Why Convex Combination Is an Effective Crossover Operation in Continuous Optimization: A Theoretical Explanation

Kelly Cohen, Olga Kosheleva, and Vladik Kreinovich

Abstract When evolutionary computation techniques are used to solve continuous optimization problems, usually, convex combination is used as a crossover operation. Empirically, this crossover operation works well, but this success is, from the foundational viewpoint, a challenge: why this crossover operation works well is not clear. In this paper, we provide a theoretical explanation for this empirical success.

1 Formulation of the Problem

Biological evolution has successfully managed to come up creatures that fit well with different complex environments, i.e., creatures for each of which the value of its fitness is close to optimal. So, when we want to optimize a complex objective function, a natural idea is to emulate biological evolution. This idea is known as *evolutional computation* or *genetic programming*.

Each living creature is described by its DNA, i.e., in effect, by a discrete (4-ary) code. In other words, each creature is characterized by a sequence $x_1x_2\ldots$, where each x_i is an element of a small discrete set (4-element set for biological evolution). Environment determines how fit is a creature is: the higher the fitness level, the higher the probability that this creature will survive – and also the higher the probability that it will be able to mate and give birth to offsprings. During the lifetime, the code is slightly changed by mutations – when some of the values x_i changes with a small probability.

Kelly Cohen
Department of Aerospace Engineering and Engineering Mechanics
College of Engineering and Applied Science, University of Cincinnati, 2850 Campus Way
Cincinnati, OH 45221-0070, USA, e-mail: Kelly.Cohen@uc.edu

Olga Kosheleva and Vladik Kreinovich
University of Texas at El Paso, 500 W. University, El Paso, Texas 79968, USA
e-mail: olgak@utep.edu, vladik@utep.edu

The most drastic changes happen when a new creature appears: its code is obtained from the codes of the two parents by a special procedure called *crossover*. For biological creatures, crossover is organized as follows: the two DNA sequences are placed together, then a few random locations $i_1 < i_2 < \ldots$ are marked on both sequences, and the resulting DNA is obtained by taking the segment of the 1st parent until the first location, then the segment of the second parent between the 1st and 2nd locations, then again the segment from the 1st parent, etc. In this scheme, for each location i, the child's value c_i at this location is equal:

- either to the value f_i of the first parent at this location,
- or to the values s_i of the second parent at this location.

To be more precise:

- for $i < i_1$, we take $c_i = f_i$;
- for $i_1 \le i < i_2$, we take $c_i = s_i$,
- for $i_2 \le i < i_3$, we take $c_i = f_i$, etc.

This is how crossover is performed in nature, this is how it is performed in discrete optimization problems, where each possible option is represented as a sequence of discrete units – e.g., 0-or-1 bits.

Similar techniques have been proposed – and successfully used – for continuous optimization problems, in which each possible alternative is represented by a sequence $x_1 x_2 \ldots$ of real numbers. In this arrangement – known as *geometric genetic programming* – mutations correspond to small changes of the corresponding values, and crossover is usually implemented as a convex combination

$$c_i = \alpha \cdot f_i + (1 - \alpha) \cdot s_i \tag{1}$$

for some real value α – which may different for different locations i; see, e.g., [1, 2, 3, 4].

Empirically, the crossover operation (1) works really well, but why it works well is not clear. In this paper, we provide a theoretical explanation for its efficiency.

2 Our Explanation

Usually, the numbers x_i that characterize an alternative are values of the corresponding physical quantities. For example, if we are designing a car, the values x_i can be lengths, heights, weights, etc. of its different components. The numerical value of each physical quantity depends on the choice of a measuring unit and, for some properties like temperature or time, on the choice of the starting point. If we replace the original measuring unit by a new unit which is a times smaller, then all the numerical values are multiplied by a: $x \mapsto a \cdot x$. For example, if we replace meters with 100 times smaller unit – centimeter – then 2 m becomes $100 \cdot 2 = 200$ cm. For some quantities like electric current, the sign is also arbitrary, so we can change x to $-x$. If we replace the original starting point with a new starting point which is b

units earlier, then this value b is added to all the numerical values: $x \mapsto x + b$. If we change both the measuring unit and the starting point, then we get a generic linear transformation $x \mapsto a \cdot x + b$.

If we change the measuring unit and the starting point, then the numerical values change but the actual values of the physical quantities do not change. Thus, a reasonable crossover operation should not change if we simply change the measuring unit and the starting point: it does not make sense to use one optimization algorithm if we have meters and another one if we use feet or centimeters. It turns out that this natural requirement explains the appearance of the convex combination crossover operation (1).

Let us describe this result in precise terms.

Definition 1. *By a* crossover operation*, we mean a real-value function of two real-valued inputs $F : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.*

Definition 2. *We say that a crossover operation is* scale-invariant *if for all $a \neq 0$ and b, whenever we have $c = F(f, s)$, we will also have $c' = F(f', s')$, where $c' \stackrel{\text{def}}{=} a \cdot c + b$, $f' \stackrel{\text{def}}{=} a \cdot f + b$, and $s' \stackrel{\text{def}}{=} a \cdot s + b$.*

Proposition 1. *A crossover operation is scale-invariant if and only if it has the form $F(s, f) = \alpha \cdot f + (1 - \alpha) \cdot s$ for some α.*

Discussion. This results explains why crossover (1) is effective: it is the only crossover operation that satisfies the reasonable invariance requirements.

Comment. Usually, only the values α from the interval $[0, 1]$ are used. However, Proposition 1 allows values α outside this interval. So maybe such value $\alpha < 0$ or $\alpha > 1$ can be useful in some optimization applications?

Proof. It is easy to check that the crossover operation (1) is scale-invariance. Let us prove that, vice versa, every scale-invariant crossover operation has the form (1).

Let us denote $\alpha \stackrel{\text{def}}{=} F(1, 0)$. By scale-invariance, for each $a \neq 0$ and b, we have $a \cdot \alpha + b = F(a \cdot 1 + b, a \cdot 0 + b)$. For each two different numbers $f \neq s$, we can take $a = f - s$ and $b = s$. In this case, $a \cdot 1 + b = f - s + s = f$, $a \cdot 0 + b = s$, and $a \cdot \alpha + b = (f - s) \cdot \alpha + s = \alpha \cdot f + (1 - \alpha) \cdot s$, and thus, scale-invariance implies that $\alpha \cdot f + (1 - \alpha) \cdot s = F(f, s)$.

The desired equality is thus proven for all the cases when $f \neq s$. So, to complete the proof, it is sufficient to prove this equality for the case when $f = s$. In this case, scale-invariance means that if $c = F(f, f)$, then for every $a \neq 0$ and b, we should have $a \cdot c + b = F(a \cdot f + b, a \cdot f + b)$. Let us take $a = 2$ and $b = -f$. In this case, $a \cdot f + b = 2 \cdot f + (-f) = f$, so we get $2c - f = F(f, f)$. We already know that $c = F(f, f)$, thus $2c - f = c$ and therefore, $c = f$. In this case, the right-hand side of the desired equality is equal to $\alpha \cdot f + (1 - \alpha) \cdot f = f$, so the desired equality is also true.

The proposition is proven.

3 An Alternative Explanation

In the previous section, we proved that the currently used crossover operation is the only one that satisfies some reasonable requirement. In this section, we will show that this operation is also optimal in some reasonable sense.

To explain this result, let us first recall what optimal means. In many cases, optimal means attaining the largest or the smallest value of some objective function $G(x)$. In this case, the optimality criterion is straightforward: an alternative x is better than the alternative y if – for the case of minimization – we have $G(x) < G(y)$. For example, we can select the crossover operation for which the average degree of sub-optimality is the smallest possible, i.e., for which the results are, on average, the closest to the desired optimum of the fitness function.

However, not all optimal situations are like that. For example, if may turn out that several different crossover operations lead to the same average degree of sub-optimality. In this case, we can use this non-uniqueness to optimize some other function $H(x)$: e.g., the worst-case degree of sub-optimality. In this case, the optimality criterion is more complex that simply comparing the values of a single function – here, x is better than y if:

- either $G(x) < G(y)$
- or $G(x) = G(y)$ and $H(x) < H(y)$.

If this still leaves us with several equally optimal alternatives, this means that our optimality criterion is not final – we can again use the non-uniqueness to optimize something else. For a final optimality criterion, there should be only one optimal alternative.

What all the resulting complex optimality criteria have in common is that they allow, for at least some pairs of alternatives a and b, to decide:

- whether a is better according to this criterion – we will denote it by $a > b$,
- or whether b is better,
- or whether these two alternatives are equally good; we will denote it by $a \sim b$.

Of course, these decisions must be consistent: e.g., if a is better than b and b is better than c, then a should be better than c. Thus, we arrive at the following definition.

Definition 3. *Let A be a set. Its elements will be called* alternatives.

- *By an* optimality criterion *on the set A, we mean a pair of binary relations $\langle >, \sim \rangle$ that satisfy the following conditions for all a, b, and c:*
  - *if $a > b$ and $b > c$, then $a > c$;*
  - *if $a > b$ and $b \sim c$, then $a > c$;*
  - *if $a \sim b$ and $b > c$, then $a > c$;*
  - *if $a \sim b$ and $b \sim c$, then $a \sim c$;*
  - *if $a > b$, then we cannot have $a \sim b$;*
  - *we have $a \sim a$.*

- *We say that an alternative a_{opt} is* optimal *if for every $a \in A$, either $a_{\text{opt}} > a$ or $a_{\text{opt}} \sim a$.*
- *We say that the optimality criterion is* final *if there exists exactly one optimal alternative.*

In our case, alternatives are crossover operations. It is reasonable to require that which crossover operation is better should not depend on re-scaling the corresponding quantity. Let us describe this natural requirement in precise terms. Let us denote the corresponding re-scaling by $T_{a,b}(x) \stackrel{\text{def}}{=} a \cdot x + b$. If we apply a crossover operation $F(f, s)$ in a new scale, this means that in the new scale, we get the value $c' = F(f', s') = F(T_{a,b}(f), T_{a,b}(s))$. If we transform this value into the original scale, i.e., apply the inverse operation $T^{-1}_{a,b}(c') = a^{-1} \cdot (c' - b)$, then we get $c = T^{-1}_{a,b}(F(T_{a,b}(f), T_{a,b}(s)))$. Thus, the use of the crossover operation F is the new scale is equivalent to applying a crossover operation $T_{a,b}(F)$ defined as $(T_{a,b}(F))(f, s) \stackrel{\text{def}}{=} T^{-1}_{a,b}(F(T_{a,b}(f), T_{a,b}(s)))$ in the original scale. In these terms, invariance means that, e.g., $F > F'$, then we should have $T_{a,b}(F) > T_{a,b}(F')$, etc.

Definition 4.

- *For every two numbers $a \neq 0$ and b, let us denote $T_{a,b}(x) \stackrel{\text{def}}{=} a \cdot x + b$ and*

$$T^{-1}_{a,b}(x) = a^{-1} \cdot (x - b).$$

- *For each $a \neq 0$ and b, and for each crossover operation $F(a, b)$, by $T_{a,b}(F)$, we will denote a crossover operation defined as*

$$(T_{a,b}(F))(f, s) \stackrel{\text{def}}{=} T^{-1}_{a,b}(F(T_{a,b}(f), T_{a,b}(s))).$$

- *We say that an optimality criterion on the set of all crossover operations is* scale-invariant *if for all crossover operations F and F' and for all $a \neq 0$ and b, $F > F'$ implies that $T_{a,b}(F) > T_{a,b}(F')$ and $F \sim F'$ implies that $T_{a,b}(F) \sim T_{a,b}(F')$.*

Proposition 2. *For every scale-invariant final optimality criterion on the set of all crossover operations, the optimal operation has the form*

$$F(s, f) = \alpha \cdot f + (1 - \alpha) \cdot s$$

for some α.

Proof. Let F_{opt} be the optimal crossover operation. This means that for every other crossover operation F, we have $F_{\text{opt}} > F$ or $F_{\text{opt}} \sim F$. In particular, this is true for operations $T^{-1}(a, b)(F)$, i.e., we have either $F_{\text{opt}} > T^{-1}(a, b)(F)$ or $F_{\text{opt}} \sim T^{-1}(a, b)(F)$. Due to the fact that the optimality criterion is scale-invariant, we get either $T_{a,b}(F_{\text{opt}}) > F$ or $T_{a,b}(F_{\text{opt}}) \sim F$. This is true for all crossover operations F. Thus, by definition of an optimal alternative, the operation $T_{a,b}(F_{\text{opt}})$ is optimal. However, we already know that the operation F_{opt} is optimal, and we assumed that the optimality criterion is final, which means that there is only one optimal alternative. Thus, we have $T_{a,b}(F_{\text{opt}}) = F_{\text{opt}}$.

This equality holds for all $a \neq 0$ and b. Thus, the crossover operation F_{opt} is scale-invariant. For such operations, we have already proved, in Proposition 1, that they have the desired form. The proposition is proven.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes), and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References

1. M. Castelli and L. Manzoni, "GSGP-C++ 2.0: A geometric semantic genetic programming framework", *SoftwareX*, 2019, Vol. 10, Paper 100313.
2. K. Krawiec and T. Pawlak, "Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators", *Genetic Programming and Evolvable Machines*, 2013, Vol. 14, pp. 31–63.
3. A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming", In: *Parallel Problem Solving from Nature PPSN'XII*, Springer, Heidelberg, Germany, 2012, pp. 21–31.
4. T. P. Pawlak, B. Wieloch, and K. Krawiec, "Review and comparative analysis of geometric semantic crossovers", *Genetic Programming and Evolvable Machines*, 2015, Vol. 16, pp. 351–386.