Partition Crossover for Continuous Optimization: ePX

Renato Tinós
University of São Paulo
Ribeirão Preto, São Paulo, Brazil
rtinos@ffclrp.usp.br

Darrell Whitley
Colorado State University
Fort Collins, Colorado, USA
whitley@cs.colostate.edu

Francisco Chicano
University of Málaga
Malaga, Spain
chicano@lcc.uma.es

Gabriela Ochoa
University of Stirling
Stirling, Scotland, UK
gabriela.ochoa@cs.stir.ac.uk

ABSTRACT

Partition crossover (PX) is an efficient recombination operator for gray-box optimization. PX is applied in problems where the objective function can be written as a sum of subfunctions $f_l(.)$. In PX, the variable interaction graph (VIG) is decomposed by removing vertices with common variables. Parent variables are inherited together during recombination if they are part of the same connected recombining component of the decomposed VIG. A new way of generating the recombination graph is proposed here. The VIG is decomposed by removing edges associated with subfunctions $f_l(.)$ that have similar evaluation for combinations of variables inherited from the parents. By doing so, the partial evaluations of $f_l(.)$ are taken into account when decomposing the VIG. This allows the use of partition crossover in continuous optimization. Results of experiments where local optima are recombined indicate that more recombining components are found. When the proposed *epsilon-PX* (ePX) is compared with other recombination operators in Genetic Algorithms and Differential Evolution, better performance is obtained when the epistasis degree is low.

CCS CONCEPTS

• **Mathematics of computing** → **Combinatorial optimization**;
• **Theory of computation** → **Random search heuristics**;

KEYWORDS

Recombination Operators; Partition Crossover, Gray-box optimization

ACM Reference Format:
Renato Tinós, Darrell Whitley, Francisco Chicano, and Gabriela Ochoa. 2021. Partition Crossover for Continuous Optimization: ePX. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459296

1 INTRODUCTION

In many problems, information about the internal structure of the problem is available by the inspection of the objective function. However, this useful information is neglected by most transformation operators because optimization is viewed as a black-box process. In contrast, gray-box transformation operators take advantage of the knowledge about the decision variables interaction to guide search [9]. Particularly for recombination, gray-box operators also allow tunneling between local optima [13].

Here, we consider gray-box optimization problems where the objective function $f : \mathbb{D}^N \rightarrow \mathbb{R}$ can be written as:

$$f(\mathbf{x}) = \sum_{l \in \Omega} f_l(\mathbf{x}), \tag{1}$$

where:

- $\mathbf{x} = \{x_1, x_2, \ldots, x_N\} \in \mathbb{D}^N$: candidate solution;
- $\mathbb{D}$: domain for the decision variables $x_i, \forall i \in \{1, \ldots, N\}$;
- Ω is the index set of k-bounded subfunctions, which means that each subfunction accepts at most k variables.
- We will use the notation 1_l to denote an *indicator function* that indicates which variables are passed to subfunction f_l. 1_l can be thought as a bitstring with 1 in the positions corresponding to variables affecting f_l and 0 in the rest.

There are several ways to implement recombination. We will only consider recombination operators that display the **gene transmission** property [8]. In these operators, a variable in the offspring must have an assignment inherited from one of the two parents. Such recombination operators are also **respectful:** if two parents share a common variable assignment, that assignment must also be inherited by the offspring [8].

Partition crossover (PX) is an efficient gray-box recombination operator originally proposed for the traveling salesman problem in [14]. PX for pseudo-Boolean problems, i.e., when $\mathbb{D} = \mathbb{B}$, was proposed in [12]. PX decomposes the *variable interaction graph* (VIG) [13] by removing variables common to both parents. The resulting graph is the *recombination graph* (G_{rec}). The *recombining components*, i.e., the connected components of G_{rec}, determine the variables that are inherited from the same parent. When $\mathbb{D} = \mathbb{B}$, common variables generally occur more often than in other domains. PX can be used in other domains; however, we need to define new ways to identify common variables, or even what that might mean if variable assignments are only similar. For example, when $\mathbb{D} = \mathbb{R}$, it is necessary to define the precision for checking when two variables are considered equal. In addition, PX does not take in account the impact of choosing a variable assignment from one or another parent on subfunctions $f_l(.)$. Consider the following example. Let $\mathbb{D} = \mathbb{B}$, $f_l(1, 1, 1) = 2$, and $f_l(.) = 1$ for all other combinations of $\mathbf{x}$. If the variables influencing $f_l(.)$ take value $(0, 1, 1)$ in one parent and $(0, 0, 0)$ in another parent, it does not

matter whether these variables are inherited from one or the other parent. In PX, the values of $f_l(.)$ are not taken into account for generating the recombination graph.

Here, we propose a new way of generating the recombination graph. We propose to decompose the VIG by removing edges associated with subfunctions $f_l(.)$ that have similar evaluation for combinations of parents' variables. For each $f_l(.)$, the worst combination of parent values for the variables in 1_l is found. If the difference between $f_l(.)$ for the worst combination and the best $f_l(.)$ between the parents is below a threshold, then the edges associated to this $f_l(.)$ are removed from the VIG. By doing so, it is possible to use PX in continuous optimization problems.

The rest of this paper is organized as follows. In Section 2, the original PX is described. The proposed *epsilon-Partition Crossover* (ePX) is described in Section 3. Section 4 presents experiments where ePX is compared to other recombination operators in pseudo-Boolean and continuous optimization. *Differential evolution* (DE) with ePX is presented; to the best of our knowledge, this is the first time a gray-box optimization recombination operator is proposed for DE. Finally, the conclusions are presented in Section 5.

2 BACKGROUND

Let an offspring, $\mathbf{x} \in \mathbb{D}^N$, be generated when two parents, $\mathbf{p} \in \mathbb{D}^N$ and $\mathbf{d} \in \mathbb{D}^N$, are recombined. It will be convenient to think of $\mathbf{p}$ as the **primary** parent and $\mathbf{d}$ as the **donor** parent. We denote with $DP(\mathbf{p}, \mathbf{d})$ the set of possible potential offspring of $\mathbf{p}$ and $\mathbf{d}$ when a recombination operator that transmit alleles is used:

$$DP(\mathbf{p}, \mathbf{d}) = \left\{ \mathbf{x} \in \mathbb{D}^N \middle| \forall i = 1, \ldots, N, x_i = p_i \text{ or } x_i = d_i \right\}. \quad (2)$$

It is very relevant to find ways to generate efficient recombination. Finding the optimal way of recombining solutions for an instance, given two parents, is generally NP-hard [5]. However, information in the parents and the structure of the problem can be used to find efficient ways to recombine solutions [4, 6, 11, 13]. Information about the interaction between decision variables can be explicitly stored in the VIG [13]. The VIG is a undirected graph $G_{VIG} = (V, E)$, where each vertex $v_i \in V$ is related to a decision variable x_i and each edge $e_{i,j} \in E$ indicates that x_i and x_j interact. In gray-box optimization, the VIG can be built by analyzing the objective function before running the optimizer. In black-box optimization, an approximation of the VIG can be built by using *linkage models* [10] or *estimation of distribution algorithms* (EDAs) [9].

In *network crossover* [7], random subgraphs of the VIG define $DP(\mathbf{p}, \mathbf{d})$. The VIG is also used by PX when generating the recombination graph. However, information of the parents is employed for decomposing the VIG. In problems with objective function given by Eq. (1), e.g., MAX-kSAT, $f(\mathbf{x})$ can also be decomposed. Thus, a greedy strategy can be used for deterministically finding the best among 2^q recombination masks, where q is the number of recombining components resulting from the VIG decomposition. As a consequence, the offspring are guaranteed to be piecewise locally optimal when the parents are local optima. In other words, PX allows tunneling between local optima.

In PX, the VIG is decomposed by removing vertices related to decision variables common to both parents $\mathbf{p}$ and $\mathbf{d}$. If $p_i = d_i$, then it does not matter if $x_i = p_i$ or $x_i = d_i$ in the offspring. The connected components of the resulting (recombination) graph determine the variables that should remain together. Two new efficient recombination operators improving PX were recently proposed [3, 4]. Both employ the same idea of removing common vertices to decompose the VIG. However, they enhance the decomposition of the recombination graph by identifying articulation points [3] and using dynamic programming [4].

3 EPSILON-PX (EPX)

Given a parent $\mathbf{p}$ and an offspring $\mathbf{x}$, let's define $1_\mathbf{p}$ as the bit mask with 1 in every position i with $x_i = p_i$. Observe that although $1_\mathbf{p}$ depends also on the offspring $\mathbf{x}$, we omit this dependence to simplify the notation. In order to define the ePX for two parents $\mathbf{p}$ and $\mathbf{d}$, it will be convenient to decompose the subfunctions f_l in Eq. (1) into three different groups.

Definition 3.1. Let us define the following subsets in Ω:

- Let $\Omega_\mathbf{p}$ denote all subfunctions depending only on variables in $1_\mathbf{p}$, that is, the subfunctions that can be evaluated by using variable assignments drawn from parent $\mathbf{p}$.
- Let $\Omega_\mathbf{d}$ denote all subfunctions from $\Omega - \Omega_\mathbf{p}$ (not in $\Omega_\mathbf{p}$) depending only on variables in $1_\mathbf{d}$, that is, subfunctions that can be evaluated by using variable assignments drawn from parent $\mathbf{d}$.
- Let $\Omega_{\mathbf{d},\mathbf{p}}$ the remaining subfunctions: $\Omega - \Omega_\mathbf{p} - \Omega_\mathbf{d}$, that is, subfunctions that must be evaluated by using a combination of assignments of variables variables drawn from $\mathbf{p}$ and $\mathbf{d}$.

The evaluation function for offspring $\mathbf{x}$ can be rewritten as:

$$\begin{aligned} f(\mathbf{x}) &= \sum_{l \in \Omega_\mathbf{p}} f_l(\mathbf{x}) + \sum_{l \in \Omega_\mathbf{d}} f_l(\mathbf{x}) + \sum_{l \in \Omega_{\mathbf{p},\mathbf{d}}} f_l(\mathbf{x}) \\ &= \sum_{l \in \Omega_\mathbf{p}} f_l(\mathbf{p}) + \sum_{l \in \Omega_\mathbf{d}} f_l(\mathbf{d}) + \sum_{l \in \Omega_{\mathbf{p},\mathbf{d}}} f_l(\mathbf{x}). \end{aligned} \quad (3)$$

There may exist a subfunction f_z where the parents $\mathbf{p}$ and $\mathbf{d}$ have the same variable assignments for all of the variables in f_z; for example $f_4(x_2, x_5, x_9)$ and

$$p_2 = d_2 = 0, \qquad p_5 = d_5 = 1, \qquad p_9 = d_9 = 2.125.$$

In this case, we could inherit from either parent $\mathbf{p}$ or parent $\mathbf{d}$. But by Definition 3.1, we exclusively evaluate these subfunctions using the primary parent $\mathbf{p}$. In the following we will assume, without loss of generality, that $\mathbf{p}$ is the best parent, that is, $f(\mathbf{p}) \geq f(\mathbf{d})$. Then in order to find an improved offspring $\mathbf{x}$, some subfunction evaluations must be improved by inheriting a subset of assignments from the donor parent $\mathbf{d}$.

Examples:

We next consider some basic examples of functions. Consider:

$$f(\mathbf{x}) = f_1(x_1, x_2) + f_2(x_3, x_4) + f_3(x_5, x_6).$$

In this case, the subfunctions are nonlinear but separable, so it is always possible to use the following reduced form of $f(\mathbf{x})$:

$$f(\mathbf{x}) = \sum_{z \in \Omega_p} f_z(\mathbf{p}) + \sum_{z \in \Omega_d} f_z(\mathbf{d}). \quad (4)$$

Because the function $f(\mathbf{x})$ is separable, either parents $\mathbf{p}$ or $\mathbf{d}$ could have the best partial solution for each of the three subfunctions. Because the evaluation of the primary parent is better than (or

equal to) the donor parent, the best possible offspring *must* inherit some variable assignments from the primary parent $\mathbf{p}$. If there is no child better than the primary parent, then the best offspring is the same as the primary parent, and we obtain $\Omega_{\mathbf{p}} = \Omega$.

We next consider a function that is not inherently separable:

$$f(\mathbf{x}) = f_1(x_1, x_2, x_3) + f_2(x_3, x_4, x_5) + f_3(x_5, x_6, x_7). \quad (5)$$

If the function in Eq. (5) is pseudo-Boolean, we might find that two parents have bit assignments $p_3 = d_3 = 0$ and $p_5 = d_5 = 1$. By substituting shared assignments for variables, we can temporarily rewrite the function in Eq. (5) as:

$$f(\mathbf{x}) = f_1(x_1, x_2, x_3{=}0) + f_2(x_3{=}0, x_4, x_5{=}1) + f_3(x_5{=}1, x_6, x_7).$$

And by ignoring the assigned variables we could temporarily rewrite the function in Eq. (5) as:

$$f(\mathbf{x}) = f_1'(x_1, x_2) + f_2'(x_4) + f_3'(x_6, x_7), \quad (6)$$

which is again a separable function. We can assume without loss of generality that each subfunction knows about the shared assignments ($x_3 = 0$ and $x_5 = 1$) and computes the appropriate evaluation so that Eq. (5) and (6) return the same evaluations.

This idea that shared common assignments of values to variables can temporarily decompose non-separable functions into separable functions is the key idea behind the standard PX operator. PX first deletes variables with shared (matching) variable assignments. It then temporarily constructs a reduced function $f'(\mathbf{x})$ by deleting the variables with matching assignments. If $f'(\mathbf{x})$ decomposes into q linearly separable functions, PX picks the best partial solution from primary parent $\mathbf{p}$ or donor parent $\mathbf{d}$. Thus, given a decomposition into q linearly separable functions, PX returns the best of 2^q possible offspring using the evaluation function in Eq. (4). Thus, for the standard PX operator, we require that $\Omega_{\mathbf{p},\mathbf{d}} = \emptyset$. In other words, only offspring for which $\Omega_{\mathbf{p},\mathbf{d}} = \emptyset$, are considered. We next look at two other ways that local decomposition can be achieved.

Articulation points. Again assume that the function in Eq. (5) is pseudo-Boolean, but assume there is only 1 shared bit assignment ($p_3 = d_3 = 0$). By substituting shared assignments for variables, we can temporarily rewrite the function in Eq. (5) as:

$$f(\mathbf{x}) = f_1(x_1, x_2, x_3{=}0) + f_2(x_3{=}0, x_4, x_5) + f_3(x_5, x_6, x_7).$$

Since the function is k-bounded, there are only a polynomial number of variable interactions. By constructing the VIG, we can detect that f_2 and f_3 could be separated by assigning a value to variable x_5. Thus, we can temporarily create two versions of $f(\mathbf{x})$:

$$f(\mathbf{x}) = f_1(x_1, x_2, x_3{=}0) + f_2(x_3{=}0, x_4, x_5{=}1) + f_3(x_5{=}1, x_6, x_7),$$

$$f(\mathbf{x}) = f_1(x_1, x_2, x_3{=}0) + f_2(x_3{=}0, x_4, x_5{=}0) + f_3(x_5{=}0, x_6, x_7).$$

We can again evaluate both of these versions of $f(\mathbf{x})$ using the separable function in Eq. (6). However, to find the best offspring we must evaluate the function twice, once with $x_5 = 0$ and once with $x_5 = 1$. This additional work is the cost of making the function $f(\mathbf{x})$ locally separable.

Removing Subfunctions: The main contribution of this paper is also showing that we can remove subfunctions as well to promote local decomposition. Consider again Eq. (5). Assume parent $\mathbf{p}$ = 0000000 and parent $\mathbf{d}$ = 0010110. The shared variables in this case are $x_1 = 0, x_2 = 0, x_4 = 0, x_7 = 0$. In this scenario, there is no

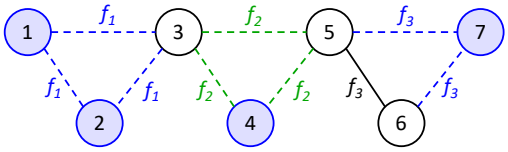

Figure 1: Example for creating the recombination graph.

decomposition because variable x_3 links subfunctions f_1 and f_2, while variable x_5 links subfunctions f_2 and f_3.

However, from a higher level perspective, it is subfunction f_2 that creates the linkage. Thus, removing subfunction f_2 also locally decomposes the function $f(\mathbf{x})$ (see Figure 1). Of course this also has a cost; multiple versions of f_2 must be evaluated:

$f_2(x_3{=}0, x_4{=}0, x_5{=}0)$ identical to parent $\mathbf{p}$

$f_2(x_3{=}0, x_4{=}0, x_5{=}1)$ x_3 from parent $\mathbf{p}$, x_5 from parent $\mathbf{d}$

$f_2(x_3{=}1, x_4{=}0, x_5{=}0)$ x_3 from parent $\mathbf{d}$, x_5 from parent $\mathbf{p}$

$f_2(x_3{=}1, x_4{=}0, x_5{=}1)$ identical to parent $\mathbf{d}$

Two points should be noted: i) removing subfunctions generalizes to continuous variable assignments. We are inheriting from parent $\mathbf{p}$ or parent $\mathbf{d}$, so this is still a binary choice, even for continuous variable assignments; ii) the main challenge we now face is to decide which subfunctions should be removed to best decompose the function and also create the best opportunities for finding improved offspring.

Definition 3.2 (ϵ-close subfunctions). Assume an offspring is obtained from parent $\mathbf{p}$ and $\mathbf{d}$ using a recombination operator that "transmits alleles," where the objective function $f : \mathbb{D}^N \rightarrow \mathbb{R}^{\geq 0}$ is written as Eq. (1). Assume **maximization** and $f_l(.) \geq 0$. Select a constant $\epsilon \in \mathbb{R}$ with $0 \leq \epsilon < 1$. Then we say that subfunction f_l is ϵ-close if for all possible assignments to offspring $\mathbf{x}$:

$$f_l(\mathbf{x}) \geq (1{-}\epsilon) \max\left[f_l(\mathbf{p}), f_l(\mathbf{d})\right] \geq (1{-}\epsilon) f_l(\mathbf{p}), \quad (7)$$

Similarly, for **minimization** we say that f_l is ϵ-close if:

$$f_l(\mathbf{x}) \leq (1{+}\epsilon) \min\left[f_l(\mathbf{p}), f_l(\mathbf{d})\right] \leq (1{+}\epsilon) f_l(\mathbf{d}). \quad (8)$$

We can use the concept of ϵ-close subfunctions as one criterion for deciding which functions can be removed from the VIG. The new partition crossover that explores ϵ-close subfunctions is called *epsilon-PX* (ePX). Of course, we want to remove subfunctions that create as much local decomposition as possible. But we also want to maximize our chances of finding an improving move. By selecting subfunctions that are ϵ-close we also exercise control over the quality of the offspring that can result from decomposition and recombination, as the next theorem proves.

Theorem 3.3. *Let $f : \mathbb{D}^N \rightarrow \mathbb{R}^{\geq 0}$ be an objective function given by Eq. (1), where $f_l(.) \geq 0, \forall l \in \Omega$. For maximization, the evaluation of an offspring, $\mathbf{x}$, generated by recombining two parents, $\mathbf{p}$ (the better) and $\mathbf{d}$, by ePX is bounded by:*

$$f(\mathbf{x}) \geq (1 - \epsilon) f(\mathbf{p}) \quad (9)$$

Proof. ePX decomposes the VIG by removing a subset of subfunctions where $f_l(\mathbf{x}) \geq (1-\epsilon)\max[f_l(\mathbf{p}), f_l(\mathbf{d})]$. The resulting graph is the recombination graph (G_{rec}) and its connected components are the recombining components. All of the vertices in a recombining component are inherited from the same parent. Under ePX the set $\Omega_{\mathbf{p,d}}$ indicates those subfunctions that produce edges in the VIG that connect vertices that are in different recombining components. Recalling Eq. (3):

$$f(\mathbf{x}) = \sum_{l\in\Omega_{\mathbf{p}}} f_l(\mathbf{p}) + \sum_{l\in\Omega_{\mathbf{d}}} f_l(\mathbf{d}) + \sum_{l\in\Omega_{\mathbf{p,d}}} f_l(\mathbf{x}).$$

In ePX, all variables in a recombining component are inherited from the parent with best partial evaluation. Thus for subfunctions $l \in \Omega_{\mathbf{p}}$ we have $f_l(\mathbf{p}) \geq f_l(\mathbf{d})$. Similarly, for subfunctions $l \in \Omega_{\mathbf{d}}$ we have $f_l(\mathbf{d}) \geq f_l(\mathbf{p})$. Let $\Delta \geq 0$ compute the improvement in the evaluation of the subfunctions in $\Omega_{\mathbf{d}}$. We have:

$$\sum_{l\in\Omega_{\mathbf{d}}} f_l(\mathbf{d}) = \sum_{l\in\Omega_{\mathbf{d}}} f_l(\mathbf{p}) + \Delta.$$

Finally, the subfunctions $l \in \Omega_{\mathbf{p,d}}$ are all ϵ-close and:

$$\sum_{l\in\Omega_{\mathbf{p,d}}} f_l(\mathbf{x}) \geq (1-\epsilon) \sum_{l\in\Omega_{\mathbf{p,d}}} f_l(\mathbf{p}).$$

As consequence, for offspring $\mathbf{x}$:

$$f(\mathbf{x}) \geq \sum_{l\in\Omega_{\mathbf{p}}} f_l(\mathbf{p}) + \sum_{l\in\Omega_{\mathbf{d}}} f_l(\mathbf{p}) + \Delta + (1-\epsilon) \sum_{l\in\Omega_{\mathbf{p,d}}} f_l(\mathbf{p})$$

$$f(\mathbf{x}) \geq (1-\epsilon) f(\mathbf{p}) + \Delta \geq (1-\epsilon) f(\mathbf{p}).$$

□

Note that if Δ is sufficiently large, then the offspring is guaranteed to be an improving move. It may be the case that $\sum_{l\in\Omega_{\mathbf{p,d}}} f_l(\mathbf{x})$ also yields an improvement.

3.1 ePX Pseudocode

Algorithm 1 shows the pseudocode for ePX. The maximum $f_l(.)$ for the parents (Eq. (7)) is denoted m_l (Step 11). In Step 12, f_l^u is the minimum value of $f_l(.)$ for all possible offspring $\mathbf{y} \in DP(\mathbf{p}, \mathbf{d})$. The exhaustive search is stopped when a partition with $f_l(.) < (1-\epsilon)m_l$ is found, eventually saving time. In Step 14, the edges representing ϵ-close subfunctions are removed. Subset Ω_h indicates the indices of subfunctions which edges are not removed (Step 16). Like in PX, the common vertices are also removed (Step 6). This is important specially when $\mathbb{D} = \mathbb{B}$.

There are two types of edges linking vertices in a recombining component C. First, there are *internal edges*, i.e., edges linking two vertices $v_i, v_j \in C$. Second, there are *external edges*, i.e., edges linking one vertex $v_i \in C$ to a vertex $v_j \notin C$. In PX, all external edges of a recombining component C are linked to common vertices. In this way, the partial evaluation of C can be computed by the sum of $f_l(.)$ related to internal and external edges.

External edges in ePX are some of the edges removed from the VIG, i.e., ϵ-close edges. External edges can connect vertices in different recombining components, i.e., different connected components of G_{rec}. The partial evaluations for the recombining components are computed in steps 21-25. The partial solutions inside the recombining components are chosen as in PX (steps 26-34). For the j-th recombining component, if $g_j(\mathbf{p}) > g_j(\mathbf{d})$, then $x_i = p_i\ \forall i$ such that vertex $v_i \in C_j$. Otherwise, $x_i = d_i\ \forall i$ such that vertex $v_i \in C_j$. In some cases, there are no internal edges in the recombining component. This is the case where the component has only one vertex; an isolated vertex appears when all its edges are external. A greedy strategy is adopted for setting x_i for isolated vertices. In this strategy, the values of $f_l(.)$ are computed based on the variables that were already set. Steps 35-44 show how this strategy is applied.

Algorithm 1 $(\mathbf{x}, f_x)$=ePX$(\mathbf{p},\mathbf{d},\epsilon)$

1: $f_x = 0$
2: $x_i = \perp \quad \forall i = 1, \ldots, N$
3: $G_{rec} = G_{VIG}$
4: **for** i=1 **to** N **do**
5: **if** $p_i = d_i$ **then**
6: Delete vertex v_i with all its edges from G_{rec}
7: **end if**
8: **end for**
9: $\Omega_h = \emptyset$
10: **for** $\forall l \in \Omega$ **do**
11: $m_l = \max\left[f_l(\mathbf{p}), f_l(\mathbf{d})\right]$
12: $f_l^u = \min_{\mathbf{y}\in DP(\mathbf{p,d})}\left[f_l(\mathbf{y})\right]$
13: **if** $f_l^u \geq (1-\epsilon)m_l$ **then**
14: delete from G_{rec} all edges associated with $f_l(.)$
15: **else**
16: $\Omega_h = \Omega_h \cup \{l\}$
17: **end if**
18: **end for**
19: Find the q connected components of G_{rec}
20: $g_j(\mathbf{p}) = g_j(\mathbf{d}) = 0, \forall j \in \{1, \ldots, q\}$
21: **for** $\forall l \in \Omega_h$ **do**
22: j is the index of the connected component where the vertices associated to f_l belong
23: $g_j(\mathbf{p}) = g_j(\mathbf{p}) + f_l(\mathbf{p})$
24: $g_j(\mathbf{d}) = g_j(\mathbf{d}) + f_l(\mathbf{d})$
25: **end for**
26: **for** j=1 **to** q **do**
27: **if** $g_j(\mathbf{p}) > g_j(\mathbf{d})$ **then**
28: $x_i = p_i\ \forall i$ such that vertex $v_i \in C_j$
29: $f_x = f_x + g_j(\mathbf{p})$
30: **else**
31: $x_i = d_i\ \forall i$ such that vertex $v_i \in C_j$
32: $f_x = f_x + g_j(\mathbf{d})$
33: **end if**
34: **end for**
35: **for** $\forall l \in (\Omega - \Omega_h)$ **do**
36: Define vector $\mathbf{x}^p$ as $x_i^p = x_i$ if $x_i \neq \perp$ and $x_i^p = p_i$ otherwise
37: Define vector $\mathbf{x}^d$ as $x_i^d = x_i$ if $x_i \neq \perp$ and $x_i^d = d_i$ otherwise
38: **if** $f_l(\mathbf{x}^p) > f_l(\mathbf{x}^d)$ **then**
39: $x_i = p_i$ for all variables in 1_l
40: **else**
41: $x_i = d_i$ for all variables in 1_l
42: **end if**
43: $f_x = f_x + f_l(\mathbf{x})$
44: **end for**

Theorem 3.4. *Let $f : \mathbb{D}^N \to \mathbb{R}^{\geq 0}$ be an objective function given by Eq.* (1) *and k be the maximum epistasis degree, i.e, maximum number of variables a subfunction depends on. If $M = |\Omega|$ is $O(N)$, then ePX runs in $O(N2^k)$ time. If k is $O(1)$, then ePX runs in $O(N)$ time.*

Proof. The number of edges of G_{VIG} for a k-bounded function is $O(Mk^2)$. If M is $O(N)$, then doing $G_{rec} = G_{VIG}$ and deleting vertices and edges associated to common variables (steps 3-8 of Algorithm 1) is $O(Nk^2)$, the same complexity of PX. However, computing f_l^u (Step 12) requires in the worst scenario checking all

possible assignment for the variables in f_l using the parents. In this way, computing f_l^u is $O(2^k)$. If M is $O(N)$, then deleting edges associated with ϵ-close subfunctions (steps 10-18) is $O(N2^k)$. Finding the q connected components of G_{rec} is $O(Nk^2)$. Because M and q are $O(N)$, computing the partial evaluations of the recombining components (steps 21-25) and assigning the variables (steps 26-44) are also $O(Nk^2)$. Thus, ePX runs in $O(N2^k)$ time. If k is $O(1)$, then the time complexity is $O(N)$. □

4 EXPERIMENTS

The proposed crossover (ePX)[1] is compared to other recombination operators. Results of experiments with pseudo-Boolean and continuous optimization problems are presented. For each type of problem, two sets of experiments are presented. First, local optima are recombined using different crossover operators and the impact of ϵ in the performance of ePX is investigated. Then, population-based metaheuristics with different crossover operators are compared. The metaheuristics' parameters are defined based on previous works [11, 12] and initial experiments not shown here.

4.1 Experimental Design

Different measures are used to compare the results. The *successful recombination rate* is given by the number of successful recombination events divided by the number of recombination events performed during the experiment. A successful recombination occurs when the offspring is better than both parents. The *worse recombination rate* is given by the number of worse recombination events divided by the number of recombination events performed during the experiment. A worse recombination occurs when the offspring is worse than at least one parent. In PX, the worse recombination rate is zero, but it can be higher than zero in ePX. The *best fitness* is given by the fitness of the best solution found during the experiment. In continuous optimization problems, we used instead the *best fitness error* because the global optimum is known. We also use *time*, given by the runtime (in seconds) for the experiment. A server with 2 processors Intel Xeon E5-2620 v2 (15 MB Cache, 2.10 GHz) and 32 GB of RAM was used. For PX and ePX, the *number of recombining components* is also computed. The mean and standard deviation for these measures are shown. The Wilcoxon signed rank test with $\alpha = 0.05$ is used to statistically compare the results.

4.1.1 NK Landscapes. The pseudo-Boolean benchmark used in the experiments is the NK landscapes model. The objective function is given by Eq. (1), where $\mathbb{D} = \mathbb{B}$, $M = |\Omega| = N$ and $k = K + 1$ for $l = 1, \ldots, M$. Ten instances are generated for each combination of models (random and adjacent) and parameters ($N = \{300, 500\}$ and $K = \{2, 4\}$). The number of runs for each instance is 10, resulting in 100 samples for each configuration. In the first set of experiments for NK landscapes, 50 local optima generated by *local search with first improvement* (LSFI) are recombined. After finding the local optima, they are exhaustively recombined, i.e., each one is recombined with the other 49 local optima. The proposed crossover ePX is compared to: two-point crossover (2X), uniform crossover (UX), multiple 2X (m2X), multiple UX (mUX), and PX. Each time m2X and mUX are applied, the two parents are recombined, respectively by 2X and UX, 10 times, and the offspring with the best evaluation is chosen. In the second set of experiments for NK landscapes, *genetic algorithms* (GAs) with tournament selection, elitism, population restart, bit-flip mutation, and different types of recombination are applied. The GA's parameters are; crossover rate $p_c = 0.6$, mutation rate $p_m = \frac{3}{N}$, population size $\mu = 100$, and size of the tournament selection poll equals to 3. Population restart is implemented by replacing every 50 generations all individuals of the current population, except the best one, by random individuals optimized by LSFI. LSFI is also used for the initial population. Each GA is run for $\frac{Nk}{10}$ seconds.

4.1.2 Continuous Optimization. Four continuous optimization test functions from the CEC'17 Competition on Single Objective Real-Parameter Numerical Optimization [1] are used in the experiments. These functions were chosen because their objective function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ can be written as Eq. (1). The objective function for the i-th test problem is:

$$F_i(\mathbf{x}) = f_i^b\left(\mathbf{M}(\mathbf{x} - \mathbf{o}_i)s_i\right) + F_i^* \tag{10}$$

where $f_i^b(\mathbf{z})$ is the basic function, $\mathbf{M}$ is the rotation matrix, $\mathbf{o}_i$ is the shift vector, s_i is the shrink factor, and F_i^* is the offset. Here, we use the default parameters $\mathbf{o}_i$, s_i, and F_i^* given in the CEC'17 benchmark [1]. The basic functions used here are: Bent Cigar Function (f_1^b); Rosenbrock's Function (f_4^b); Rastrigin's Function (f_5^b); and Modified Schwefel's Function (f_{10}^b). We transformed the CEC'17 benchmark minimization problems into maximization problems.

The basic function f_4^b is non-separable; it can be written as $f_4^b(\mathbf{x}) = \sum_{l=1}^{N-1} f_l(x_l, x_{l+1})$. The basic functions f_1^b, f_5^b, and f_{10}^b are separable, and can be written as $f_i^b(\mathbf{x}) = \sum_{l=1}^{N} f_l(x_l)$. However, by using a rotation matrix, the test function $F_i(\mathbf{x})$ (Eq. (10)) can become non-separable. Here we use $\mathbf{M} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, for F_4. For F_1, F_5, and F_{10}, we use the rotation matrix ($\mathbf{M_1}$) given in the CEC'17 benchmark [1]. The matrix $\mathbf{M_1}$ is composed of submatrices, each one impacting an independent subset of decision variables. Each submatrix is a rotation matrix generated from normally distributed entries by Gram-Schmidt ortho-normalization with condition number equals to 1 or 2. As a result, the VIG is composed by up to 10 connected components (Table 1); each component corresponds to one submatrix[2]. The maximum size of the submatrices corresponds to the maximum epistasis degree $k = \max_{\forall l \in \Omega} |1_l|$. The objective function for the continuous optimization problems used here can be written as Eq. (1), where $\mathbb{D} = \mathbb{R}$, and $M = |\Omega|$ is $O(N)$. Other properties of the test problems are given in Table 1 according to the dimension (N), test function, and rotation matrix.

Table 1: Properties of the continuous problems.

Function	$\mathbf{M}$	N	$k = \max_{\forall l \in \Omega} \lvert 1_l \rvert$	n. connected comp. in G_{VIG}
F_4	$\mathbf{I}$	30,50,100	2	1
F_1,F_5,F_{10}	$\mathbf{M_1}$	30	9	6
		50	10	8
		100	14	10

[1]The source code for ePX is freely available on GitHub (https://github.com/rtinos).

[2]In the CEC'17 competition [1], only black-box algorithms were allowed. However, it is interesting to note that some of the problems can be decomposed in smaller and independent subproblems. Despite being not allowed to explicitly explore the structure of the problem, algorithms, e.g., EDA, could implicitly explore the decomposition of the objective function.

In the first set of experiments for continuous optimization, 50 local optima generated by standard *differential evolution* (DE/rand/1) [15] are recombined. The number of runs for each test problem is 10. The DE parameters are: population size $\mu = 100$, scaling factor $F = 0.8$, and binomial crossover rate $CR = 0.9$. The stopping criterion is best fitness stagnation for $100 + 2N$ generations or number of generations equal to $\frac{50000N}{\mu}$. After running the DE, the found local optima are exhaustively recombined, i.e., each one is recombined with the other 49 local optima. The proposed crossover ePX is compared to: exponential crossover (EX), binomial crossover (BX), multiple EX (mEX), multiple BX (mBX) and PX. EX and BX are respectively similar to 2X and UX used in GAs [15]. Each time mEX and mBX are applied, the two parents are recombined 10 times, respectively by EX and BX, and the offspring with the best evaluation is chosen. In the second set of experiments for continuous optimization, DE with different types of recombination is applied. The DE parameters are the same used in the first set of experiments, except that each DE is run for N seconds. The number of runs is 50.

4.2 Results

This section presents and analyses the results. Additional results and complete tables are presented in the supplementary materials.

4.2.1 NK Landscapes. We analyzed the impact of ϵ in ePX when local optima were recombined[3] in random NK landscapes with $N = 500$ and $K = 2$. We found that when ϵ was increased, the number of recombining components also increased, as expected. As a consequence, the successful recombination rate also increased. However, the mean fitness did not significantly change. Despite improving the successful recombination rate when increasing ϵ, the worse recombination rate also increased. In other words, when increasing ϵ, the average number of offspring better than both parents increased. However, the average number of offspring worse than at least one parent also increased. Less time was required when ϵ increased. This is a result of testing less combinations in Step 12 of Algorithm 1. We set $\epsilon = 0.05$ for the rest of the experiments with NK landscapes.

Tables 2 and 3 present the results of different recombination operators when recombining local optima for NK landscapes[4]. Table 2 shows that ePX found more recombining components than PX. The results show that removing common vertices and edges associated with ϵ-close subfunctions resulted in more recombining components. As a consequence, more potential offspring were explored by ePX than in PX in these experiments.

Finding more recombining components did not necessarily imply finding more successful recombination events when ePX is compared with PX. In PX, all offspring have fitness better than or equal to those of the parents. Table 3 shows that the mean best fitness was significantly better for ePX, when it is compared to all recombination operators, with exception of PX. When compared to PX in the experiments with $N = 300$ and $N = 500$, the mean best fitness was better for ePX in 7 out of 8 experiments. However, we cannot affirm that fitness results for ePX and PX are significantly different, except for one case. The time required by ePX was higher than the time required by 2X and UX. ePX required more time than m2X and mUX in, respectively, 9 and 6 out of 12 experiments; the better results occurred for small epistasis degree ($K = 2$). PX also required less time than ePX, except for two cases.

Table 2: Mean number of recombining components when recombining local optima for NK landscapes. The symbols '=', '+', and '−' respectively indicate that the results of the last column (ePX in this case) is equal, better or worse than the results related to the respective column. The statistical test is used to compare the results. The letter *s* indicates that the differences are statistically significant. The best mean results are in bold.

Model	N	K	PX	ePX
random	300	2	2.2411±0.2574(s+)	**2.2473±0.2556**
		4	1.0012±0.0019(+)	**1.0012±0.0018**
	500	2	2.7026±0.2708(s+)	**2.7172±0.2729**
		4	1.0013±0.0012(+)	**1.0013±0.0012**
adjacent	300	2	22.8852±0.5819(s+)	**22.9176±0.5749**
		4	9.5854±0.1801(s+)	**9.6069±0.1805**
	500	2	37.3933±0.9161(s+)	**37.4822±0.8960**
		4	15.9802±0.2622(s+)	**16.0085±0.2613**

In the experiments recombining local optima with different recombination operators, the number of recombination events is fixed. In the GA experiments, the time is fixed. Because a recombination with ePX generally requires more time than a recombination with 2X, UX, or PX, the mean number of GA generations was smaller for ePX[5]. However, ePX resulted in significantly better fitness than 2X and UX in all the experiments with $N = 300$ and $N = 500$, with 2 exceptions for $K = 4$ (Table 4). The GA with ePX also outperformed m2X and mUX, but in these cases the mean number of generations was higher for ePX. When compared to PX, ePX produced two significantly worse results. The difference in fitness was not statistically different for the other results. In these experiments, the mean number of generations for ePX was smaller than that for PX.

4.2.2 Continuous Optimization. While the difference between the minimum and maximum fitness of the search space is at most 1.0 in NK landscapes, in the test functions for the CEC'2017 benchmark the difference is much bigger. We tested the impact of changing ϵ in ePX when local optima were recombined[6] in problem F_4 with $N = 100$. Again, when ϵ increased, the number of recombining components of the recombination graph increased. As a consequence, the successful and worse recombination rates also increased. However, the mean fitness changed for different values of ϵ. In addition, the time in these experiments did not significantly change when increasing ϵ. In the rest of the experiments, $\epsilon = 0.9$.

The results of different recombination operators when recombining local optima for the continuous optimization problems are

[3]Figure S1 in the supplementary materials shows the number of recombining components, the successful and worse recombination rates, and the time for different values of ϵ.

[4]Tables S1-S4 in the supplementary materials show the complete results.

[5]Table S5 in the supplementary materials shows the number of generations of the GA and Table S6 shows the complete results for the fitness.

[6]Figure S2 in the supplementary materials shows the number of recombining components, the successful and worse recombination rates, and the best fitness error for different values of ϵ.

Table 3: Best fitness in the experiment with recombining local optima for the NK landscapes.

Model	N	K	2X	UX	m2X	mUX	PX	ePX
random	300	2	7.28e-1±1.05e-2(s+)	7.28e-1±1.05e-2(s+)	7.28e-1±1.05e-2(s+)	7.28e-1±1.05e-2(s+)	7.32e-1±1.10e-2(+)	**7.32e-1±1.09e-2**
		4	7.44e-1±5.78e-3(s+)	7.44e-1±5.78e-3(s+)	7.44e-1±5.78e-3(s+)	7.44e-1±5.78e-3(s+)	7.44e-1±5.79e-3(+)	**7.44e-1±5.79e-3**
	500	2	7.27e-1±5.48e-3(s+)	7.27e-1±5.48e-3(s+)	7.27e-1±5.48e-3(s+)	7.27e-1±5.48e-3(s+)	7.30e-1±5.33e-3(+)	**7.30e-1±5.36e-3**
		4	7.39e-1±4.02e-3(s+)	7.39e-1±4.02e-3(s+)	7.39e-1±4.02e-3(s+)	7.39e-1±4.02e-3(s+)	7.39e-1±4.03e-3(+)	**7.39e-1±4.03e-3**
adjacent	300	2	7.24e-1±8.55e-3(s+)	7.20e-1±8.45e-3(s+)	7.27e-1±8.48e-3(s+)	7.20e-1±8.45e-3(s+)	7.38e-1±8.55e-3(s+)	**7.38e-1±8.56e-3**
		4	7.26e-1±6.15e-3(s+)	7.21e-1±6.68e-3(s+)	7.30e-1±6.12e-3(s+)	7.21e-1±6.68e-3(s+)	7.39e-1±6.56e-3(+)	**7.39e-1±6.56e-3**
	500	2	7.21e-1±8.19e-3(s+)	7.17e-1±8.14e-3(s+)	7.23e-1±7.95e-3(s+)	7.17e-1±8.14e-3(s+)	7.36e-1±8.03e-3(+)	**7.36e-1±8.03e-3**
		4	7.20e-1±4.45e-3(s+)	7.15e-1±4.35e-3(s+)	7.23e-1±4.20e-3(s+)	7.15e-1±4.35e-3(s+)	**7.33e-1±4.20e-3**(-)	7.33e-1±4.20e-3

Table 4: Best fitness in the experiment with the GA for the NK landscapes.

Model	N	K	2X	UX	m2X	mUX	PX	ePX
random	300	2	7.40e-1±1.07e-2(s+)	7.43e-1±1.08e-2(s+)	7.41e-1±1.09e-2(s+)	7.47e-1±1.07e-2(s+)	7.47e-1±1.10e-2(+)	**7.48e-1±1.09e-2**
		4	7.58e-1±4.82e-3(s+)	7.62e-1±6.65e-3(-)	7.60e-1±5.51e-3(s+)	**7.66e-1±6.55e-3**(s-)	7.62e-1±5.12e-3(+)	7.62e-1±5.10e-3
	500	2	7.35e-1±5.38e-3(s+)	7.40e-1±5.84e-3(s+)	7.36e-1±5.07e-3(s+)	7.47e-1±5.45e-3(s+)	7.48e-1±5.53e-3(+)	**7.48e-1±5.42e-3**
		4	7.50e-1±3.46e-3(s+)	7.51e-1±3.80e-3(+)	7.50e-1±3.69e-3(s+)	**7.58e-1±4.32e-3**(s-)	7.51e-1±3.28e-3(-)	7.51e-1±3.29e-3
random	300	2	7.49e-1±8.74e-3(s+)	7.40e-1±9.58e-3(s+)	7.49e-1±8.73e-3(=)	7.47e-1±8.80e-3(s+)	**7.49e-1±8.73e-3**(=)	**7.49e-1±8.73e-3**
		4	7.75e-1±7.36e-3(s+)	7.48e-1±8.07e-3(s+)	7.77e-1±7.00e-3(s+)	7.62e-1±7.74e-3(s+)	**7.78e-1±6.90e-3**(s-)	7.78e-1±6.90e-3
	500	2	7.48e-1±8.14e-3(s+)	7.34e-1±8.65e-3(s+)	7.49e-1±8.22e-3(s+)	7.45e-1±8.56e-3(s+)	**7.49e-1±8.22e-3**(s-)	7.49e-1±8.22e-3
		4	7.68e-1±4.54e-3(s+)	7.34e-1±5.41e-3(s+)	7.75e-1±4.48e-3(s+)	7.53e-1±6.14e-3(s+)	**7.78e-1±4.28e-3**(-)	7.78e-1±4.27e-3

presented in Tables 5 and 6[7]. Table 5 compares ePX and PX regarding the number of recombining components found in the recombination graph. In PX and ePX, we define that the i-th vertex is common (for parents $\mathbf{p}$ and $\mathbf{d}$) when $|p_i - d_i| \leq 1.0e-8$. Table 5 shows that the VIG was not decomposed in these experiments for PX. For problem F_4, the number of recombining components found by PX was only one. For the experiments with $\mathbf{M_1}$, the number of recombining components is similar to the number of components in G_{VIG} (Table 1). In other words, PX found more than one recombining component in these problems because G_{VIG} was already fragmented. The difference between the mean number of recombining components and components of G_{VIG} is not zero (e.g., 5.9996 recombining components and 6 components of G_{VIG} for F_1 and $N = 30$) because removing vertices and edges can eventually reduce the number of components in G_{rec}. Thus, the results indicate that removing common vertices was not useful in these problems. For this reason, the results of PX are not presented in the following.

Table 5 shows that ePX produced better results, mainly for F_4 and for higher dimension. For F_4, the epistasis degree is $k = 2$ (Table 1). Finding more recombining components for smaller k was also observed in the experiments with NK landscapes (Table 2). Smaller k implies a less dense G_{VIG}, making it easier to decompose G_{rec}. This is also true when N increases. ePX produced a significantly higher successful recombination rate than all the other recombination operator (EX, BX, mEX, mBX). Finding more recombining components results in exploring more potential offspring. In the experiments with F_4, this was a result of removing edges associated to ϵ-close subfunctions. In the experiments with $\mathbf{M_1}$, this is a result of using the information that the G_{VIG} is already fragmented. When the time is compared, the results of ePX were generally worse. This is specially true when k increases, which can be explained by the time complexity of ePX: while EX and BX are $O(N)$, ePX is $O(N2^k)$.

In the DE experiments, the time is fixed. Because a recombination with ePX generally requires more time than a recombination with EX and BX, the mean number of DE generations for ePX is smaller[8]. However, ePX outperformed the other recombination operators for F_4 and when $\mathbf{M_1}$ was employed, except when compared to mEX; the results for mEX were statistically better in 4 out of 9 experiments with $\mathbf{M_1}$, while the results of ePX were statistically better in 2 experiments. The results are particularly good for F_4, where ePX found the global optima in all runs. When $\mathbf{M_1}$ is used, the worse results of ePX, when compared to mEX, are mainly explained by running the DE for fewer generations because ePX requires more time than EX when k increases. Therefore, a new variant is proposed, where mEX is used in the initial 95% of the running time, and ePX is used in the last 5%. This variant is called *mEX+ePX*. Table 7 shows that DE with mEX+ePX produced similar or better results than DE with mEX in all experiments. It also improves DE with ePX in 7 out of 9 experiments with $\mathbf{M_1}$. However, it presents worse results for function F_4 with $N = 100$.

Table 5: Mean number of recombining components in the experiment with recombining local optima for the continuous optimization problems.

Problem	$\mathbf{M}$	N	PX	ePX
F_4	$\mathbf{I}$	30	1.0000±0.0000(s+)	**3.8441±0.0840**
		50	1.0000±0.0000(s+)	**5.6429±0.1439**
		100	1.0000±0.0000(s+)	**10.8253±0.3574**
F_1	$\mathbf{M_1}$	30	**5.9996±0.0013**(s-)	5.9828±0.0044
		50	**7.9994±0.0019**(s-)	7.9988±0.0021
		100	**9.9993±0.0022**(-)	9.9992±0.0025
F_5		30	**5.9996±0.0013**(s-)	5.8971±0.0170
		50	**7.9994±0.0019**(s-)	7.9936±0.0036
		100	**9.9993±0.0022**(=)	**9.9993±0.0022**
F_{10}		30	**5.9996±0.0013**(s-)	5.9049±0.0210
		50	**7.9994±0.0019**(s-)	7.9910±0.0061
		100	**9.9993±0.0022**(=)	**9.9993±0.0022**

[7]Tables S7-S10 in the supplementary materials show the complete results.

[8]Table S11 in the supplementary materials shows the number of generations of the DE and Table S12 shows the statistical comparison of DE with ePX and the DE with other recombination operators regarding the best fitness error.

Table 6: Best fitness error in the experiment with recombining local optima for the continuous optimization problems.

Problem	M	N	BX	mEX	mBX	ePX
F_4	**I**	30	4.04e+002±2.12e+002(s+)	3.62e+002±1.75e+002(s+)	3.61e+002±1.86e+002(+)	**3.26e+002 ± 1.52e+002**
		50	8.77e+002±6.33e+002(s+)	7.60e+002±4.99e+002(+)	7.53e+002±5.37e+002(+)	**7.11e+002±5.08e+002**
		100	4.25e+003±2.01e+003(s+)	3.99e+003±1.92e+003(s+)	4.09e+003±1.93e+003(s+)	**3.53e+003±1.80e+003**
F_1	$\mathbf{M_1}$	30	3.74e+009±9.10e+008(s+)	3.39e+009±9.46e+008(+)	3.55e+009±1.04e+009(s+)	**3.10e+009±1.00e+009**
		50	1.88e+010±1.51e+010(s+)	1.70e+010±1.34e+010(s+)	1.69e+010±1.40e+010(s+)	**1.47e+010±1.26e+010**
		100	1.73e+010±2.10e+010(s+)	1.70e+010±2.06e+010(+)	1.62e+010±1.96e+010(+)	**1.59e+010±1.90e+010**
F_5		30	2.75e+002±2.58e+001(s+)	2.49e+002±2.74e+001(+)	2.58e+002±2.77e+001(s+)	**2.33e+002±2.22e+001**
		50	5.48e+002±3.40e+001(s+)	5.33e+002±3.85e+001(s+)	5.35e+002±2.96e+001(s+)	**4.75e+002±3.15e+001**
		100	1.25e+003±7.13e+001(s+)	1.18e+003±8.84e+001(s+)	1.19e+003±9.01e+001(s+)	**1.11e+003±6.78e+001**
F_{10}		30	7.35e+003±2.56e+002(s+)	7.13e+003±1.87e+002(s+)	7.00e+003±2.17e+002(s+)	**5.71e+003±3.23e+002**
		50	1.39e+004±2.97e+002(s+)	1.34e+004±3.85e+002(s+)	1.35e+004±2.57e+002(s+)	**1.13e+004±2.99e+002**
		100	3.19e+004±3.80e+002(s+)	3.09e+004±3.82e+002(s+)	3.09e+004±4.36e+002(s+)	**2.72e+004±3.16e+002**

Table 7: Best fitness error in the experiment with the DE for the continuous optimization problems.

Function	M	N	DE with BX	DE with mEX	DE with mBX	DE with ePX	DE with mEX+ePX
F_4	**I**	30	2.05e-006±1.68e-006(s+)	**0.00e+000±0.00e+000**(=)	9.84e-001±4.81e-001(s+)	**0.00e+000±0.00e+000**(=)	**0.00e+000±0.00e+000**
		50	3.81e+001±1.06e+000(s+)	**0.00e+000±0.00e+000**(=)	8.91e+001±2.06e+001(s+)	**0.00e+000±0.00e+000**(=)	**0.00e+000±0.00e+000**
		100	5.47e+002±1.17e+002(s+)	9.19e+001±5.41e-001(s+)	6.27e+003±2.00e+003(s+)	**0.00e+000±0.00e+000**(s-)	7.83e+001±5.64e-001
F_1	$\mathbf{M_1}$	30	3.17e-006±4.17e-006(s+)	**0.00e+000±0.00e+000**(=)	2.48e+002±1.92e+002(s+)	3.02e-005±2.49e-005(s+)	**0.00e+000±0.00e+000**
		50	1.40e+004±1.31e+004(s+)	1.37e+003±6.50e+002(s+)	3.76e+008±1.64e+008(s+)	**8.94e+001±5.68e+001**(s-)	1.69e+002±8.14e+001
		100	1.17e+009±4.12e+008(s+)	2.01e+008±3.93e+007(s+)	6.81e+010±1.70e+010(s+)	2.79e+010±3.49e+009(s+)	**1.24e+008±2.47e+007**
F_5		30	1.38e+002±6.26e+001(s+)	5.46e+001±6.46e+000(s+)	1.89e+002±2.60e+001(s+)	5.43e+001±8.95e+000(+)	**5.05e+001±5.77e+000**
		50	3.96e+002±2.12e+001(s+)	2.31e+002±1.47e+001(s+)	4.27e+002±2.05e+001(s+)	**1.76e+002±1.15e+001**(s-)	2.20e+002±1.35e+001
		100	1.00e+003±2.90e+001(s+)	1.00e+003±3.63e+001(s+)	1.20e+003±5.47e+001(s+)	1.12e+003±2.41e+001(s+)	**9.79e+002±3.21e+001**
F_{10}		30	8.15e+003±5.05e+002(s+)	4.12e+003±2.03e+002(s+)	7.88e+003±5.05e+002(s+)	4.22e+003±3.12e+002(s+)	**3.97e+003±1.87e+002**
		50	1.52e+004±3.99e+002(s+)	9.10e+003±3.83e+002(s+)	1.49e+004±3.72e+002(s+)	9.15e+003±3.06e+002(s+)	**8.94e+003±3.53e+002**
		100	3.33e+004±5.10e+002(s+)	2.52e+004±4.90e+002(s+)	3.30e+004±4.85e+002(s+)	2.81e+004±4.56e+002(s+)	**2.51e+004±5.16e+002**

5 CONCLUSIONS

In PX, the variable interaction graph (VIG) is decomposed by removing common vertices. The connected components of the resulting graph (G_{rec}) indicate the variables that should be inherited together after recombination. In the proposed ePX, the VIG is decomposed by removing edges associated with ϵ-close subfunctions (Definition 3.2). Common vertices are also removed. Theoretical results show that the evaluation of an offspring generated by recombining parents $\mathbf{p}$ and $\mathbf{d}$, respectively, with evaluation $f(\mathbf{p})$ and $f(\mathbf{d})$, is always better than or equal to $(1-\epsilon)\max\big(f(\mathbf{p}), f(\mathbf{d})\big)$, where $\epsilon \in \mathbb{R}$ is a parameter with $0 \leq \epsilon < 1$ (Theorem 3.3).

By removing edges associated with ϵ-close subfunctions, the partial evaluations of $f_l(.)$ are taken into account when decomposing the VIG. This enables the use of partition crossover in continuous optimization. Here, ePX was compared to other recombination operators in one pseudo-Boolean problem class (NK landscapes) and four continuous optimization problems. Results of experiments where local optima are recombined indicate that more recombining components are found by ePX than when only common vertices are removed (in PX). When ePX is compared against other recombination operators in GA and DE, better performance was obtained in instances where the maximum epistasis degree (k) is low.

The *articulation points PX* (APX) [3] and the *dynastic potential crossover* (DPX) [4] enhance PX by breaking the connected components of G_{rec}. More connected components result in the exploration of more potential offspring. However, APX and DPX also decompose initially the VIG by removing common vertices. Thus, a future work is to enhance APX and DPX by removing edges associated to ϵ-close subfunctions. A disadvantage of ePX is that its time complexity is $O(N2^k)$, while PX is $O(Nk^2)$. Investigating ways of reducing the time complexity of ePX is another avenue for future work. The GA and DE with ePX presented here are not intended to establish new best known algorithms for pseudo-Boolean or continuous optimization problems. However, we believe that ePX can be incorporated into efficient algorithms, including some specially developed for gray-box optimization [2, 4], in order to create state-of-art algorithms. A simple strategy is to use ePX to recombine solutions produced by an efficient algorithm, e.g., CMA-ES, or solutions produced by different algorithms in gray-box optimization problems. Finally, adapting ePX for gray-box optimization problems with constraints is another relevant future work.

ACKNOWLEDGMENTS

This work was partially supported in Brazil by São Paulo Research Foundation (FAPESP), under grants 2013/07375-0 and 2019/07665-4, and National Council for Scientific and Technological Development (CNPq), under grant 305755/2018-8. It was partially funded in Spain by Universidad de Málaga, Consejería de Economía y Conocimiento de la Junta de Andaluía and FEDER, under grant UMA18-FEDERJA-003 (PRECOG), and the Spanish Ministry of Science, Innovation and Universities and FEDER under contracts RTC-2017-6714-5 (Eco-IoT).

REFERENCES

[1] N. H. Awad, M. Z. Ali, P. N. Suganthan, J. J. Liang, and B. Y. Qu. 2016. *Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization.* Technical Report. Nanyang Technological University, Jordan University of Science and Technology, Zhengzhou University.

[2] A. Bouter, S. C. Maree, T. Alderliesten, and P. A. N. Bosman. 2020. Leveraging conditional linkage models in gray-box optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proc. of GECCO'2020.* 603–611.

[3] F. Chicano, G. Ochoa, D. Whitley, and R. Tinós. 2018. Enhancing Partition Crossover with Articulation Points Analysis. In *Proc. of GECCO'2018.* 269–276.

[4] F. Chicano, G. Ochoa, D. Whitley, and R. Tinós. 2019. Quasi-Optimal Recombination Operator. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar).* Springer, 131–146.

[5] A. V. Eremeev and J. V. Kovalenko. 2014. Optimal recombination in genetic algorithms for combinatorial optimization problems: Part II. *Yugoslav Journal of Operations Research* 24, 2 (2014), 165–186.

[6] A. V. Eremeev and Y. V. Kovalenko. 2017. Genetic algorithm with optimal recombination for the asymmetric travelling salesman problem. In *Int. Conf. on Large-Scale Scientific Computing.* 341–349.

[7] M. W. Hauschild and M. Pelikan. 2010. Network crossover performance on NK landscapes and deceptive problems. In *Proc. of GECCO'2010.* 713–720.

[8] N. J. Radcliffe. 1994. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence* 10, 4 (1994), 339–384.

[9] R. Santana. 2017. Gray-box optimization and factorized distribution algorithms: where two worlds collide. *arXiv preprint arXiv:1707.03093* (2017).

[10] D. Thierens. 2010. The Linkage Tree Genetic Algorithm. In *Parallel Problem Solving from Nature, PPSN XI*, R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 264–273.

[11] R. Tinós. 2020. Artificial neural network based crossover for evolutionary algorithms. *Applied Soft Computing* 95 (2020), 106512.

[12] R. Tinós, D. Whitley, and F. Chicano. 2015. Partition crossover for pseudo-boolean optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII.* 137–149.

[13] D. Whitley. 2019. Next generation genetic algorithms: a user's guide and tutorial. In *Handbook of Metaheuristics.* Springer, 245–274.

[14] D. Whitley, D. Hains, and A. Howe. 2009. Tunneling between optima: partition crossover for the traveling salesman problem. In *Proc. of GECCO'2009.* 915–922.

[15] D. Zaharie. 2009. Influence of crossover on the behavior of differential evolution algorithms. *Applied Soft Computing* 9, 3 (2009), 1126–1138.