Abstract

This work deals with the problem of function learning by genetic algorithms where the function is used as a preference predicate. In such a case, learning the exact function is not necessary since any function that preserves the order induced by the target function is sufficient. The paper presents a methodology for solving the problem with genetic algorithms. We first consider the representation issues involved in learning such a function, and conclude that canonical representation, relative coding, and search restrictions, are required. We then show that the traditional homologous genetic operators are not appropriate for such learning, and introduce a new configurable analogous genetic operator, named derivative crossover. This operator works on the derivative of the chromosomes and is therefore suitable for preference predicate learning where only the relative values of the functions are important. We support our methodology by a set of experiments performed in the domain of continuous function learning and in the domain of evaluation-function learning for game-playing. The experiments show that indeed using derivative operators increases the speed of learning significantly.

†E-mail address: david@CS.Technion.AC.IL
1 Introduction

The problem of function learning has been studied by several researchers in the GA community [6, 9, 3] and is recognized as a difficult problem due to the large space of possible functions. In many cases, however, the learned function is used as a preference predicate [10] to compare the relative merit of objects, e.g., states in a search space [9]. In such cases, there is no need to learn the exact function. It is sufficient to learn any function that induces the same order over the set of compared objects.

This work studies the problem of learning functions by genetic algorithms under the following conditions:

1. The target function $f \in \mathcal{F}$ is needed for its use as a preference predicate over $\text{Objects} \times \text{Objects}$ for comparing objects.

2. A preference predicate over $\mathcal{F} \times \mathcal{F}$, that allows comparing two alternative functions, is supplied.

3. No other fitness measure of functions is available.

The hypothesis space is essentially a set of relations, where a function $f \in \mathcal{F}$ encodes a preorder $\sqsubseteq_f$ over $\text{Objects} \times \text{Objects}$ such that for all $x, y \in \text{Objects}$, $x \sqsubseteq_f y \iff f(x) \leq f(y)$. For example, if $S = \langle s_1, s_2, s_3, s_4, s_5 \rangle$ is an ordered set of objects, then the function

$$f(s_i) = [2\sin(2i)]$$

as a preference predicate over $S \times S$ encodes the preorder $s_2 \sqsubseteq s_3 = s_5 \sqsubseteq s_1 = s_4$ depicted in Fig. 1.

The learning problem can be represented as an ordered partitioning problem as follows: Find and order the equivalent classes induced by the kernel of $\sqsubseteq_f$, $f$ being the target function. The target ordered partition for (1) is:

$$\langle \{s_2\} \sqsubseteq \{s_3, s_5\} \sqsubseteq \{s_1, s_4\} \rangle$$

Denoting each class in the target partition by a letter from an ordered alphabet $\langle \mathcal{C}, \preceq \rangle$, the learning problem can be represented as the following classification problem: Find a classification function $\text{class}: \text{Objects} \rightarrow \mathcal{C}$ such that $x \sqsubseteq_f y \iff \text{class}(x) \preceq \text{class}(y)$.

In the evolution of classification functions as preference predicates, we allow only a competitive fitness measure [1] to be used in guiding the search in the hypothesis space.
Figure 1: $f(x) = [2sin(2x)]$ as a preference predicate over $S \times S$.

Outline. In the next section we discuss representational issues involved in applying genetic algorithms for such classification problems, considering design and coding decisions. We show that the straightforward vector representation introduces redundancy and therefore increases the size of the hypothesis space unnecessarily, and describe a method for reducing this problem. We proceed with showing in Sect. 3 that the traditional crossover operator is not appropriate for learning preference predicates, and suggest a new analogous configurable derivative crossover operator. In Sect. 4 we demonstrate the potential of this new methodology by performing experiments in function learning. Finally, in Sect. 5 we offer conclusions.

2 The Search Space

Each classification function can be encoded as a vector over $C^l, l \in \mathbb{N}$, represented by a chromosome $(t_1, t_2, \ldots, t_l)$, whose $i^{th}$ element, $t_i \in C$, is the class assigned to the $i^{th}$ object. For example, setting $C = \{a, b, c\}$, the target vector for (2) would be the chromosome: $\langle c, a, b, c, b \rangle$.

2.1 Vector Representation

Using a vector representation requires two practical decisions:

1. Deciding on the size of the vector, i.e., the chromosome length.
2. Deciding on the size of the alphabet, i.e., the number of classes.
Since it is unreasonable to expect the chromosome length be the total number of objects, we assume that each object is associated with some index (attribute) \(1 \leq i \leq l, i \in \mathbb{N}\), where \(l\) is the length of the chromosome. Therefore, the target preference-predicate is actually a double mapping: Two compared objects \(x, y \in \text{Objects}\) are first mapped to their corresponding gene locations using a revealed ordering function known to the learner,

\[
\text{Index} : \text{Objects} \rightarrow \mathbb{N}
\]  

Then \(\text{Index}(x)\) and \(\text{Index}(y)\) are mapped to two classes using the learned (concealed) function,

\[
f : \mathbb{N}^{\text{Chromosome}} \rightarrow \mathcal{C}
\]  

The object mapped to a higher class is preferred. For the remaining of this paper we assume \(\text{Dom}(f)\) to be \(\text{Range(\text{Index})}\), and concentrate in evolving (4).

Deciding a priori on the number of classes, \(n = |\mathcal{C}|\), can lead to two types of problems. Let \(n_{\text{optimal}}\) be the size of the partition induced by the target preference-predicate. Either \(n < n_{\text{optimal}}\) in which case the representation scheme is too restrictive, or \(n > n_{\text{optimal}}\) in which case the representation scheme is too expressive. In both cases the same relation may have multiple representations. For example, setting \(\mathcal{C} = \langle b, b \rangle\), the target partition (2) cannot be expressed, and yet two distinct vectors might encode the same relation, e.g., \(\langle a, a, a, a, a \rangle\) and \(\langle b, b, b, b, b \rangle\). Setting \(\mathcal{C} = \langle a, b, c, d \rangle\), the four vectors \(\langle c, a, b, c, b \rangle\), \(\langle d, a, b, d, b \rangle\), \(\langle d, a, c, d, c \rangle\), and \(\langle d, b, c, d, c \rangle\), encode the same partition (2).

To prevent the scheme from being too restrictive, an infinite alphabet,

\[
\mathcal{C} = \langle \ldots, c_{-2}, c_{-1}, c_0, c_1, c_2, \ldots \rangle
\]  

is assumed. To prevent coding from being redundant in the sense that the number of possible encodings exceeds the number of classifiers, a canonical representation is used. A canonical classification function maps objects on a condensed finite subset \(\tilde{\mathcal{C}} \subseteq \mathcal{C}\), forcing the entire range of \(\tilde{\mathcal{C}}\) to be used, and setting the first position in the vector to \(c_0\). The vector \(\langle d, a, b, d, b \rangle\), for example, is not in a canonical form since \(\langle a, b, d \rangle\) is not condensed. \(\langle c, a, b, c, b \rangle\) is not canonical since its first gene is not ‘a’. However, allowing negative gene values, \(\langle a, -c, -b, a, -b \rangle\) is a canonical representation for (2). Having
decided not cut down the number of classes in $C$, we perform a qualitative reduction of the search space according to function smoothness characteristics, allowing the number of classes to be determined dynamically.

## 2.2 Smoothness

In order to make the search space tractable, it is often required to limit the search to a subspace. The supplied index mapping (3) already reduced the search space by partitioning the domain into equivalent classes of objects. Objects that are mapped to the same gene location are considered a priori equally desirable. We further exploit the knowledge available by (3), and assume that objects that were mapped to adjacent gene locations have similar desirability, and thus should belong to similar classes. This assumption is incorporated by adding a restriction forcing successive genes to be mapped to neighboring or identical classes. The corresponding classification function (4) is hence assumed to be smooth.

Given the ordered alphabet (5), define $\|c_i - c_j\| \triangleq |i - j|$. The desired smoothness of the classification function may be imposed by keeping $M$-bounded slopes:

$$\forall x, y \in \text{Dom}(f), \quad \frac{\|f(x) - f(y)\|}{\|x - y\|} \leq M \quad (6)$$

If we denote

$$f'(x) = \frac{\|f(x) - f(x - 1)\|}{\|x - (x - 1)\|} = \frac{\|f(x) - f(x - 1)\|}{1} \quad (7)$$

then having

$$\|f'(x)\| \leq M \quad \forall x \in \text{Dom}(f) \quad (8)$$

guarantees $M$-bounded slopes. Under the smoothness assumption, the search can be focuses on a subspace, $\mathcal{F}_M \subset \mathcal{F}$, of functions with an $M$-bounded derivative.

## 3 Configurable Derivative Operators

The traditional homologous-positional one-point crossover does not work well with the suggested representation. The number of classes is never increased,
(except perhaps by mutation, which we ignore at this state), requiring their number to be determined in advance. A produced offspring of canonical parents might not be in a canonical form, requiring post-production adjustments. Crossing of two smooth functions, might produce a non-smooth child, spending learning resources on candidates regarded a priori illegal. Lastly, the order between two objects, which both parents agree on, might be scrambled in the child. For example, the two vectors \(\langle a, b, c, d, d \rangle\) and \(\langle a, a, a, b, c \rangle\) agree that \(s_3 \sqsubseteq s_4\). However, \(s_3 \sqsubseteq s_4\) in the vector \(\langle a, b, c, b, c \rangle\) obtained by crossing these vectors after position 3.

### 3.1 Motivation

Generalization has an important role in learning systems. Relative operators, as opposed to absolute operators, allows generalization to evolve more naturally. The idea presented is to preserve the order between adjacent genes of the parents. In the above crossover example, for instance, the parents vectors have genes’ order \(\langle * \sqsubseteq * \sqsubseteq * \sqsubseteq * = \rangle\) and \(\langle * = * = * \sqsubseteq * \sqsubseteq * \rangle\) respectively, thus \(\langle * \sqsubseteq * \sqsubseteq * \sqsubseteq * \sqsubseteq * \rangle\) is not a legitimate child.

**Example 1** Suppose our domain includes five objects, \(\{s_1, s_2, s_3, s_4, s_5\}\), and that the target partition is \(\langle \{s_1\}, \{s_2, s_3\}, \{s_4, s_5\}\rangle\). Assume that the following two promising preference predicates have emerged. A chromosome \(\langle a, b, b, b, b \rangle\) which succeeded in capturing the precedence of objects \(s_2, s_3, s_4, s_5\) over the object \(s_1\), encoding the partition \(\langle \{s_1\}, \{s_2, s_3\}, \{s_4, s_5\}\rangle\), and a chromosome \(\langle a, a, a, b, b \rangle\) which failed yet to distinguish between \(s_1\) and \(s_2\) but managed to learn an equally significant precedence of objects \(s_4, s_5\) over object \(s_1, s_2, s_3\), and encodes the partition \(\langle \{s_1, s_2, s_3\}, \{s_4, s_5\}\rangle\). A positional crossover after position 2, would produce the undesirable chromosome

\[
\text{Crossover}(\langle a, b, b, b, b \rangle, \langle a, a, a, b, b \rangle) = \langle a, b, a, a, b \rangle
\]

encoding the partition \(\langle \{s_1, s_3\}, \{s_2, s_4, s_5\}\rangle\). On the other hand, applying an analogous crossover \([4]\) that preserves the relations between two neighboring genes rather than their absolute values, could produce the chromosome

\[
\text{Crossover}(\langle * \sqsubseteq * = * = \rangle, \langle * = * = * \sqsubseteq * \rangle) = \langle * \sqsubseteq * = * \sqsubseteq * = \rangle
\]
i.e., \( \langle a, b, b, c, c \rangle \), combining this way both parents’ information and automatically introducing a third new class, ‘c’.

### 3.2 Derivative Crossover

Let \( \mathcal{F} \) be the set of functions \([0, 1] \rightarrow \mathbb{R} \). The traditional one-point Crossover can be view as an operator that, given a random crossing point \( p \in \mathbb{R} [0, 1] \), returns an operator \( \text{Crossover}_p \) over \( \mathcal{F} \times \mathcal{F} \). Let \( f, g \in \mathcal{F} \) be two functions. \( \text{Crossover}_p(f, g) \) returns the function combined out of \( f \)'s part over the interval \([0, p]\) and \( g \)'s part over the interval \((p, 1]\). For simplicity we assume only one offspring per crossover operation, and consider the other offspring to be \( \text{Crossover}_p(g, f) \).

Using a step function \( \sigma_p(x) = \begin{cases} 0 & x \leq p \\ 1 & x > p \end{cases} \) and using a lambda expression \( \lambda v.e \), to denote a function whose result is the value of \( e \) in an environment where \( v \) stands for the argument of the function, we can express \( \text{Crossover}_p(f, g) \) as:

\[
\text{Crossover}_p(f, g) = \lambda x. (1 - \sigma_p(x)) f(x) + \sigma_p(x) g(x)
\]

The derivative crossover performs one-point crossover on the derivatives of the two functions, and then integrates the result to produce the new offspring:

\[
\text{Der.Crossover}_p(f, g) = \lambda x. \int_0^x \text{Crossover}_p(f', g')
\]

This can be rewritten as:

\[
\text{Der.Crossover}_p(f, g) = \lambda x. (1 - \sigma_p(x)) \int_0^x f'(x) + \sigma_p(x) \left( f(p) + \int_p^x g'(x) \right) = \lambda x. (1 - \sigma_p(x)) f(x) + \sigma_p(x) (g(x) + f(p) - g(p))
\]

Both the standard crossover and the derivative crossover can be seen as special cases of a general family of derivative operators defined as follows. Let \( \rho \) be a function \( \mathbb{R} \rightarrow \mathbb{R} \). A derivative crossover using \( \rho \) is defined by:

\[
\text{Der.Crossover} : (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow ([0, 1] \rightarrow (\mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}))
\]

\[
\text{Der.Crossover}[\rho]_p(f, g) = \lambda x. (1 - \sigma_p(x)) f(x) + \sigma_p(x) (g(x) + \rho (f(p) - g(p)))
\]
The function \( \rho \) determines the method of combination at the crossing point, as illustrated in Fig. 2. With \( \rho \equiv 0 \), (12) become the traditional crossover (9). Using the identity function \( \rho(x) = x \), (12) is simply (11). There are many other meaningful combinations worth considering, e.g., \( \rho(x) = x + \text{sign}(x) \).

![Figure 2: Derivative Crossover at position \( p \), a schematic view of (12).](image)

The derivative mutation performs conventional mutation to the function’s derivative, and then integrates the result to produce the mutated offspring:

\[
\text{Der.\,Mutation}_p(f) = \lambda x. \int_0^x \text{Mutation}_p(f'(x))
\]

It is interesting to note that (13) may also be seen as a special case of (12), by setting \( \rho(x) = x + \mu \), i.e., \( \text{Der.\,Crossover}[\lambda x. x + \mu]_\rho(f, f), \mu \) picked at random. Equation (12) is also applicable for discrete functions, as demonstrated in the next example.

**Example 2** Performing a derivative crossover after position 3 of two chromosomes of length 5 over the ordered alphabet \( \ldots, -c, -b, a, b, c, \ldots \) might be:

\[
\text{Der.\,Crossover}[\lambda x. x]_3 (\langle a, b, c, b, c \rangle, \langle a, b, b, c, d \rangle) = \\
= \langle a, b, c, c + (c - b), d + (c - b) \rangle = \\
= \langle a, b, c, d, e \rangle
\]

Similarly, performing a derivative mutation after position 2 might result in:

\[
\text{Der.\,Mutation}_2 (\langle a, b, c, b, c \rangle) = \\
= \text{Der.\,Crossover}[\lambda x. x - 2]_2 (\langle a, b, c, b, c \rangle, \langle a, b, c, b, c \rangle) = \\
= \langle a, b, c + (0 - 2), b + (0 - 2), c + (0 - 2) \rangle = \\
= \langle a, b, a, -b, a \rangle
\]
4 Comparative Results

Learning quality and convergence rates were compared between the traditional scheme and the proposed derivative frame. The traditional operators were tested with different sizes of alphabet. The derivative operators were applied on different subspaces $\mathcal{F}_M$, using an infinite alphabet.

The experiments were carried out in the following manner: the population size was kept fixed at 64, the number of generations was 100. Fitness values were assigned according to overall performance. Crossover and mutation were applied with a low probability of 0.1. Rank selection was used.

4.1 Continuous Functions

In a set of experiments, preference predicates were created from continuous functions. The system ability to reveal the concealed function was tested. For this purpose a preference predicate "better than" over pairs of preference predicates was defined as follows:

1. Preference predicate $\mathcal{P}$ is said to be better than preference predicate $\mathcal{Q}$, if it wins more matches in a tournament among the set of competing preference predicates.

2. Preference predicate $\mathcal{P}$ is said to win a match against preference predicate $\mathcal{Q}$, if preference predicate $\mathcal{P}$ correctly orders more out of $n$ random pairs of objects, competing against preference predicate $\mathcal{Q}$, letting $n \to \infty$.

In each generation a full tournament was conducted with the chromosomes as the classifiers applied. Two compared chromosomes were independently used to order 500 randomly generated pairs of points. The chromosome that had more correctly ordered pairs $\langle x, y \rangle$, won the match. An ordered pair $\langle x, y \rangle$ was considered correct if $x \silver f y$, where $f$ is the concealed function. The precise identity or quantity of correctly ordered pairs was not revealed to the learner, just as in a real object-space situation.

Figure 3 shows best chromosomes evolved with different genetic schemes for the target function $f(x) = 10 \left[ \sin \frac{2\pi x}{50} \right]$. Traditional operators were applied with alphabet sizes 6, 12, 24, and $\text{Der.Crossover}[\lambda x.x]$ operator was applied over $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$. Points represent genes' values. Solid lines represent
the target function. Using derivative operators achieved better preference predicates in less generations. Similar results were obtained with more complex functions, as well as with longer chromosomes.

4.2 Game Playing

In a second series of experiments the system was tested in a real state-space condition. The domain of game playing was chosen, and the task was to learn the best evaluation function for Checkers, based on a single attribute [7]. Figure 4 shows the performance of best individuals of the first 40 generation, tested against an external player. The benefit of derivative learning compared to learning with the standard genetic operator, is especially important in this domain since calculating the fitness measure requires considerable computing resources.

5 Concluding Remarks

In this paper we study the use of genetic algorithms for learning preference predicates, as follows.

1. We offer a methodology of learning the derivatives of a function instead of the function itself.

2. We introduce a family of configurable derivative crossover operators that preserves the relativeness among neighboring genes.

Canonical representation and smoothing restrictions allowed better exploitation of the learning resources in the genetic search. In a series of experiments performed with the new operators, the program acquired arbitrary continuous functions and an evaluation function for the game of Checkers, in significant speedup.

In contrast to the traditional operators, the derivative operators are more suitable for order dependent domains. They preserves canonical representation and bounded-slopes, and allows the number of classes to be dynamically changed. Gene values are allowed to be negative. Rather than a quantitative limitation of the alphabet, a qualitative smoothness characteristics may be imposed. \( \mathcal{F}_M \) is closed under the \( \text{Der.Crossover} \) \([\lambda x.x]\) operator. Objects
Figure 3: Best chromosomes using traditional crossover with different alphabet sizes \((n)\) on the left, compared with best chromosomes using derivative crossover over \(F_M\) with different smoothness restrictions \((M)\) on the right, as evolved for the target function \(f(x) = 10 \left[ \sin \frac{2\pi x}{30} \right]\) after 100 generations using chromosomes of length 50.
Figure 4: Learning using traditional operators with different alphabet sizes ($n$) compared with learning using derivative operators on $F_1$, in evolving an evaluation function for the game of Checkers.

which both parents agree on their order relative to the cutting position, are kept in the same order in the produced offspring, preserving relativeness of genes rather than absolute values.

An important feature of the derivative operators, yet not common among analogous operators [4, 5], is their compatibility with traditional positional operators. Moreover, different kinds of derivative operators can be applied with different probabilities on the same population, just as crossover and mutation are normally used. A potential user can decide what analogous derivative information is wished to be preserved and with what probability, and apply appropriate derivative crossover operators accordingly.

Acknowledgments I would like to thank Yuval Davidor and Shaul Markovitch for inspiring discussions.
References


