B}^{L \times 2 \times N}$, selection matrix, S, and the recombination frequency vector, $r \in [0, 0.5]^{N-1}$. The output is the genotype of the progeny population. The genetic information are inherited from parents to progeny according to the inheritance distribution defined in Han et al. (2017).

Definition 2.3. Action function is defined as: $[b_t] = \texttt{Action}(t, T, B_{t-1}, A)$. The input parameters are the current generation, t, total number of generations, T, the available resources to be spent in generations t to the end, B_{t-1}, and possible set of actions, A. The output is resources or progeny size for generation t. We choose an action randomly from a finite set of values $\tilde{a} \in A,\ \ A = \{a_1, a_2, \ldots, a_k\}$. We produce at least α progeny for each generation ($\alpha = \min_{i=1}^k a_i$). Therefore the output, b_t, can be calculated as follows: $b_t = \min(\tilde{a}, B_{t-1} - \alpha \times (T-t))$.

Generating learning/training data in complex stochastic environments can be time consuming. Neural networks usually need more training data and thus are not the best approach here since the learning process is considerably time-consuming. After exploring three function approximators including generalized additive model (GAM), support vector machine (SVM), and random forest (RF), we decided to choose random forest considering both efficiency and computational time. The inputs to the random forest model are the data generated using Algorithm 1 with $km+3$ features including the maximum current GEBV ($g_t^{\max}$), the potential matrix (C_t), remaining budget (B_{t-1}), and action (b_t) where k and m are the dimensions of the potential matrix. The output is the GEBV of best individual in the final generation, $g_T^{\max}$. More detailed information is provided in the Appendix section.

2.3.2. Greedy policy improvement

The ultimate goal of the agent is to find an optimal policy π^* that maximizes the value function. After learning the value function, we employ a greedy approach to improve the policy by selecting the action with the highest estimated value in each state. Let $\hat{Q}_t(s, b, \theta)$ represent the approximated value function for each generation except final. We can calculate the optimal policy for all generations from 1 to $T-1$ as follows:

$$\pi_t^*(s) = \text{argmax}_{b \in A} \hat{Q}_t(s, b, \theta), \ \ \forall t \in \{1, 2, \ldots, T-1\} \tag{9}$$

Moreover, the optimal policy in the final generation is to allocate all the remaining budget. Therefore $\pi_T^*(s) = B_{T-1}$, where B_{T-1} presents the remaining resources for the final generation.

3. Results

3.1. Simulation settings

The genotypic data, marker effects and recombination rates are based on Moeinizade et al. (2019). The genotypic data contains genotypes of 369 maize inbred lines consisting of $L = 1.4$M SNPs distributed across ten maize chromosomes. To reduce the dimension, we define haplotype blocks. The resulted data has $L = 10,000$ markers.

Let's assume there exist 5 total generations of breeding ($T = 5$) and the amount of total budget is 1000. We consider seven possible action values as follows: $A = \{50, 100, 150, 200, 250, 300, 350\}$. Note that the amount of budget in each generation indicates the total number of progeny produced in that generation. Additionally, we consider that no more than 10 crosses are made in each generation.

Fig. 3 demonstrates the simulation flowchart. We start with the initial population by randomly choosing 200 individuals out of 369. The state is observed by calculating the tuple $\left(g_t^{\max}, C_t, B_{t-1}\right)$. The C_t matrix is generated by solving the optimization problem presented in (2)-(8). Here we choose $k = 5$, and $m = 5$. These are parameters and can be changed according to the data and time required for solving the optimization problem. Fig. 4

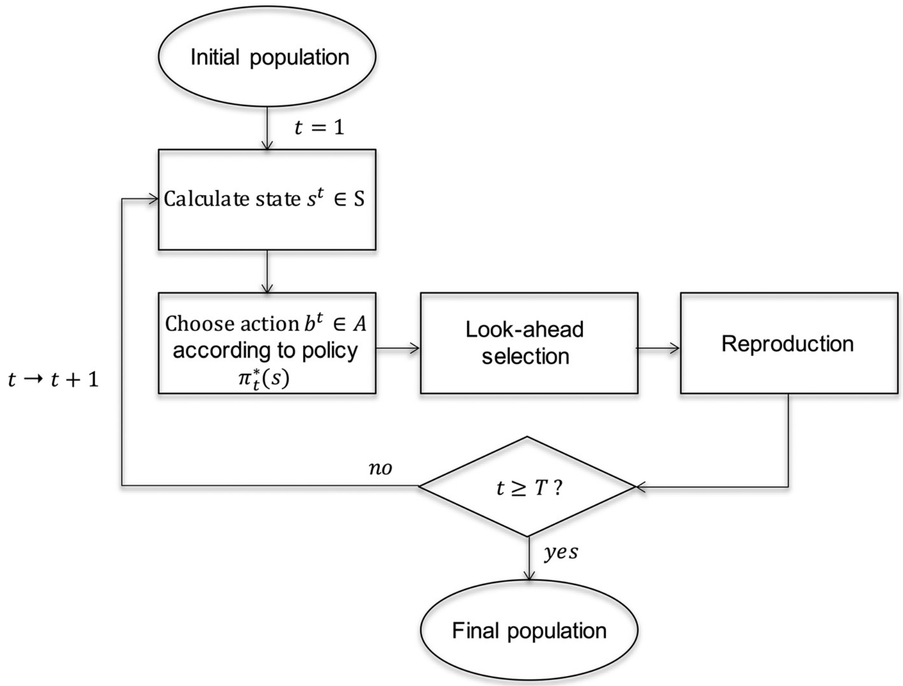


Fig. 3. The simulation flowchart. The process starts with the initial population and goes through resource allocation, selection and reproduction steps until getting to the deadline. To find the optimal resources, we first calculate the current state and then take the action with highest value according to the optimal policy. The action represents the number of progeny to be produced for that generation.

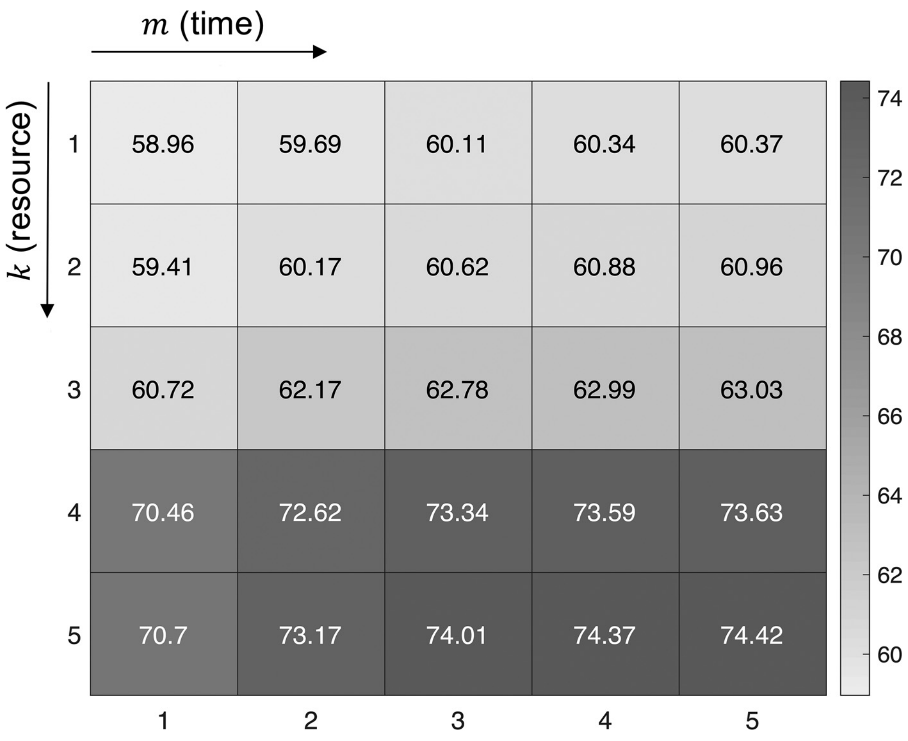
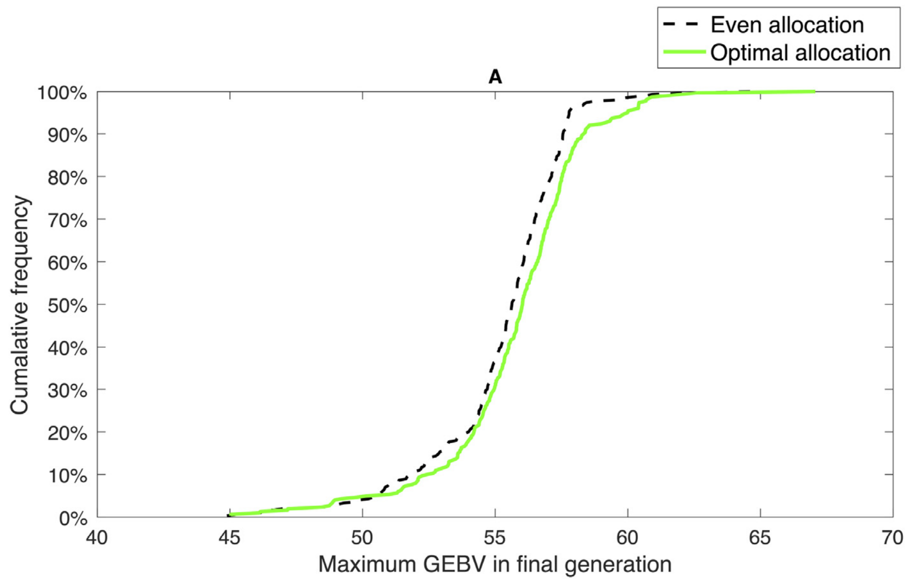


Fig. 4. Heat map for one sample C matrix where $k = 5$, and $m = 5$. Each square demonstrates the best achieved GEBV value considering different levels of time and resources. The bottom right square has the highest potential GEBV value since it considers having the most time and resources.

shows the heat map for one sample C_t in a simulation. The x-dimension of this plot is representing the possibility of combining more alleles from multiple individuals in case of having more time. Moreover, the y-dimension is representing the possibility of allowing more recombination to happen in case of having more resources. As expected, the performance becomes better towards the right and bottom of the plot by considering more time and resources. In addition, this C matrix indicates that given the current generation, potential genetic gain is more sensitive to resources than to time constraint, which is helpful for the reinforcement learning algorithm to make resource allocation decisions.

Next, the optimal policy will be calculated using the current generation action-value function in a greedy approach. Then, can-



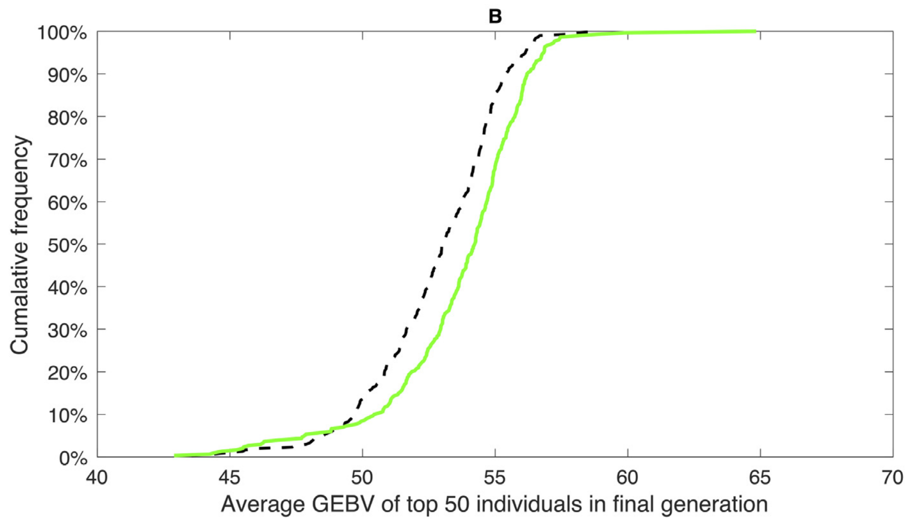


Fig. 5. Cumulative distribution functions of population maximum in the final generation (A) and average performance among top 50 individuals in the final generation (B) for two strategies of resource allocations among 300 independent simulations. The black dashed curve represents the even allocation strategy and the green curve represents the optimal allocation strategy. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

didates are selected according to look-ahead selection as parents to produce next generation. This continuous till reaching the deadline. Finally, we evaluate the performance based on the GEBV of individuals in the final population.

3.2. Simulation results

To approximate the action-value function, we first generated learning data including state-action pairs using simulation and then trained a random forest algorithm for each generation separately to estimate the objective value. The size of training observations that were generated in the simulation vary between 1500 to 6000 for each generation, and there were a total 28 predictors including the action (b), maximum current GEBV ($g^{\max}$), remaining budget (B) and 25 values from the potential matrix, C. For training the random forest model, we did a search grid over three parameters including the number of selected features, minimum leaf node size, and maximum number of splits. The set of parameters with the least out of bag error were selected. The out of bag mean square errors for the first generation until the fourth generation are 2.39, 2.41, 2.31, 2.25, respectively.

We compared the optimal resource allocation strategy with the even allocation strategy (i.e., allocating resources equally across all generations). Three hundred independent simulations were conducted for each strategy using MATLAB (R2021-a).

Fig. 5 (A) demonstrates the cumulative distribution functions (CDFs) of the population maximum in the final generation. The performance becomes better as the CDF moves towards the right direction. Take for example, point (60, 92) means 92% of the simulations achieved maximum GEBV less than or equal to 60. As demonstrated in this figure, the optimal allocation strategy outperforms even allocation strategy in almost all percentiles. Although, this improvement is not by a high margin, but it is considerable given that the improvement is across almost all percentiles for 5 generations of breeding. More improvements can be achieved for longer-term breeding. Moreover, if we compare the average performance of top 50 individuals instead of top 1, we can see a wider gap between the two curves as shown in Fig. 5 (B).

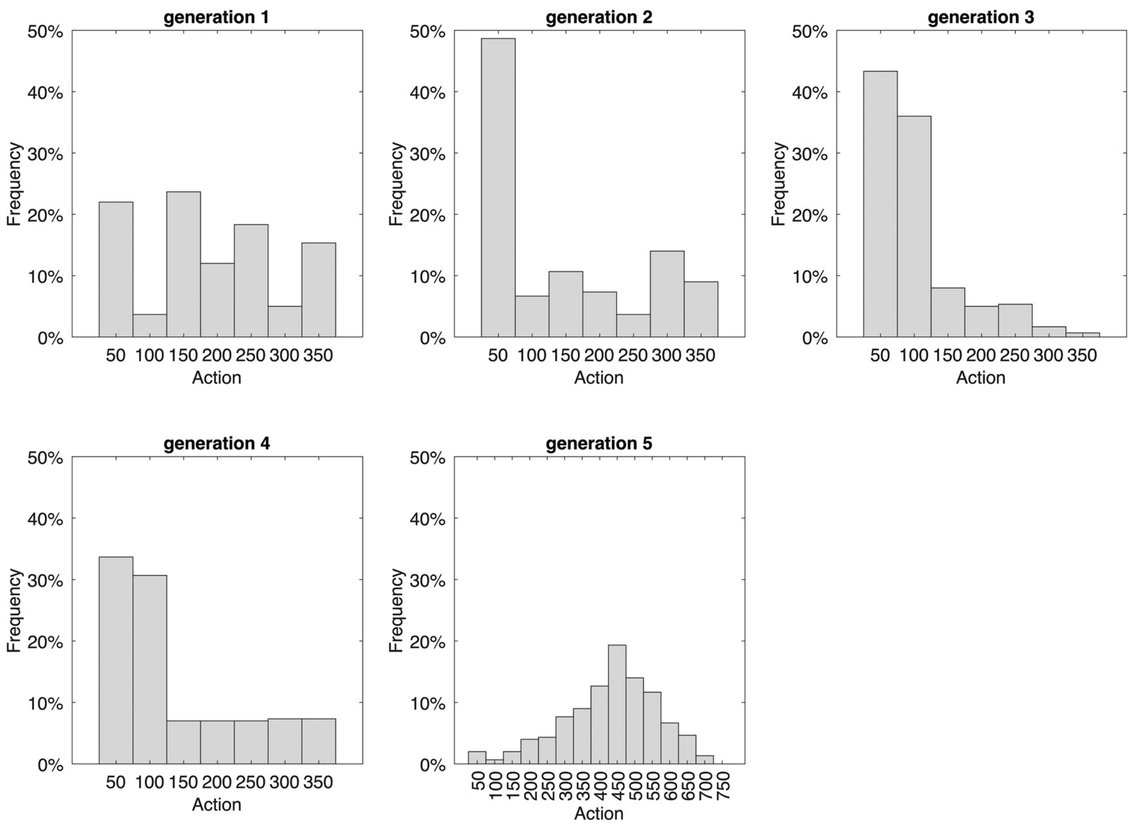


Fig. 6. Histograms of resource allocation across 5 generations for the optimal resource allocation strategy. The amount of resources that can be spent in all generations till one before final is chosen from a predefined set of actions. Here, we have seven different possibilities (50, 100, 150, 200, 250, 300, 350) for generations one till four and the remaining budget will be spent in the final generation. Note that for the even allocation strategy the action is deterministic which is $b = 200$ for all generations.

So far, we have observed the improvements of our proposed optimal allocation strategy with respect to the even allocation strategy. Thus, the question arises: what is the behavior of the optimal allocation strategy and why that behavior results in improvements? To understand this better, we examined the histograms of resource allocation among 5 generations for the optimal strategy. As illustrated in Fig. 6, three different behaviors are observed. In the first generation, the optimal strategy tends to be around the average, in the middle generations, there is more tendency towards spending less resources, and in the final generation almost half of the total budget is spent.

In this case study, we demonstrated the effectiveness of our proposed method against evenly allocating resources across breeding cycles. Our optimal strategy suggests investing in the first generation and then spending moderate amount of resources in the middle generations and finally investing more in the final generation to exploit the best performance that can be achieved.

4. Conclusion

This study provides a framework to find the optimal resources that should be allocated throughout different generations in a breeding program by integrating the recently proposed look-ahead selection algorithm for genomic selection and reinforcement learning techniques. Look-ahead selection is capable of estimating the consequences of selection and mating decisions under uncertain recombination events. Reinforcement learning is able to balance the trade-off between cost and time but its performance is sensitive to the definitions and dimensions of the state and action spaces. Therefore, look-ahead selection is integrated into the reinforcement learning framework to optimize resources in addition to the selection and mating steps and new solution techniques are proposed to battle the curse of dimensionality.

We considered MDPs with very large and continuous state spaces, and we used random forest to construct an approximate function to store the value functions used by the algorithms. We implemented a greedy policy improvement to learn optimal policy in a backward manner. We benefit from the structure of the genomic selection problem and assume the best policy in the target generation is known (which is spending all the remaining budget). Then, we approximate the value function from the last generation to the first one and use it to improve the policy in a greedy way. Numerical results suggested the improvement of the proposed optimal allocation strategy versus even allocation strategy.

The RL framework presented in this work has two major contributions. The first contribution is the definition of the state space. It is analytically and computationally challenging to simplify the state space definition for a large scale stochastic environment. To avoid the explosion of state space, we propose an integer linear program that captures the genomic information of the population by considering the trade-offs between time and resources. The second one is integrating the look-ahead selection and reinforcement learning. Given the optimal allocation strategy, look-ahead selection further improves the genetic gain by optimizing the selection and mating steps.

Future research is needed to address the limitations of this study. First, the current paper considers a sparsely discrete action space with predefined values. Future research should consider a more complete action space and investigate algorithms to optimize policy in such space. Second, here we assume the budget is proportional to the total number of progenies. In certain situations, however, it is also possible that genotyping or crossing must be performed in batches that contain up to a certain number of plants, then the cost is not proportional to the number of individuals, but the number of batches, making the cost function a step function; this would certainly cause additional complexity to the model. Future research can investigate extending this work to those cases where cost is not proportional to the number of individuals. Furthermore, deep neural networks can be used for function approximation if we generate more learning data by making the simulation more efficient. Finally, the case study presented here is for a single data set from a single crop organism. Future research considering more species is necessary to demonstrate the generalization of our proposed method.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Saba Moeinizade: Methodology, Formal analysis, Writing – original draft, Visualization. **Guiping Hu:** Supervision, Investigation, Writing – review & editing. **Lizhi Wang:** Conceptualization, Methodology, Writing – review & editing.

Acknowledgments

This work was partially supported by the Agriculture and Food Research Initiative Grant no.2017-67007-26175/Accession No. 1011702 from the USDA National Institute of Food and Agriculture. Additionally, this work was partially supported by the National Science Foundation under the LEAP HI and GOALI programs (Grant number 1830478). This work was also supported by the Plant Sciences Institute's Faculty Scholars program at Iowa State University.

Appendix

In this section, we provide detailed descriptions about the function approximation method used in this study. The objective is to estimate the GEBV of best individual in the final generation, $g_T^{\max}$ given maximum current GEBV ($g_t^{\max}$), the potential matrix (C_t), remaining budget (B_{t-1}), and action (b_t). This function should be updated during the backward learning process as more data is generated and added to the training information. For example, we first generate learning data for generation $T-1$ (generation 4 where $T=5$) and learn function $\hat{f}$ to map the state-action pairs to $g_T^{\max}$. Then learning data is generated for generation 3 and therefore function $\hat{f}$ should be updated to include training data from both generations 3 and 4. This process continues until getting to the first generation.

Fig. 7 presents the predictions versus true values of $g_t^{\max}$ for generation 4 given three different function approximators including generalized additive model (GAM), support vector machine (SVM) and random forest (RF). The root mean square error (RMSE) metric is used to find the best approximator. As shown in this figure, random forest has achieved the least RMSE, outperforming the other two methods.

Table 1 presents the RMSE values for generations 4 to 1 given the three function approximation methods. As expected, moving from generation 4 to 1 results in a decrease in RMSE since the model has more learning data. Overall, the random forest method achieved better RMSE values and therefore was selected as the function approximation method in this study.

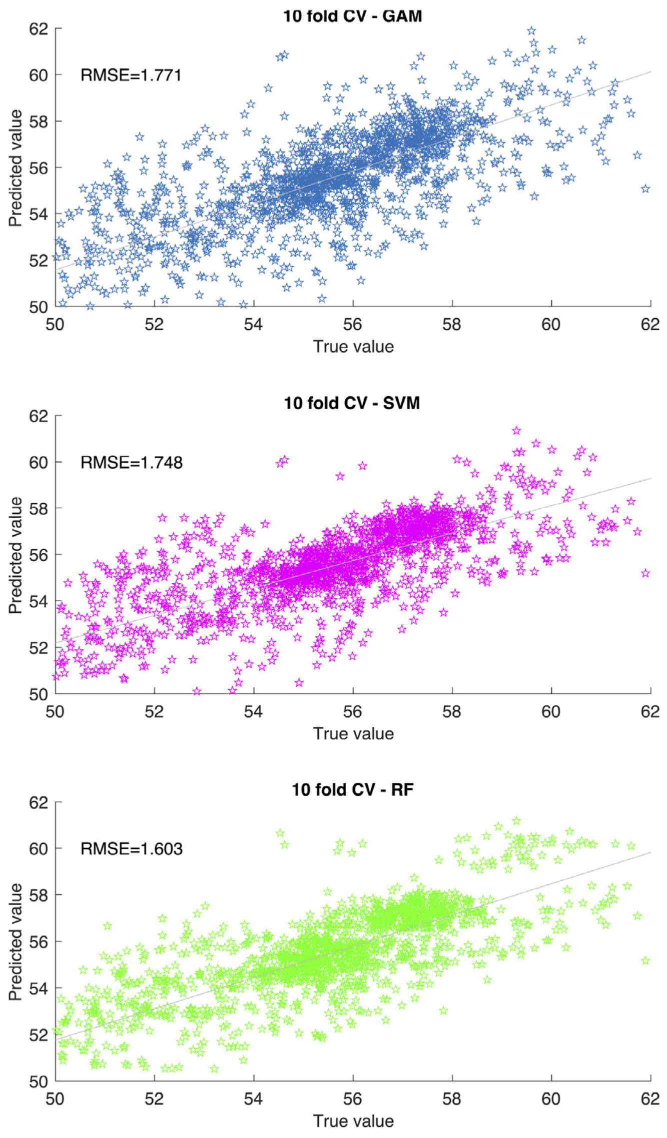


Fig. 7. True value versus predicted values for three function approximation methods including generalized additive model (GAM), support vector machine (SVM) and random forest (RF). Results are for 10 fold cross validation (CV). The root mean square error (RMSE) is reported for each case.

Table 1
Root mean squre error (RMSE) values for three function approximation methods across all generations during the backward learning process.

Generation	GAM	SVM	RF
4	1.771	1.748	1.603
3	1.705	1.675	1.548
2	1.641	1.609	1.516
1	1.595	1.591	1.505

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38.
Bellman, R. (1966). Dynamic programming. *Science, 153*(3731), 34–37.
Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.
Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*: vol. 1. Athena scientific.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Ijcai: vol. 91* (pp. 726–731). Citeseer.

Chen, L., Li, C., Sargolzaei, M., & Schenkel, F. (2014). Impact of genotype imputation on the performance of gblup and bayesian methods for genomic prediction. *PLoS One, 9*(7), e101544.

Crossa, J., Pérez-Rodríguez, P., Cuevas, J., Montesinos-López, O., Jarquín, D., de los Campos, G., Burgueño, J., González-Camacho, J. M., Pérez-Elizalde, S., Beyene, Y., et al. (2017). Genomic selection in plant breeding: Methods, models, and perspectives. *Trends in Plant Science, 22*(11), 961–975.

Dayan, P., & Niv, Y. (2008). Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neurobiology, 18*(2), 185–196.

Dong, L., Xiao, S., Wang, Q., & Wang, Z. (2016). Comparative analysis of the gblup, embayesb, and gwas algorithms to predict genetic values in large yellow croaker (larimichthys crocea). *BMC Genomics, 17*(1), 460.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning*: vol. 1. Springer series in statistics New York.

Gambella, C., Ghaddar, B., & Naoum-Sawaya, J. (2021). Optimization problems for machine learning: A survey. *European Journal of Operational Research, 290*(3), 807–828.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing, 21*(2), 178–192.

Han, Y., Cameron, J. N., Wang, L., & Beavis, W. D. (2017). The predicted cross value for genetic introgression of multiple alleles. *Genetics, 205*(4), 1409–1423.

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Howe, A., & Pyeatt, L. (1998). Decision Tree Function Approximation in Reinforcement Learning. *Technical Report*. Colorado State University.

Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., & Wassick, J. M. (2020). Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems, 30*.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. *arXiv preprint arXiv:1803.08475*.

Li, H., Wang, J., & Bao, Z. (2015). A novel genomic selection method combining gblup and lasso. *Genetica, 143*(3), 299–304.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Liu, Y., Lu, S., Liu, F., Shao, C., Zhou, Q., Wang, N., Li, Y., Yang, Y., Zhang, Y., Sun, H., et al. (2018). Genomic selection using bayescπ and gblup for resistance against edwardsiella tarda in japanese flounder (paralichthys olivaceus). *Marine Biotechnology, 20*(5), 559–565.

Liu, Y., & Wang, D. (2017). Application of deep learning in genomic selection. In *2017 ieee international conference on bioinformatics and biomedicine (bibm)* (p. 2280). IEEE Computer Society.

Lobo, I., & Shaw, K. (2008). Thomas hunt morgan, genetic recombination and gene mapping. *Nature Education*.

Mahendran, N., Vincent, P. D. R., Srinivasan, K., & Chang, C.-Y. (2020). Machine learning based computational gene selection models: A survey, performance evaluation, open issues, and future research directions. *Frontiers in Genetics, 11*.

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research, 134*, 105400.

Meuwissen, T. H. E., Hayes, B. J., & Goddard, M. E. (2001). Prediction of total genetic value using genome-wide dense marker maps. *Genetics, 157*(4), 1819–1829. doi:11290733

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Moeinizade, S. (2018). A stochastic simulation approach for improving response in genomic selection. *Master's Thesis*.

Moeinizade, S. (2021). *Simulation-Based Optimization and Reinforcement Learning Methods to Improve Decision Making in Agriculture*. Iowa State University Ph.D. thesis..

Moeinizade, S., Han, Y., Pham, H., Hu, G., & Wang, L. (2021). A look-ahead monte carlo simulation method for improving parental selection in trait introgression. *Scientific Reports, 11*(1), 1–12.

Moeinizade, S., Hu, G., Wang, L., & Schnable, P. S. (2019). Optimizing selection and mating in genomic selection with a look-ahead approach: An operations research framework. *G3: Genes, Genomes, Genetics, 9*(7), 2123–2133.

Moeinizade, S., Kusmec, A., Hu, G., Wang, L., & Schnable, P. S. (2020a). Multi-trait genomic selection methods for crop improvement. *Genetics, 215*(4), 931–945.

Moeinizade, S., Wellner, M., Hu, G., & Wang, L. (2020b). Complementarity-based selection strategy for genomic selection. *Crop Science, 60*(1), 149–156.

Montesinos-López, O. A., Montesinos-López, A., Crossa, J., Gianola, D., Hernández–Suárez, C. M., & Martín-Vallejo, J. (2018). Multi-trait, multi-environment deep learning modeling for genomic-enabled prediction of plant traits. *G3: Genes, Genomes, Genetics, 8*(12), 3829–3840.

Neves, H. H., Carvalheiro, R., & Queiroz, S. A. (2012). A comparison of statistical methods for genomic selection in a mice population. *BMC Genetics, 13*(1), 100.

Nowak, A., Villar, S., Bandeira, A. S., & Bruna, J. (2018). Revised note on learning quadratic assignment with graph neural networks. In *2018 ieee data science workshop (dsw)* (pp. 1–5). IEEE.

Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality*: vol. 703. John Wiley & Sons.

Pryce, J., Gredler, B., Bolormaa, S., Bowman, P., Egger-Danner, C., Fuerst, C., Emmerling, R., Sölkner, J., Goddard, M., & Hayes, B. (2011). Genomic selection using a multi-breed, across-country reference population. *Journal of Dairy Science, 94*(5), 2625–2630.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Nips: vol. 99* (pp. 1057–1063). Citeseer.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning, 4*(1), 1–103.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., & Levine, S. (2018). The mirage of action-dependent baselines in reinforcement learning. *arXiv preprint arXiv:1802.10031*.

Van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995–2003).

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292.