IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 2, 2021, accepted October 2, 2021, date of publication October 4, 2021, date of current version October 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3117987

Ring-Based Crossovers in Genetic Algorithms: Characteristic Decomposition and Their Generalization

SUNISA RIMCHAROEN AND NUTTHANON LEELATHAKUL
Faculty of Informatics, Burapha University, Saen Suk 20131, Thailand

Corresponding author: Nutthanon Leelathakul (nutthanon@buu.ac.th)

This work was supported by the Faculty of Informatics, Burapha University.

ABSTRACT Effective crossover methods are essential for genetic algorithms because they enhance population diversity, resulting in an increase in the search space to be explored and the mitigation of trapping at local optima. Ring-based crossover methods are designed to address the shortcomings of traditional crossover techniques; for example, the ends of binary-encoded chromosomes tend to remain unaltered, even when their fitness values are low. By conjoining the leftmost and rightmost parts of a chromosome to form a ring, the ring-based crossovers allow offspring bits to be inherited from parent bits at the front and rear positions. Such a ring-based technique helps increase population diversity while requiring a relatively low number of fitness evaluations. To the best of our knowledge, there has been no comparative study or comprehensive analysis of ring-based crossover techniques. In this paper, we study and compare the characteristics of various existing ring-based crossover techniques (i.e., circle-ring (CRC), annular (AC), ring (RC) and front-rear crossover (FRC)). Although they each have distinct characteristics, they share a common crossover concept: bits at different positions can be swapped. We propose a generalized ring-based crossover (GRC) as an umbrella of the ring-based crossovers, embodying all of their beneficial characteristics. Chromosome shifting, ring forming, chromosome exchange and offspring creation are integrated into the proposed model. The experimental results show that GRC outperforms the other ring-based crossover methods in all experiments, and these results are consistent with the results of behavioral analysis. In a trap problem, GRC outperformed the state-of-the-art BOA method and require 40-times fewer fitness evaluations. GRC tends to maintain a variety of candidates in populations and to preserve the building blocks of solution, and it requires 93% and 57% fewer fitness evaluations (on average) compared to traditional crossover and other ring-based crossover methods.

INDEX TERMS Building block, generalized crossover, genetic algorithm, ring representation.

I. INTRODUCTION

Evolution is the most essential process for living organisms to survive. They adapt or evolve to fit in ever-changing environments and/or to compete with other species. In the reproduction process, genetic operators (such as chromosome crossover and mutation) make natural evolution possible. Naturally, creatures survive and evolve by inheriting superior genomes from their ancestors, crossing over/mutating the genomes, and passing the genomes down to subsequent generations. The crossover and mutation operators can improve the survival of even the fittest species and allow species to continually evolve. Genetic algorithms (GAs) adopt natural evolutionary processes and employ both crossover and mutation for offspring generations. In [1], the author (who proposed the first GA) wrote that the crossover operator helps GAs to explore (or jump through) the search space, and mutation is merely ''a background operator''. There have been several studies in the literature confirming the usefulness of the crossover operator. Jansen and Wegener [2] verified that the use of the crossover could decrease the expected run time of a model. Doerr *et al.* [3] showed that the crossover operator was useful for solving the classical all-pair shortest path problem. Sudholt [4] demonstrated that this operator could accelerate building-block assembly. Pinto and Doerr [5] presented a mathematical proof and empirical evaluations of the usefulness of crossover in black-block optimizations.

There are various crossover methods in the literature. Pavai and Geetha [6] classified more than one hundred crossover methods into two broad categories: 1) methods for the

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

representation of applications and 2) methods for improving GA performance. For the former category, the methods were designed based on chromosome representations, e.g., binary encoding, integer encoding, and permutation encoding. For the latter, the methods (such as adaptive crossover and multiparent crossover) were designed with the aim of achieving optimal solutions. Gwiazda [7] also provided a comprehensive review of crossover techniques in his reference book; he described 11 standard, 66 binary-coded, 89 real-coded and 9 statistic-based crossover operators.

At present, research on crossover techniques is ongoing to address problems in various GA domains. For example, Manzoni *et al.* [8] investigated the effect of balanced crossover operators in combinatorial optimization problems. Ali *et al.* [9] introduced four new crossover operators to solve a dynamic job scheduling problem. Koohestani [10] presented a new version of partially mapped crossover (PMX) to improve the efficiency of permutation-based GA. Pan *et al.* [11] proposed a modification of simulated binary crossover (SBX) to improve the performance of multiobjective evolutionary algorithms (MOEAs) in problems involving rotated Pareto sets.

Among the various crossover techniques in the literature, the crossover methods designed for binary-encoded representations are general-purpose and classic methods; binary encoding is the most common form and simplest to implement. A unique group of crossover methods that transform linear binary-encoded chromosomes into circular chromosomes (i.e., forming rings) exists [12], [13]–[15]. This kind of technique promotes the diversity of the chromosomes and prevents the algorithm from being trapped at local optima. For example, the front-rear crossover (FRC [12]), of which simple operation can yield promising results in solving the trap problem, which is known as one of the hard benchmark problems. One state-of-the-art algorithm that can efficiently solve the trap problem is the Bayesian optimization algorithm (BOA) [16]. However, the BOA requires a long computational time to construct and evolve a Bayesian network model, which is used to estimate the joint probability distribution of all promising candidate bit pairs. Surprisingly, FRC could solve the trap problem and require 40-times fewer fitness evaluations than the BOA, in which the trap size is 180 bits. Pavez-Lazo and Soto-Cartes [14] also noted the disadvantage of the linear-chromosome representation: genetic information at the ends of the chromosomes tends to remain unaltered, even after multiple crossovers; they addressed this disadvantage by employing annular crossover to form a ring representation. They reported that the method provides faster convergence and better solutions than traditional crossover techniques. Kaya *et al.* [15] showed that ring and traditional crossover methods provide significant differences in terms of variety in the population; notably, ring crossover tends to generate more genetically variant children than does traditional crossover.

According to the studies mentioned above, ring-based crossover techniques seem promising. Nevertheless, to the best of our knowledge, there have been no comprehensive studies or comparative analyses of the behavior of ring-based crossover techniques. In this paper, we study and compare the existing ring-based crossover methods (i.e., circle-ring (CRC), annular (AC), ring (RC) and front-rear crossover (FRC)), all of which share a common crossover concept: transforming the chromosome representations from linear to circular. We then propose a generalized ring-based crossover (GRC) method that incorporates the advantages of various techniques (e.g., chromosome shifting, ring forming, chromosome exchange and offspring creation).

Our contributions are outlined as follows.

1) A new crossover technique called GRC is proposed; it is able to capture and embody the characteristics of the previous ring-based crossover methods in one framework.
2) The characteristics of the previous ring-based crossover methods are studied. The underlying components of the crossover techniques are decomposed to identify the features commonly shared among the techniques.
3) The steps in RC are transformed to create offspring by forming two rings, making RC generalizable while retaining the same functionality. In particular, CRC, AC and FRC include crossover steps that are easily generalized. In contrast, RC generates offspring differently by forming only one ring (instead of two rings, as in the other methods), thus requiring transformation.
4) Performance and behavioral analyses of all ring-based crossover methods are conducted, and these methods are compared with traditional crossover. The analysis results are used to explain why ring-based crossover methods are promising.
5) The benefits and limitations of ring-based crossover techniques are outlined.

The rest of this paper is organized as follows. Section II introduces the background on the genetic algorithms, crossover methods and benchmark problems addressed in the experiments. Section III explains the proposed generalized crossover approach. Section IV presents the comparative results obtained from experiments. Section V discusses and analyzes the behaviors and advantages of ring-based crossover methods. The conclusions are drawn in Section VI.

II. BACKGROUND

A. GENETIC ALGORITHMS

Genetic algorithms (GAs) are known as search and optimization techniques inspired by the principle of natural evolution. The basic processes of GAs are population representation, population initialization, fitness evaluation, selection, crossover, mutation, and termination. Solutions to particular problems are represented by chromosomes (which are sequences of genes). The initialization process is a step to create the first pool of candidate solutions (called a population). To determine how close candidate solutions are to the desired solution, the fitness function is designed specifically for each problem; it takes a candidate solution as the input and

returns a fitness value as a score, used to guide the algorithms to search for targets. The fitter a candidate is, the greater the chance they are selected to become parents of the next generation. Crossover creates offspring by conjoining the parents' gene segments, allowing the gene properties to be passed from generation to generation. Some modifications to one or more genes are performed using a mutation operator. These processes are repeated until the termination condition is met.

The success of applying GAs depends on multiple factors, and the design of crossover is one of them. The significance of each crossover method lies in its capability to help GAs explore a wide area of the search space [1], thus improving GA performance [17]. Over time, various crossover methods have been developed; some are designed for general purposes, and some are designed for certain problems. In this section, we begin explaining traditional crossover, including single-point, two-point and uniform crossover methods, followed by the crossover methods that are the focus of this paper, i.e., unique crossover methods that transform linear chromosomes into circular chromosomes before crossover is performed.

B. TRADITIONAL CROSSOVER TECHNIQUES

1) SINGLE-POINT CROSSOVER (SPC)

Single-point crossover (also called one-point crossover) [1] is one of the earliest forms of crossover. One cut point on each parent's chromosome is randomly selected. The parents' bitstrings at the cut point are swapped to form offspring, as shown in Fig. 1.

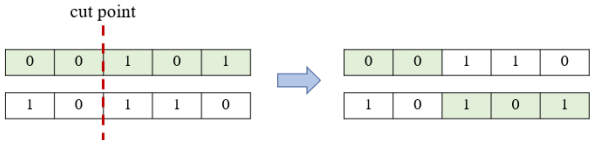


FIGURE 1. Single-point crossover.

2) TWO-POINT CROSSOVER (TPC)

Two-point crossover [18] involves the random selection of two cut points on the parents' chromosomes. The offspring are produced by exchanging the middle segments of the parents' chromosomes, as illustrated in Fig. 2.

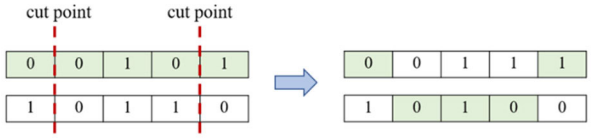


FIGURE 2. Two-point crossover.

3) UNIFORM CROSSOVER (UC)

Uniform crossover [19] involves determining whether the parents' bits at the same position should be swapped while passing them to the offspring. In other words, with equal chance, each bit in one child's chromosome is inherited from one parent, while that in the other child's from the other, as shown in Fig. 3.

FIGURE 3. Uniform crossover.

C. RING-BASED CROSSOVER TECHNIQUES

1) CIRCLE-RING CROSSOVER (CRC)

Circle-ring crossover was proposed by Zhang *et al.* [13] and involves joining both ends of each parent's chromosomes to form a ring. The chromosome-mixing method starts by rotating both rings by an arbitrary angle. In other words, CRC performs an n-bit cyclic shift. The cut point is always at the middle of the bitstrings. The bits following the cut point are swapped between the two parents to form offspring. As shown in Fig. 4, 3-bit cyclic shifting is performed on both parents' chromosomes. The cut point is between the second ($\lfloor 5/2 \rfloor = 2$) and third bits.

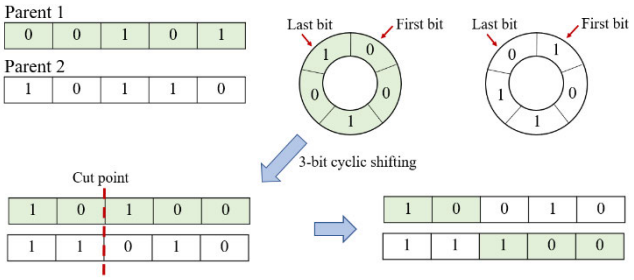


FIGURE 4. Circle-ring crossover.

2) ANNULAR CROSSOVER (AC)

Annular crossover was proposed by Pavez-Lazo and Soto-Cartes [14] and involves transforming parents' linear chromosomes into rings. Both ends of a chromosome are conjoined. Let l be the length of a semiring (or a ring segment), which is uniformly chosen in the range of 1 to $L/2$, where L is the chromosome length (in AC, the lengths of both parents' semirings are equal.) Then, one cut point is randomly selected in the range of 1 to L for each parent. (The parents' cut points can be different.) The semirings of length l from both parents (starting from the cut points) are swapped. In Fig. 5, we illustrate the 3-bit-length semirings. The cut points are the second and fourth positions associated with the first and second parents' chromosomes, respectively.

3) RING CROSSOVER (RC)

Ring crossover, proposed by Kaya *et al.* [15], combines pairs of parent chromosomes into rings by conjoining the genes in head-to-head and tail-to-tail arrangements. The cut point is

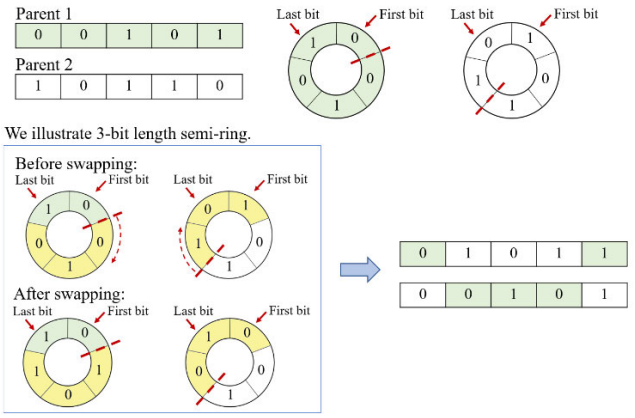


FIGURE 5. Annular crossover.

randomly selected at the position of each ring to separate it into halves.

The first offspring's chromosome starts from the bit at the cut point and proceeds to the next bits in the clockwise direction. The other's chromosome consists of the bits in the other half of the ring, starting from the bit at the cut point and proceeding to those in the counterclockwise direction. The example is shown in Fig. 6.

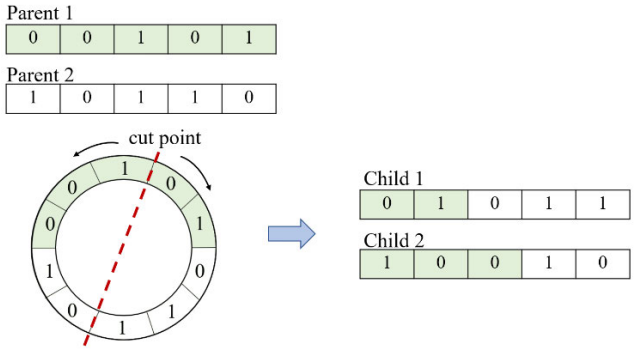


FIGURE 6. Ring crossover.

Note that RC forms a ring in a different fashion than the other ring-based crossover methods. The other ring-based crossover methods convert one linear chromosome into one ring. Generating an offspring requires two parents' chromosomes and hence two rings. In contrast, RC joins two chromosomes to form only one ring. Therefore, before we can generalize RC, we have to transform (or reduce) the ring formation to the common form (i.e., one ring representing only one chromosome), which is a shared characteristic in the proposed generalized method. Fig. 7 illustrates the reduced form of RC. Reduced RC starts with randomly determining the cut point of the first parent (the i^{th} position). The second parent's cut point is then at the $(L\text{-}i)^{th}$ position among chromosomes, where L is the chromosome length. Next, the parents' chromosomes are converted into two rings, and the associated semirings are determined. For the first ring, the bits starting from the first bit to the cut point are reversed.

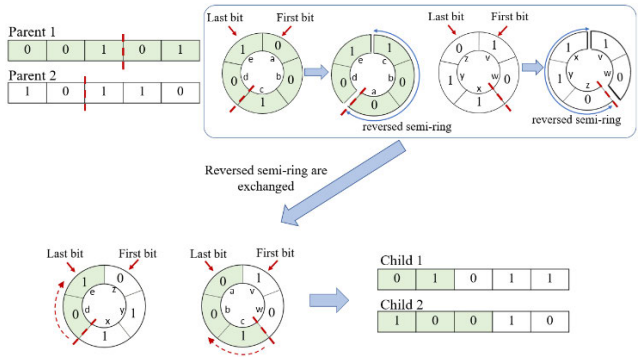


FIGURE 7. Reduced form of ring crossover.

For the second ring, the bits starting from the cut point to the last bit are reversed, as illustrated in Fig 7. The two reversed semirings are then exchanged. The rings (starting from the ring cut points) are converted back into the linear chromosomes of offspring and are the same as those obtained from nonreduced RC, as shown in Fig. 6.

4) FRONT-REAR CROSSOVER (FRC)

Front-rear crossover was proposed by Pumsuwan *et al.* [12]. During this process, FRC produces each offspring by continuously selecting two parents and swapping the first parent's front chromosome with the second parent's rear chromosome. Given the chromosome length L, the segment length (l) is uniformly randomized in the range of (0, L). The cut points for the first and second parents are at positions l and L-l of the chromosomes, respectively. Each of the offspring is created by swapping the first parent's $1^{st} - l^{th}$ bits with the second parent's $(L - l + 1)^{th} - L^{th}$ bits. Fig. 8 illustrates an example of FRC.

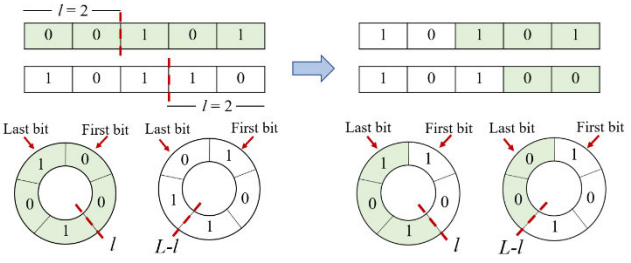


FIGURE 8. Front-rear crossover.

D. BENCHMARK PROBLEMS

The crossover techniques stated in the previous subsection and the proposed generalized ring-based crossover method are evaluated based on the benchmark problems described below. We selected four benchmark problems because they epitomize problems in which chromosomes are binary coded and fitness functions involve unitation. Furthermore, these problems are commonly used to compare and analyze the

behaviors of algorithms in composing building blocks considering their ease of identification and inspection.

1) ONE-MAX AND ZERO-MAX PROBLEMS

One-max and zero-max problems are simple bit counting problems designed for assessing Gas. We selected these problems to ensure that the crossover techniques are unbiased toward all-one-bit and all-zero-bit candidates. The candidates' fitness values in one-max and zero-max problems are calculated by counting the numbers of one and zero bits, respectively. Given the length L of a chromosome, the optimal solutions of the one-max and zero-max problems are the chromosomes for which all L bits are ones and zeros, respectively. Although the problems usually require unimodal fitness functions that are relatively simple, Culberson [20] noted that using the single-point crossover could cause the algorithm to be stuck at a local optimum. Notably, the number of local optima could increase exponentially in the one-max problem as the chromosome length increases.

2) ROYAL ROAD PROBLEM

The royal road problem [21] is designed for benchmarking GAs, especially in terms of their capability to compose building blocks (i.e., short low-order schemas that are a part of the optimal solution [22]). A schema is represented by a set of characters $\{b_i | b_i \in \{0, 1, *\}$ and $1 \leq i \leq L\}$. For example, the schema '$***111$' describes a set of $2^3 = 8$ bitstrings, and the first three bits of each can be either 0 or 1; last three bits are always '111'. The royal road fitness function is defined as

$$F(x) = \sum_{s_i \in S} c_{s_i} \sigma_{s_i}(x), \tag{1}$$

where x is a bitstring, S is the set of all schemas, c_{s_i} is a value of the i^{th} schema, and σ_{s_i} is defined as shown in Eq. (2).

$$\sigma_{s_i} = \begin{cases} 1, & \text{if } x \text{ follows the } i^{th} \text{ schema} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

For the royal road problem, the set of schemas $S = \{s_1, \ldots, s_{15}\}$ is defined as shown in Table 1.

TABLE 1. The royal road schemas.

Schema	Value
s_1 = 11111111**	$c_1 = 8$
s_2 = ********11111111**	$c_2 = 8$
s_3 = ****************11111111**	$c_3 = 8$
s_4 = ************************11111111**	$c_4 = 8$
s_5 = ********************************11111111************************************	$c_5 = 8$
s_6 = **11111111********************************	$c_6 = 8$
s_7 = **11111111********************	$c_7 = 8$
s_8 = **11111111************	$c_8 = 8$
s_9 = 1111111111111111**	$c_9 = 16$
s_{10} = ****************1111111111111111**	$c_{10} = 16$
s_{11} = ********************************1111111111111111**************************	$c_{11} = 16$
s_{12} = **1111111111111111**********	$c_{12} = 16$
s_{13} = 11111111111111111111111111111111**************************************	$c_{13} = 32$
s_{14} = ********************************11111111111111111111111111111111**********	$c_{14} = 32$
s_{15} = 1111111111111111111111111111111111111111111111111111111111111111	$c_{15} = 64$

3) HIERARCHICAL IF-AND-ONLY-IF PROBLEM

The hierarchical if-and-only-if (H-IFF) problem was proposed by Watson *et al.* [23]. The evaluation structure in the H-IFF problem is a binary tree, in which each node stores one character (either '0', '1', or '−'). Fitness value determination is based on all the node points at each level of the tree. At the lowest level, all leaf nodes, with characters corresponding to '0' or '1' in the bitstring, are assigned points of 1. A parent node stores '0' or '1' if both of its children store '0' or '1', respectively; otherwise, the parent stores '−' (or a *NULL* value). The H-IFF fitness function is defined as

$$H - IFF(x) = \sum_{h=0}^{H} \sum_{i=1}^{2^{(H-h)}} c_i^h, \tag{3}$$

where H is the height of the binary tree, h denotes the tree level, and c_i^h is the point associated with the i^{th} node at the level of h defined in Eq. (4).

$$c_i^h = \begin{cases} 2^i, & \text{if the } i^{th} \text{ node stores 0 or 1} \\ 0, & \text{if the } i^{th} \text{ node stores '}-\text{'} \end{cases} \tag{4}$$

Fig. 9 shows an example. The bitstring '00011111' has a fitness value of 18. The optimal solution of this problem is either all zeroes or all ones (with '11' or '00' as the building block).

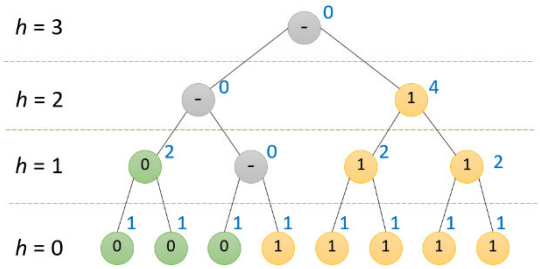


FIGURE 9. Fitness value calculation for an H-IFF problem.

4) TRAP PROBLEM

The trap problem [24]–[28] is one of the most difficult problems considered in GA benchmarking; it was designed to delude gradient-based optimizers into favoring zero bits, but the optimal solution is actually all one bits. In the k-bit trap problem [29], each block score is defined by the trap function as follows:

$$f_k(b_0 \ldots b_{k-1}) = \begin{cases} v_{\text{high}}, & \text{if } u = k \\ v_{\text{low}} - u\dfrac{v_{\text{low}}}{k-1}, & \text{otherwise,} \end{cases} \tag{5}$$

where b_i is the i^{th} gene (or the i^{th} bit) in a chromosome where $b_i \in \{0, 1\}$, k is the trap size (or block size), u is the summation of bit values in the block, where $u = \sum_{i=0}^{k-1} b_i$, and v_{high} and v_{low} are the scores of the k-bit block when all bits are ones and zeroes, respectively. Usually, v_{high} is set to k, and v_{low} is set to k-1. For example, if the trap size, v_{high} and v_{low}

are 3, 3, and 2, the scores of the '111' ($u = 3$), '000' ($u = 0$), '001' ($u = 1$) and '011' ($u = 2$) blocks are 3, 2, 1 and 0, respectively. Fig. 10 illustrates the value of $f_k(b_0 \dots b_{k-1})$ as the number of bits in the block increases.

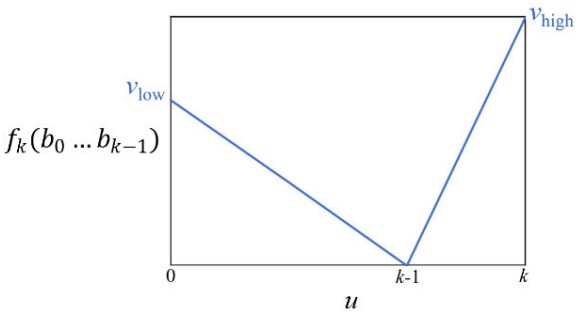


FIGURE 10. Trap function.

To increase the difficulty of solving a k-bit trap problem, m blocks of k bits are formed as additive blocks ($B_0, B_1 \dots$, and B_{m-1}). The corresponding fitness function is defined as follows:

$$F_{k\times m}(B_0 \dots B_{m-1}) = \sum_{i=0}^{m-1} f_k(B_i), \quad B_i \in \{0,1\}^k \tag{6}$$

5) HIERARCHICAL TRAP PROBLEM

The hierarchical trap problem (H-Trap) was proposed by Pelikan and Goldberg [30]. The optimal solution is an all-one bitstring. The H-Trap structure is a balanced k-ary tree, where $k \geq 3$ (throughout this paper, we set k to 3.) Similar to the H-IFF structure, in the H-Trap structure, each node stores one character (either '0', '1', or '−'). If all three children store '0', the parent node stores '0'. Similarly, if all three children store '1', the parent stores '1'. Otherwise, the parent stores '−'. Let h be the height of nodes. At the root level, $v_{high}^{h=H}$ and $v_{low}^{h=H}$ are 1 and 0.9, respectively, where H is the highest tree height. At the lower tree levels, both values of $v_{high}^{h\neq 1,H}$ and $v_{low}^{h\neq 1,H}$ are 1, where $1 < h < H$. The point of the i^{th} nonleaf node (at tree level h) is determined according to Eq. (7) and (8), where b_0, b_1 and b_2 are the characters stored by the child nodes. The fitness value is determined by considering all values at nonleaf node points based on the H-Trap fitness function in Eq. (9).

$$f_k^h(b_0 \dots b_{k-1}) = \begin{cases} v_{high}^h, & \text{if } u = k \\ v_{low}^h - u\dfrac{v_{low}^h}{k-1}, & \text{otherwise} \end{cases} \tag{7}$$

$$c_i^h = \begin{cases} f_3^h(b_0 b_1 b_2) \times 3^h, & \text{if } b_j \neq \text{'}-\text{'} \textit{for all } 0 \leq j \leq 2 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

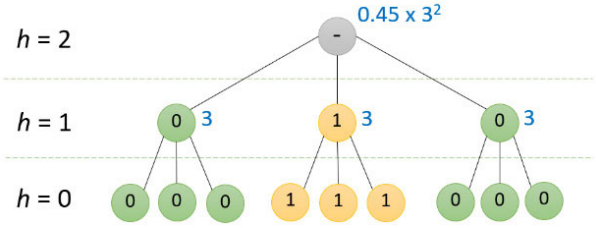


FIGURE 11. Fitness value calculation in an H-Trap problem.

Fig. 11 shows the fitness value calculation when the bit-string is '000111000'.

$$H - Trap(x) = \sum_{h=1}^{H} \sum_{i=1}^{3^{(H-h)}} c_i^h \tag{9}$$

III. THE PROPOSED GENERALIZED CROSSOVER METHOD

This section presents the characteristics of earlier ring-based crossover techniques. The characteristics are extracted, correlated, grouped, and, finally, generalized to establish a newly designed method. The pseudocode of the proposed generalized ring-based crossover approach, its relation to the previous ring-based techniques, and a time complexity analysis are also presented in this section.

A. GENERALIZED CROSSOVER AND THE OTHER RING-BASED CROSSOVER METHODS

As mentioned in Section II, there are crossover techniques that benefit from ring-based chromosome formation. Such techniques share the concept of connecting both ends of chromosomes to form rings before exchanging semirings (i.e., the segments of a ring). However, there are subtle nuances among the ring-based crossover techniques. For instance, CRC shifts bits before exchanging semirings. AC allows the swapping of semirings starting at different chromosome positions. RC performs semiring reversion before an exchange. FRC always chooses the front and rear parts of chromosomes as semirings to be swapped. Table 2 summarizes the characteristics of the existing ring-based crossover techniques.

We analyzed the characteristics of the four ring-based crossover techniques and found that FRC is an AC variant. AC operates exactly as FRC does in the case that the semirings (being swapped) are always the front and rear parts of chromosomes. Furthermore, CRC and AC are similar. AC can be considered CRC if the semirings being swapped are always halves of the full rings and bit shifting is omitted. RC and FRC share only one common characteristic: both select the front and rear parts of chromosomes as semirings. The differences between them are 1) RC reverses semiring bits before semiring exchange, and 2) RC converts rings to linear chromosomes from the parent cut points, and FRC converts rings to linear chromosomes starting from the first bit of parents.

The above observations were considered in the design of generalized ring-based crossover (GRC), which incorporates

TABLE 2. Characteristics of the existing ring-based crossover techniques.

Characteristics	CRC	AC	RC	FRC
- Circular bit shift	✓	×	×	×
- Semirings at different circular positions are involved in crossover	×	✓	✓	✓
- The semiring length is always half a chromosome	✓	×	×	×
- Semirings involved in crossover are always a pair of the front and rear semirings of the parents	×	×	✓	✓
- The bit order is reversed during the exchange of semirings	×	×	✓	×
- The linear chromosomes of offspring start from their parents' first bits and proceed to the last bits, regardless of the cut point	✓	✓	×	✓
- The linear chromosomes of offspring start from their parents' cut points	×	×	✓	×

TABLE 3. Time complexity.

Crossover technique	Time complexity				
	Circular bit shift	Determine cut points	Semiring exchange	Offspring's chromosome rearrangement	Total
Circle-ring crossover	O(N)	O(1)	O(N)	-	O(N)
Annular crossover	-	O(1)	O(N)	-	O(N)
Ring crossover	-	O(1)	O(N)	O(N)	O(N)
Front-rear crossover	-	O(1)	O(N)	-	O(N)
Generalized ring-based crossover	O(N)	O(1)	O(N)	O(N)	O(N)

all the characteristics discussed. In other words, all previous techniques can be considered GRC variants. Algorithm 1 illustrates each step of GRC. In particular, GRC has three main steps: 1) chromosome shifting, 2) chromosome exchange, and 3) chromosome rearranging.

Based on the shifting probability, GRC determines whether a chromosome is circularly *sLength*-bit shifted (lines 2–6). The second step (lines 7–32) is to randomize the semiring lengths and the cut point locations before swapping them. Note that the semiring length of the two chromosomes is always the same. The variables *pos_r1* and *pos_r2* are the cut points of parents' chromosomes, which may be at different positions. With the reverse probability, GRC reverses semirings while exchanging chromosome parts (lines 15–24). The first parent's semiring (starting at *pos_r1*) and the second parent's semiring (starting at *pos_r2*) are swapped. Note that all arrays are circular. With the probability of 1-*reverse_prob*, the semirings are exchanged without a reverse procedure (lines 26–31). The final step (lines 33–41) is to rearrange the offspring bits. The algorithm uses the rearranging probability to determine whether the offspring chromosomes should be rearranged so that they start from the parents' cut points instead of the parents' first-bit positions. If the algorithm determines that the offspring chromosomes start from the parents' first-bit positions, bit rearrangement is unnecessary. Otherwise, the offspring chromosomes are rearranged by first copying each parent's bit at the cut point to the corresponding offspring's first bit. The copying (in circular fashion) process continues until all parent bits are copied.

Given the two parents, which are 11111111 and 00000000, Fig. 12 illustrates the parameter configuration for GRC, which corresponds with that for the previous ring-based crossover techniques.

B. TIME COMPLEXITY ANALYSIS

The run time of the proposed GRC along with the existing ring-based crossover techniques can be derived in each of the following steps: 1) circular bit shifting, 2) cut-point determination, 3) semiring exchange, and 4) offspring chromosome rearrangement. As shown in Table 3, the worst-case run time complexities for all the ring-based crossover techniques are O(N), where N is the chromosome length.

The overall GA run time includes the time spent creating (i.e., crossover and mutation operations) all offspring and evaluating their fitness values. Let M, T, and $f_e(N)$ be the population size, the number of generations, and the amount of time used for fitness evaluation, respectively.

Given that the run time of each mutation is O(1), as the total number of generated offspring is MT, the run time required to create all offspring is O($MT(N+1)$). The run time required to evaluate all offspring is O($MTf_e(N)$). Therefore, GA's total run time is O($MT(N+1+f_e(N))$).

IV. EXPERIMENTAL RESULTS

In this section, we present the experimental and behavioral analysis results regarding the crossover techniques. We first discuss the experimental results and the behavioral analysis for a case in which crossover techniques are used to solve the one-max and zero-max problems, which are considered simple problems. We then report the experimental results and perform behavioral analyses for royal road problems, representing any problems with solutions composed of building

Algorithm 1 Generalized Ring-Based Crossover

```
1: procedure GRC(parent1, parent2)
2:   if prob_1 < shifting_prob then
3:         sLength = random(1, chromosomeLength-1)
4:         parent1 = CircularShift(parent1, sLength))
5:         parent2 = CircularShift(parent2, sLength)
6:   end if
7:   child1 = parent1
8:   child2 = parent2
9:   semiRingLength = random(1, chromosomeLength-1)
10:  pos_r1 = random(1, chromosomeLength-1)
11:  pos_r2 = random(1, chromosomeLength-1)
12:  p1 = pos_r1
13:  p2 = pos_r2
14:  if prob_2 < reverse_prob then
       // reverse and exchange semirings
15:    for i = 0 to semiRingLength-1
16:      revpos = (p2 + (semiRingLength – 1)) mod
17:      child1[p1] = parent2[revpos]
18:      child2[revpos] = parent1[p1]
19:      p1 = (p1 + 1) mod chromosomeLength
20:      p2 = p2 – 1
21:      if p2 < 0 then
22:        p2 = chromosomeLength - 1
23:    end if
24:  end for
25:  else
       // exchange semirings without reverse
26:    for i = 0 to semiRingLength-1
27:      child1[p1] = parent2[p2]
28:      child2[p2] = parent1[p1]
29:      p1 = (p1 + 1) mod chromosomeLength
30:      p2 = (p2 + 1) mod chromosomeLength
31:    end for
32:  end if
33:  if prob_3 < rearrange_prob then
34:    pos_r1 = (pos_r1 + semiRingLength) mod
35:    for i = 0 to chromosomeLength −1
36:      child1[i] = parent1[pos_r1]
37:      child2[i] = parent2[pos_r2]
38:      pos_r1 = (pos_r1 + 1) mod
           chromosomeLength
39:      pos_r2 = (pos_r2 + 1) mod
           chromosomeLength
40:    end for
41:  end if
42:  return child1, child2
43: end procedure
```

blocks. Furthermore, the performance and behavior of each crossover technique for the H-IFF problem are studied. The H-IFF problem is a multimodal problem (i.e., a problem with multiple optimal solutions), and solutions are based on hierarchical building-block structures. The final two experiments are conducted to study the performance and behavior of crossover methods when solving trap and hierarchical trap problems, which are considered very difficult benchmark problems. These problems are designed to fool algorithms, causing them to search in the direction opposite the one that leads to the optimal solution.

TABLE 4. Experimental environment and parameter setting.

Environment	Details
OS	Microsoft Windows 10 Enterprise
CPU	Intel(R) Core(TM) i7–3770 (3.40 GHz)
RAM	12 GB
Language	C++
Compiler	g++ 10.2.0
Parameter	**Value**
Population size	30 (for one-max, zero-max and trap (k=3)) 80 (for trap (k=5)) 500 (for royal road and H-IFF) 1000 (for H-Trap)
Crossover rate	0.8
Mutation rate	0.01
Tournament size	4
Maximum number of generations	500 (for one-max, zero-max and trap (k=3)) 1000 (for royal road, H-IFF, trap (k=5) and H-Trap)
Number of runs	100

In all experiments, we set the crossover rate to 0.8 and chose bitwise mutation as the mutation scheme, with a mutation rate of 0.01. A tournament selection of size 4 is used in the selection process, and the parameters are the same as those in [12]. Table 4 shows the experimental and parameter settings.

To quantify the algorithm's performance, we consider not only fitness values but also the number of fitness evaluations, which is a computational cost measurement commonly used in the field of evolutionary computations. Efficient algorithms require fewer fitness evaluations than inefficient algorithms. The number of fitness evaluations represents the number of times that each algorithm consults the clairvoyant to guide the search and identify the optimal solution. Therefore, the number of fitness evaluations can be used to fairly compare different genetic algorithms. The performance results are presented in Tables 5–9.

The first row of the table for each problem contains the best solution (i.e., the greatest fitness value) each crossover technique yields after all 100 runs. The second row of each table includes the average of the 100 best fitness values, each of which is associated with a specific run. The third and fourth rows of each table contain the average numbers of fitness evaluations and the percentages of successful runs in which the algorithms discover the optimal solution, respectively. The last row in each table lists the exact time consumption (in seconds) of each crossover technique.

The behavior of each algorithm in terms of diversity and convergence is illustrated in Figs. 13–19. The

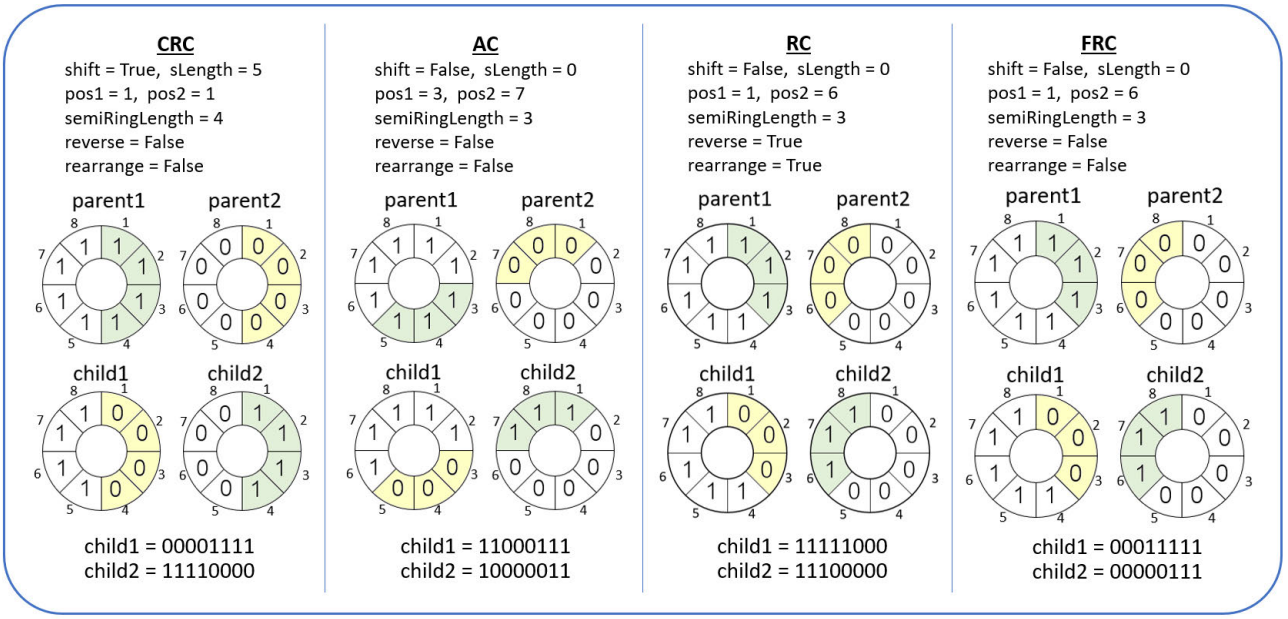
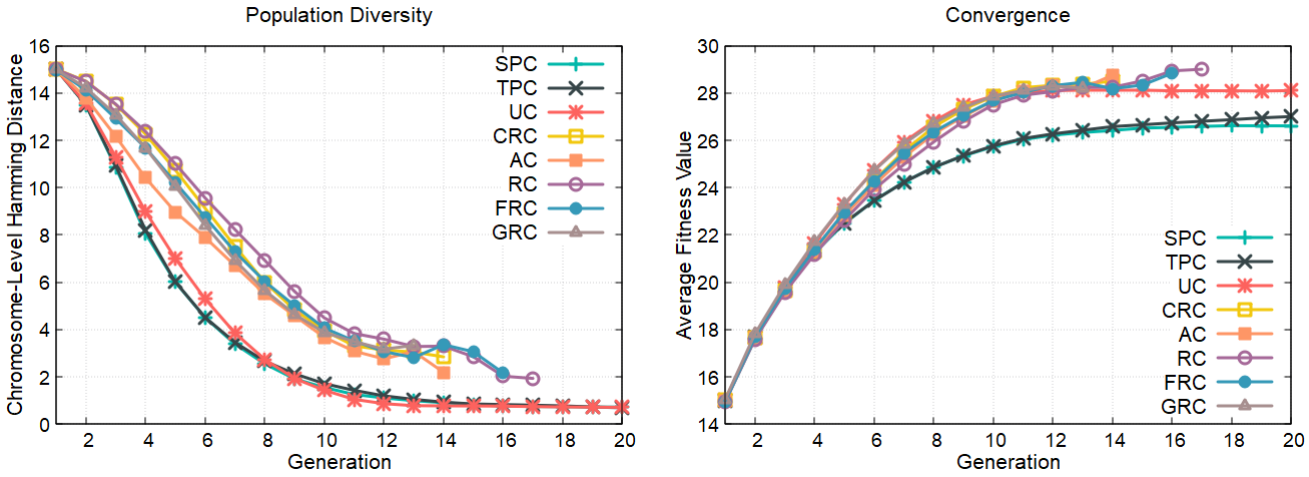


FIGURE 12. Parameter-setting examples in GRC, in which the method operates in the same ways as the methods in previous ring-based techniques.

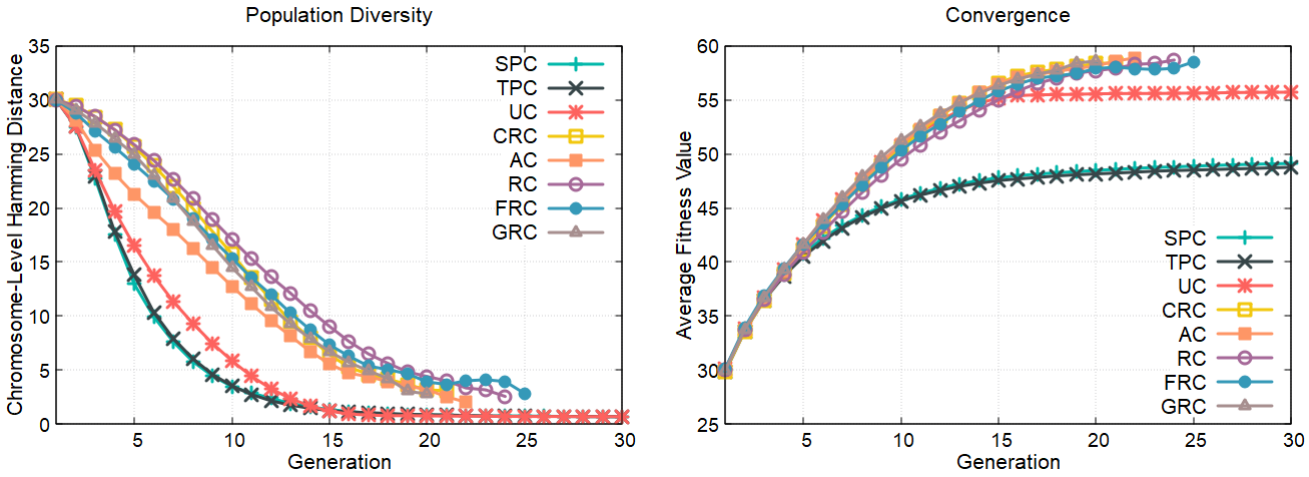
TABLE 5. The performance of crossover techniques for the one-max and zero-max problems.

Problem	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
One-max size=30	30	Best	30	30	30	30	30	30	30	30
		Avg. best	30	30	30	30	30	30	30	30
		Avg. #FE	5461.2	5447.7	1976.4	291.0	293.1	319.8	298.5	**273.9**
		% success	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0120	0.0089	0.0025	0.0003	0.0002	0.0004	0.0002	0.0003
One-max size=60	60	Best	60	60	60	60	60	60	60	60
		Avg. best	58.9	58.9	59.6	60	60	60	60	60
		Avg. #FE	13599.9	13848	10606.2	479.4	481.2	553.8	506.1	**470.1**
		% success	37%	35%	68%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0312	0.0372	0.0230	0.0005	0.0004	0.0007	0.0005	0.0006
One-max size=120	120	Best	117	116	120	120	120	120	120	120
		Avg. best	109.9	109.7	115.7	120	120	120	120	120
		Avg. #FE	15000	15000	14969.4	793.8	804	878.4	835.2	**782.7**
		% success	0%	0%	3%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0419	0.0485	0.0597	0.0011	0.0013	0.0018	0.0011	0.0015
Zero-max size=30	30	Best	30	30	30	30	30	30	30	30
		Avg. best	30	30	30	30	30	30	30	30
		Avg. #FE	5640.0	5257.8	1825.2	288.6	290.7	311.1	295.2	**276.0**
		% success	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0113	0.0104	0.0028	0.0002	0.0002	0.0003	0.0002	0.0003
Zero-max size=60	60	Best	60	60	60	60	60	60	60	60
		Avg. best	59.1	59.1	59.6	60	60	60	60	60
		Avg. #FE	13949.1	13559.4	10525.5	483.0	490.8	537.6	506.7	**464.4**
		% success	38%	39%	70%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0283	0.0302	0.0318	0.0004	0.0004	0.0007	0.0005	0.0006
Zero-max size=120	120	Best	116	116	120	120	120	120	120	120
		Avg. best	110.6	109.9	115.8	120	120	120	120	120
		Avg. #FE	15000.0	15000.0	14939.4	801.0	810.3	890.1	839.1	**786**
		% success	0%	0%	3%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0394	0.0499	0.0573	0.0011	0.0010	0.0019	0.0010	0.0014

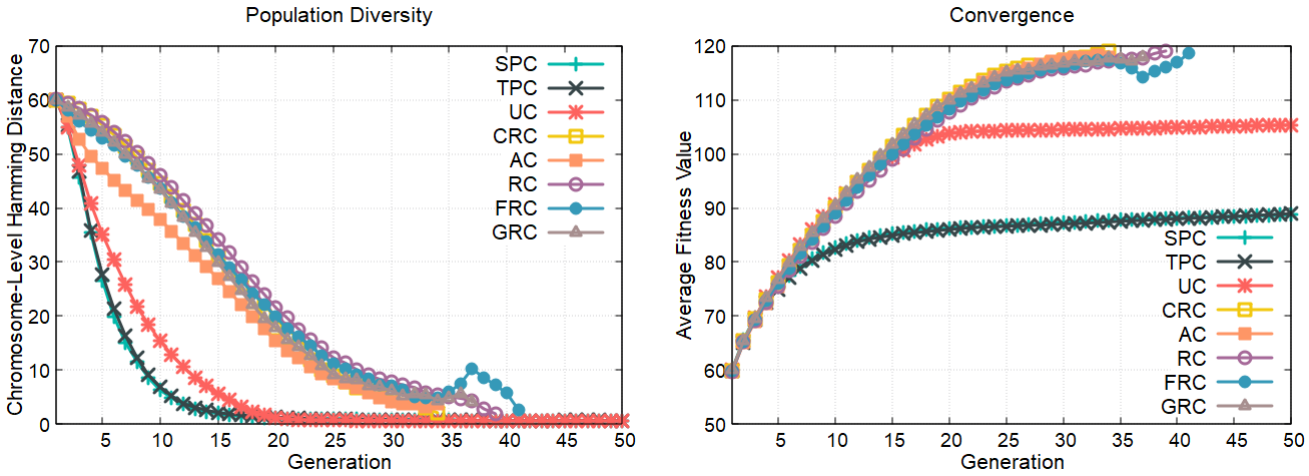
algorithms, which perform crossover operations, terminate under two conditions: 1) the optimal solution is discovered or 2) the algorithm reaches the maximum number of fitness evaluations allowed. Additionally, note that diversity is measured by the Hamming distance at the chromosome level, as described in [31].



(a) One-max problem (size=30): population diversity (left) and convergence (right)



(b) One-max problem (size=60): population diversity (left) and convergence (right)



(c) One-max problem (size=120): population diversity (left) and convergence (right)

FIGURE 13. **Population diversity and convergence for the one-max problem.**

A. ONE-MAX AND ZERO-MAX PROBLEMS

The fitness value of each candidate is calculated directly from the number of ones or zeros that appear in the chromosome. The one-max problem favors 1 bit, while the zero-max problem favors 0 bits. We used both problems to verify that the crossover techniques are unbiased toward ones or zeros.

TABLE 6. The performance of crossover techniques for the royal road problem.

Problem	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
R size=64	256	Best	256	256	256	256	256	256	256	256
		Avg. best	178.5	191.6	194.6	256	256	256	256	256
		Avg. #FE	287470	251265	259595	6075	6085	5710	5725	**5500**
		% success	44%	51%	52%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.2674	0.2253	0.2364	0.0073	0.0068	0.0094	0.0062	0.0071

TABLE 7. The performance of crossover techniques for H-IFF problems.

Problem	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
H-IFF 32	192	Best	192	192	192	192	192	192	192	192
		Avg. best	189.6	189.4	156.1	192	192	192	192	192
		Avg. #FE	203900	157750	391480	3845	3980	3540	3510	**3360**
		% success	93%	92%	22%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.2736	0.2797	0.9574	0.0072	0.0080	0.0070	0.0066	0.0071
H-IFF 64	448	Best	448	448	448	448	448	448	448	448
		Avg. best	355.6	350.8	296.6	448	448	448	448	448
		Avg. #FE	426190	416295	495120	6160	6455	6015	5950	**5750**
		% success	15%	17%	1%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	1.9457	1.8628	1.9779	0.0199	0.0209	0.0227	0.0189	0.0197
H-IFF 128	1024	Best	702	716	720	1024	1024	1024	1024	1024
		Avg. best	576.5	580.4	546.8	1024	1024	1024	1024	1024
		Avg. #FE	500000	500000	500000	9620	9640	9230	9205	**9170**
		% success	0%	0%	0%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	3.6199	3.6370	3.5729	0.0568	0.0588	0.0638	0.0552	0.0589

Table 5 presents the efficiency of the crossover methods as the size of each problem increases. In the one-max and zero-max problems of size 30, all crossover techniques can obtain the optimal solution. However, there are large gaps in the numbers of fitness evaluations performed by the SPC, TPC, UC and ring-based crossover methods. When the problem size is large, the traditional crossover methods (SPC, TPC and UC) seem unpromising; notably, for a problem size of 60, their success rates vary from approximately 35–70%. If the problem size increases to 120, SPC and TPC cannot find the optimal after 100 runs. UC performs just slightly better than SPC and TPC, as it still provides a few successful runs.

The ring-based crossover methods manage to successfully discover the optimal solution for all problem sizes. However, the number of fitness evaluations varies among techniques. The results show that the proposed GRC method requires the fewest fitness evaluations.

In terms of population diversity and convergence, Figs. 13 and 14 demonstrate how the crossover methods behave for the one-max and zero-max problems, respectively. The population diversities of SPC, TPC and UC decrease more quickly than those for other methods during the first few generations. The larger the problem size is, the larger the diversity gap between the traditional and ring-based crossover results. The convergence graphs also confirm that as the problem size increases, the traditional crossover methods are unlikely to converge to the optimal solutions.

B. ROYAL ROAD PROBLEM

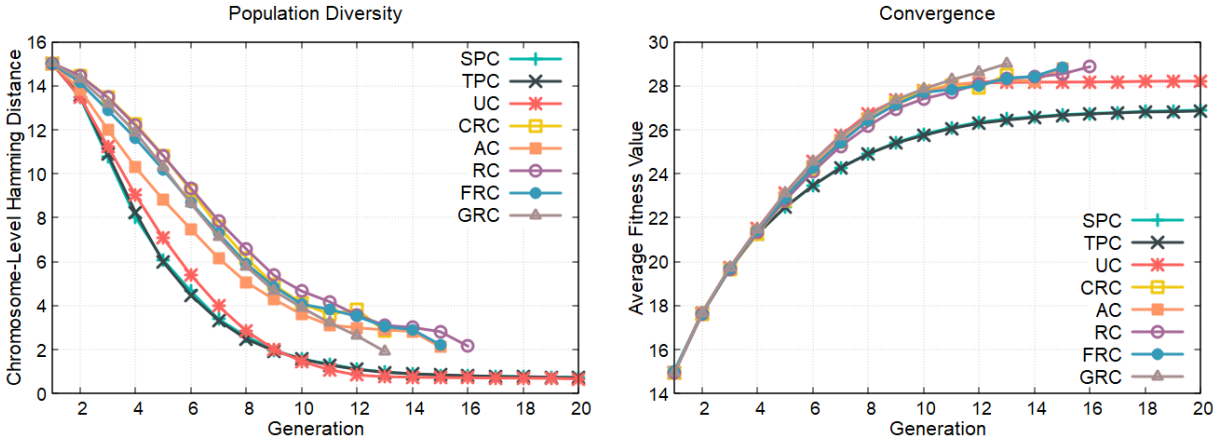
In this experiment, we tested the algorithms based on problems with solutions composed of building blocks. In the GA literature, building-block preservation is essential for GAs to obtain satisfactory solutions [32]. The famous royal road function is often used in studies and behavioral analyses of GAs.

Table 6 shows the performance of the algorithms. The traditional crossover methods are able to reach the optimal solution after a certain number of runs (with a success rate of 44–52%). In contrast, the ring-based crossover methods always successfully discover the optimal solution. Among them, GRC requires the fewest fitness evaluations.

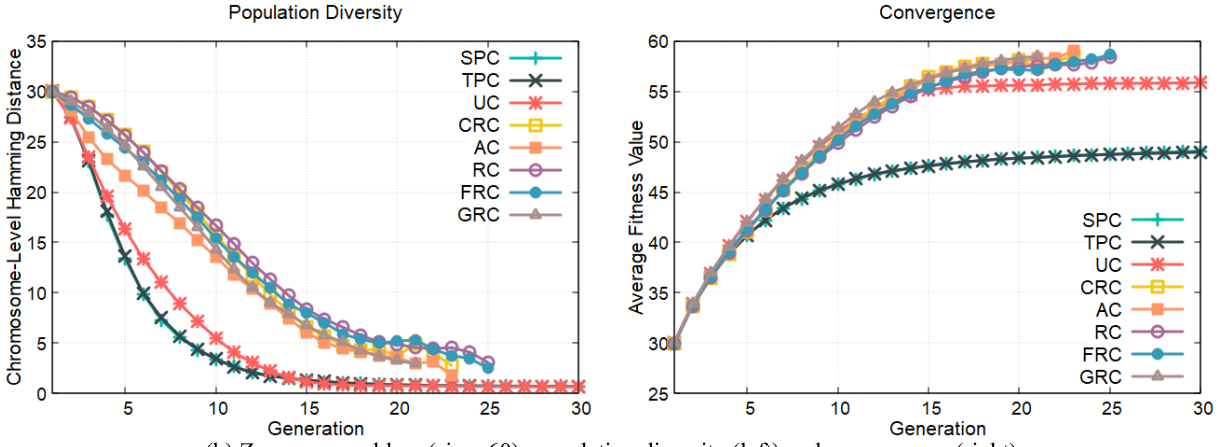
The population diversity and the convergence of each algorithm are shown in Fig. 15. For traditional crossover, the diversity decreases early before discovering the optimal solution. Although UC is able to maintain high diversity, it converges slowly toward local optima.

C. HIERARCHICAL IF-AND-ONLY-IF PROBLEM

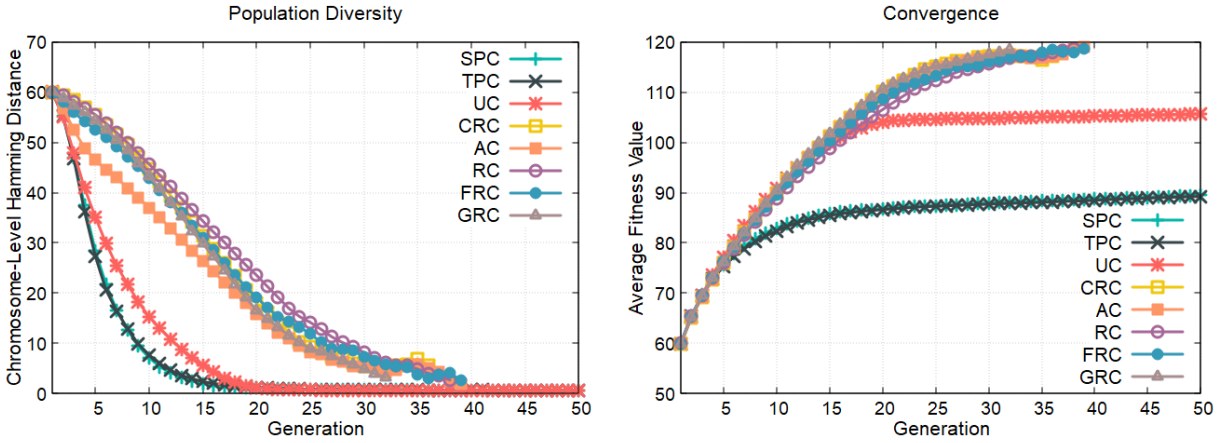
The experiments in this subsection are based on the H-IFF problem, which is hierarchical and multimodal. As shown in Table 7, for almost all runs, SPC and TPC are successful in discovering the optimal solution to the 32-bit problem; notably, the optimal solutions are found with success rates of 93% and 92%, respectively. Nevertheless, when the problem size increases to 64 and 128 bits, the success rates drop to approximately 15% and 0%, respectively. UC performs worse than SPC and TPC; its success rates are 22%, 1% and 0% for the problems with different numbers of bits. The behaviors in terms of population diversity and convergence are shown in Fig. 16. SPC, TPC and UC behave the same way as they did when solving the royal road problem. The diversities of SPC and TPC decrease very early in the algorithm process. Although the UC diversity is high, its convergence time is large, and it tends to move toward local optima. All the



(a) Zero-max problem (size=30): population diversity (left) and convergence (right)



(b) Zero-max problem (size=60): population diversity (left) and convergence (right)



(c) Zero-max problem (size=120): population diversity (left) and convergence (right)

FIGURE 14. **Population diversity and convergence for the zero-max problem.**

ring-based crossover methods successfully obtain the optimal solution. As in the previous two problems, GRC requires the fewest fitness evaluations.

D. TRAP PROBLEM

In this subsection, we benchmark the algorithms based on the trap problem (with $k = 3$ and $k = 5$). This problem is more challenging than those in subsections 4.1–4.3 because it is designed to persuade the algorithms to search in the wrong direction.

Table 8 presents the performance results for all crossover methods. The success rates of SPC, TPC, and UC are generally low; these methods rarely discover the global optimum for 30-bit trap problems (with k values of 3 and 5) and never

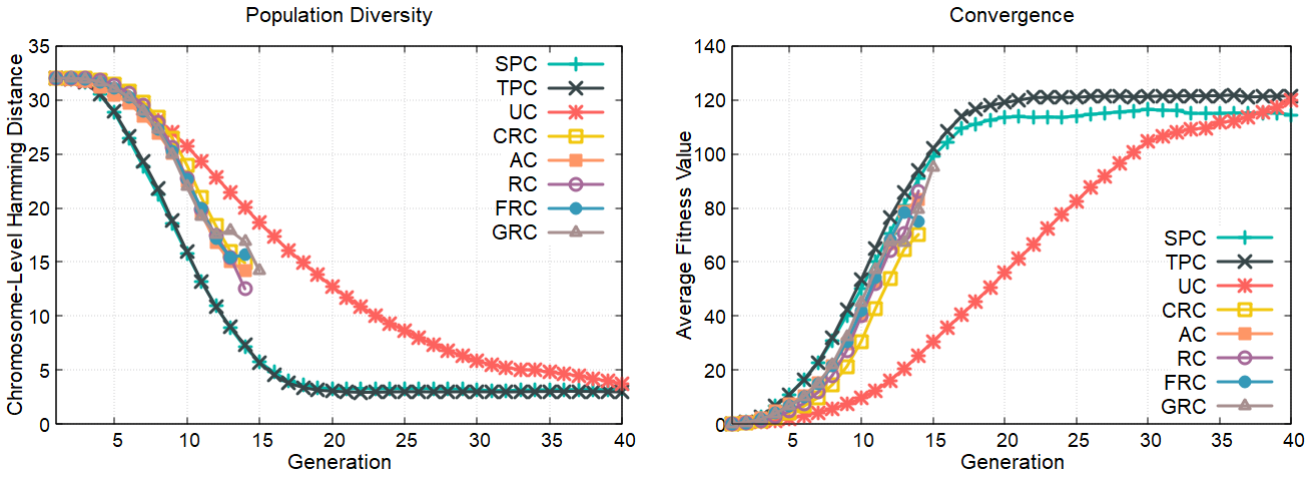


FIGURE 15. Population diversity (left) and convergence (right) for the royal road problem.

find the optimum in 60- and 120-bit trap problems. The ring-based crossover methods all achieve a 100% success rate for the 30-, 60-, and 120-bit trap problems (with $k = 3$). Our proposed GRC approach requires the fewest fitness evaluations.

For a trap size of 5, only FRC and GRC achieve a 100% success rate, while AC, RC, and CRC succeed in discovering the optimal solutions in approximately 96–97%, 60–81%, and 47–81% of all runs, respectively. The competency of FRC in solving the trap problem was discussed in [12], and FRC solved problems with 40 fewer fitness evaluations than BOA [16]. Our proposed GRC approach surprisingly outperforms FRC in terms of the number of fitness evaluations.

The graphs in Figs. 17–18 confirm that the diversities of the traditional crossover methods decrease more quickly than those of the ring-based crossover methods, resulting in trapping at local optima. Fig. 18 clearly demonstrates that FRC and GRC yield the two highest diversities among the crossover methods compared.

E. HIERARCHICAL TRAP PROBLEM

This section presents the experimental results for the H-Trap problem, which is more challenging than the problems addressed above because it is designed to fool GAs. The problem structure is also hierarchical, making the solutions comparatively obscure to the algorithms.

Table 9 presents the performance results for the algorithms in solving the 27-bit H-Trap problem. Although the problem size is small, no crossover technique can successfully discover the optimal solution 100% of the time.

The graphs in Fig. 19 show that all crossover methods lose diversity in early generations. The populations of FRC, AC and GRC (ranked in ascending order) are more diverse than those of other methods in later generations. Nevertheless, this diversity is insufficient for obtaining the optimal solution, and all algorithms converge to a local optimum.

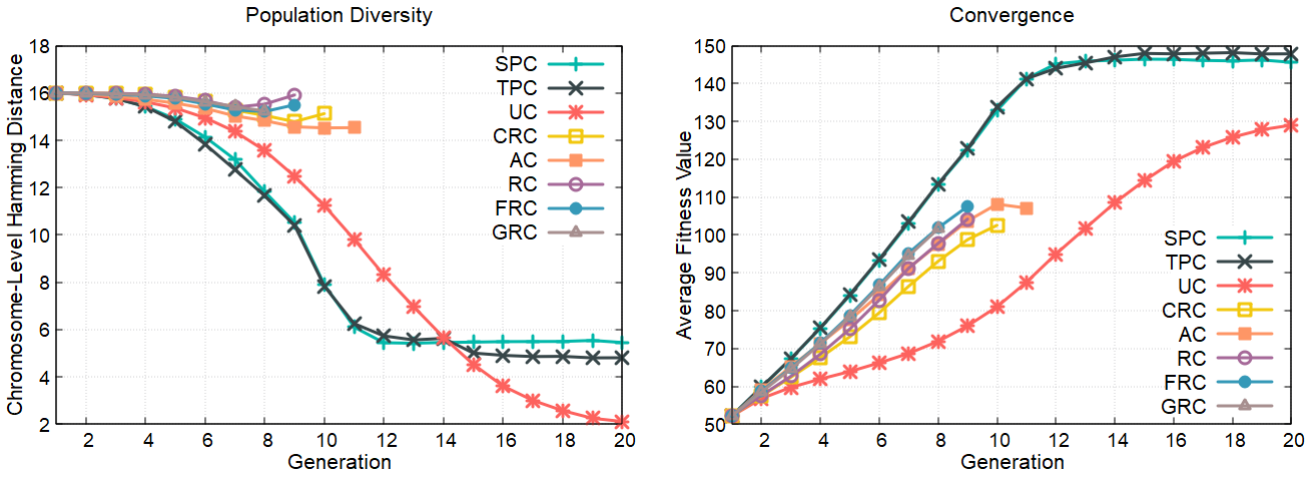
For extremely difficult problems such as hierarchical traps, traditional and ring-based crossover techniques cannot ensure the discovery of the global optimum. Although the proposed GRC method outperforms the other methods for this problem, it yields a maximum of 88% success. Thus, more complex methods (e.g., HBOA) are required at the expense of increased computational resources and additional fitness evaluations to solve the H-Trap problem.

V. DISCUSSION

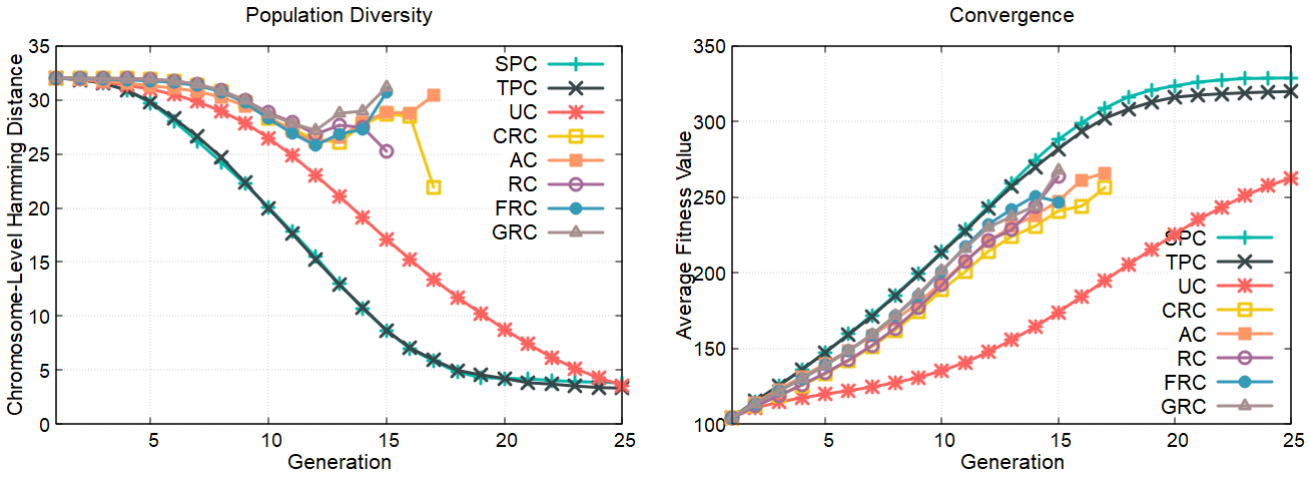
Based on our main experimental results in the previous section, this section discusses the behaviors of the ring-based crossover algorithms that are beneficial for discovering better solutions than those of traditional algorithms. The reasons why the proposed GRC technique outperforms other methods are discussed, and the limitations of GRC and future work are described.

A. TRADITIONAL VERSUS RING-BASED CROSSOVER TECHNIQUES

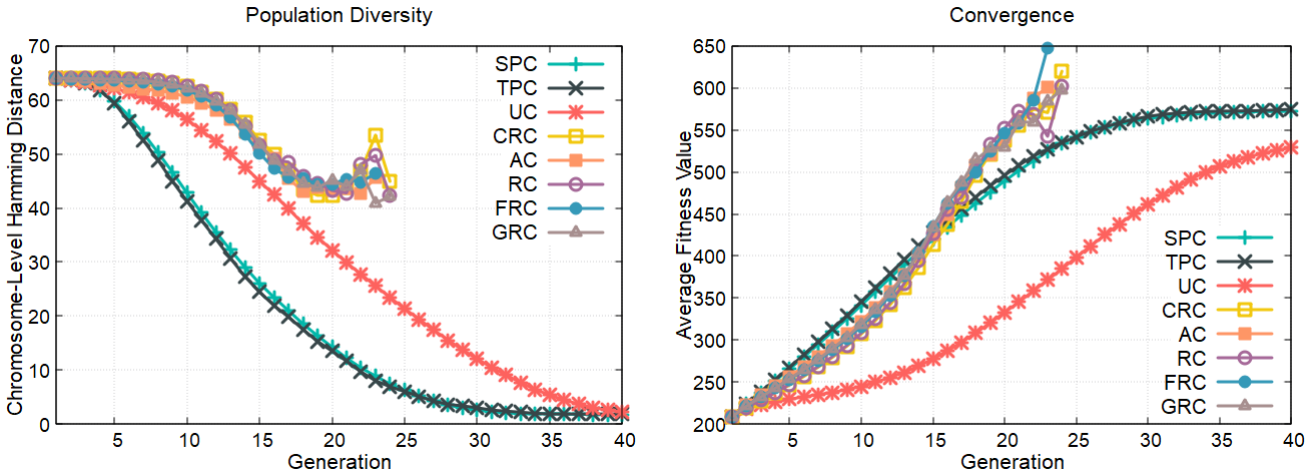
For comparison, assume the first bit of each parent chromosome is 0. In this situation, offspring, for which the left-most bit is 1, are never produced by SPC, TPC and UC because these methods preserve bit positions. Specifically, after determining the cut points of the parent chromosomes, SPC always exchanges only the right parts of the chromosomes. Similarly, TPC randomly selects 2 cut points, and then the middle chromosome parts are swapped. UC produces offspring by selecting either the first or second parent's i^{th} bit. Therefore, if the 1^{st} bits of both parents are zeros, it is impossible for the offspring's 1^{st} bit to be 1. Conversely, ring-based crossover methods allow offspring bits to be inherited from the parents' bits at different positions. It is possible that the leftmost bit of the offspring chromosome can be inherited from one of the parents' rightmost bits, and vice versa. In particular, CRC might shift bits before crossover. AC allows the swapping of semirings, the cut points of which are at different positions. RC forms a ring by conjoining the parents' chromosomes and splitting rings in half at a random cut point. The i^{th} bit of the



(a) H-IFF (size=32): population diversity (left) and convergence (right)



(b) H-IFF (size=64): population diversity (left) and convergence (right)



(c) H-IFF (size=128): population diversity (left) and convergence (right)

FIGURE 16. **Population diversity and convergence for the H-IFF problem.**

offspring might be inherited from parent chromosomes at any position. FRC allows the swapping of the front and rear parts of chromosomes. GRC encompasses all the features of the abovementioned ring-based crossover methods. Such characteristics of ring-based crossover are beneficial for enhancing chromosome diversity and avoiding premature convergence.

TABLE 8. The performance of crossover techniques for trapping problems.

Problem	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
Trap k=3 size=30	30	**Best**	30	30	30	30	30	30	30	30
		Avg. best	26.6	26.5	25.7	30	30	30	30	30
		Avg. #FE	14854.8	14853.3	14853.0	373.5	358.2	376.8	350.1	**331.5**
		% success	1%	1%	1%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0353	0.0372	0.0386	0.0004	0.0004	0.0005	0.0005	0.0004
Trap k=3 size=60	60	**Best**	54	55	56	60	60	60	60	60
		Avg. best	49.2	49.3	49.4	60	60	60	60	60
		Avg. #FE	15000.0	15000.0	15000.0	732.6	681.0	748.5	717.9	**632.7**
		% success	0%	0%	0%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0428	0.0455	0.0523	0.0013	0.0009	0.0016	0.0008	0.0010
Trap k=3 size=120	120	**Best**	99	100	102	120	120	120	120	120
		Avg. best	91.2	91.5	93.1	120	120	120	120	120
		Avg. #FE	15000.0	15000.0	15000.0	1340.1	1163.1	1420.5	1315.8	**1139.4**
		% success	0%	0%	0%	**100%**	**100%**	**100%**	**100%**	**100%**
		Avg. run time	0.0558	0.0604	0.0743	0.0031	0.0029	0.0051	0.0027	0.0034
Trap k=5 size=30	30	**Best**	30	30	27	30	30	30	30	30
		Avg. best	27.6	27.7	25.1	28.9	29.8	28.9	30	30
		Avg. #FE	78416.8	77622.4	80000.0	17840.8	3920.0	16392.8	1092.8	**1061.6**
		% success	2%	3%	0%	81%	97%	81%	**100%**	**100%**
		Avg. run time	0.1515	0.1863	0.1940	0.0398	0.0017	0.0416	0.0010	0.0010
Trap k=5 size=60	60	**Best**	57	56	54	60	60	60	60	60
		Avg. best	52.9	52.9	49.6	55.8	59.6	56.0	60	60
		Avg. #FE	80000.0	80000.0	80000.0	30575.2	4993.6	27889.6	2148.8	**2076.0**
		% success	0%	0%	0%	65%	97%	67%	**100%**	**100%**
		Avg. run time	0.2147	0.2312	0.2399	0.0892	0.0246	0.1018	0.0021	0.0027
Trap k=5 size=120	120	**Best**	109	108	103	120	120	120	120	120
		Avg. best	103.0	103.1	98.8	107.3	119.0	110.4	120	120
		Avg. #FE	80000.0	80000.0	80000.0	44413.6	7667.2	33861.6	3783.2	**3629.6**
		% success	0%	0%	0%	47%	96%	60%	**100%**	**100%**
		Avg. run time	0.2689	0.2994	0.3491	0.1624	0.0344	0.1459	0.0073	0.0084

TABLE 9. The performance of crossover techniques for the hierarchical trapping problem.

Problem	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
H-Trap 27	81	**Best**	81	81	78.3	81	81	81	81	81
		Avg. best	80.0	80.3	78.1	79.9	79.8	80.5	80.5	80.7
		Avg. #FE	364780	265550	1000000	394550	434540	185450	175380	**125760**
		% success	64%	74%	0%	61%	57%	82%	83%	**88%**
		Avg. run time	1.0101	0.8422	2.0073	0.8147	0.9345	0.5924	0.4826	0.4673

Generating offspring chromosomes with a variety of bit patterns is important because avoiding premature convergence is vital in GA research [33].

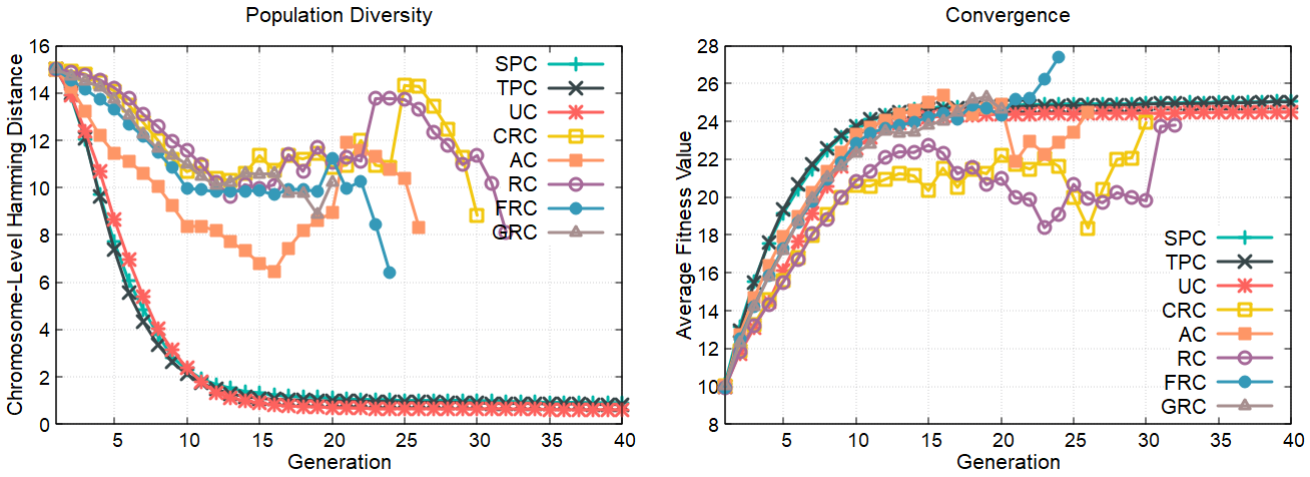
Based on the experimental results, SPC, TPC and UC perform inefficiently when solving the trap and H-Trap problems. The problems are designed to fool GA algorithms to favor zeros, but the optimal solution is all ones. According to the behavior of these methods, as previously described, the algorithms produce chromosomes dominated by zero bits. If the 1st bits of both parents' chromosomes are zeros, it is impossible for SPC, TPC and UC to determine any paths leading toward the optimal solution. In contrast, ring-based crossover methods produce a more diverse population and are more likely to escape from traps.

B. SPECIFIC VERSUS GENERALIZED RING-BASED CROSSOVER TECHNIQUES

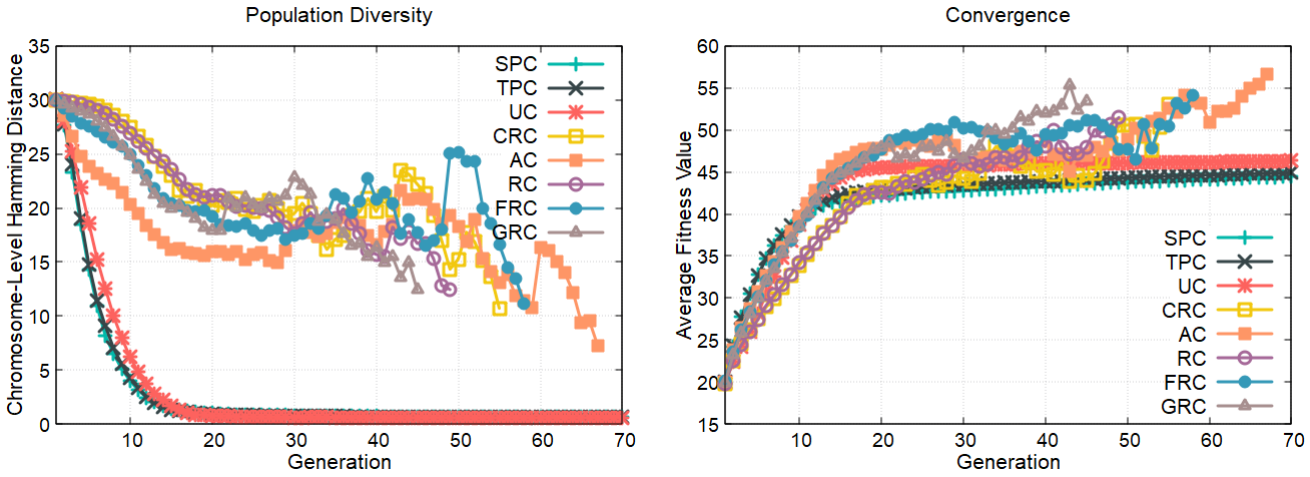
The proposed GRC method outperforms the other ring-based crossover techniques because it incorporates all the distinct characteristics of each method, including diversity and building-block preservation.

CRC, AC and RC are able to maintain high diversity. They allow the i^{th} bit of the first parent to be exchanged with the j^{th} bit of the second parent, where i is not always equal to j. This characteristic encompasses CRC bit shifting, AC random semiring selection from both parents, and RC semiring reversion. The diversity graphs illustrated in Figs. 13–17 show that CRC and RC are the two highest-diversity methods. Figs. 18–19 show that AC produces more diversified populations than CRC and RC for difficult problems (the trap and H-Trap problems). Although the three methods maintain diversity, their mechanisms differ. CRC bit shifting and RC semiring reversion could destroy the bit patterns containing the solution building blocks. However, AC's random semiring selection mechanism is able to preserve the bit pattern, and building blocks are likely to be preserved.

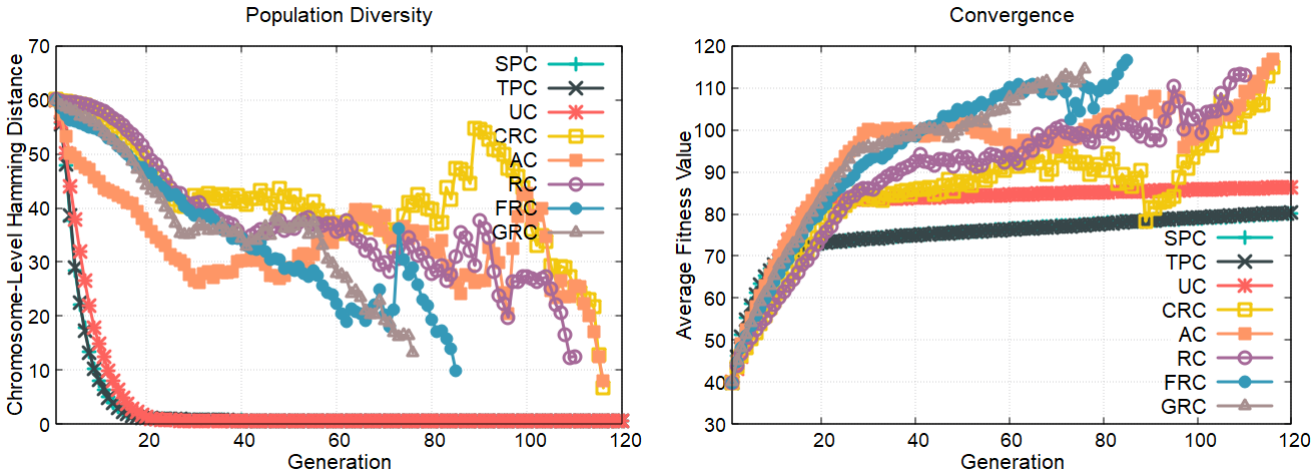
Dissimilar to the above three methods, FRC allows the swapping of the front and rear parts of chromosome.



(a) Trap 3 (size=30): population diversity (left) and convergence (right)



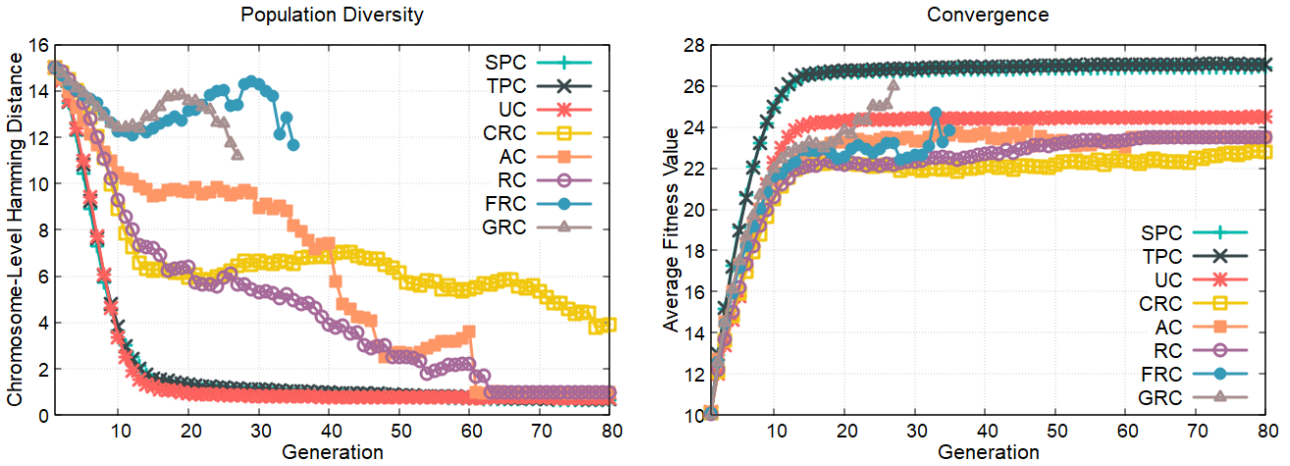
(b) Trap 3 (size=60): population diversity (left) and convergence (right)



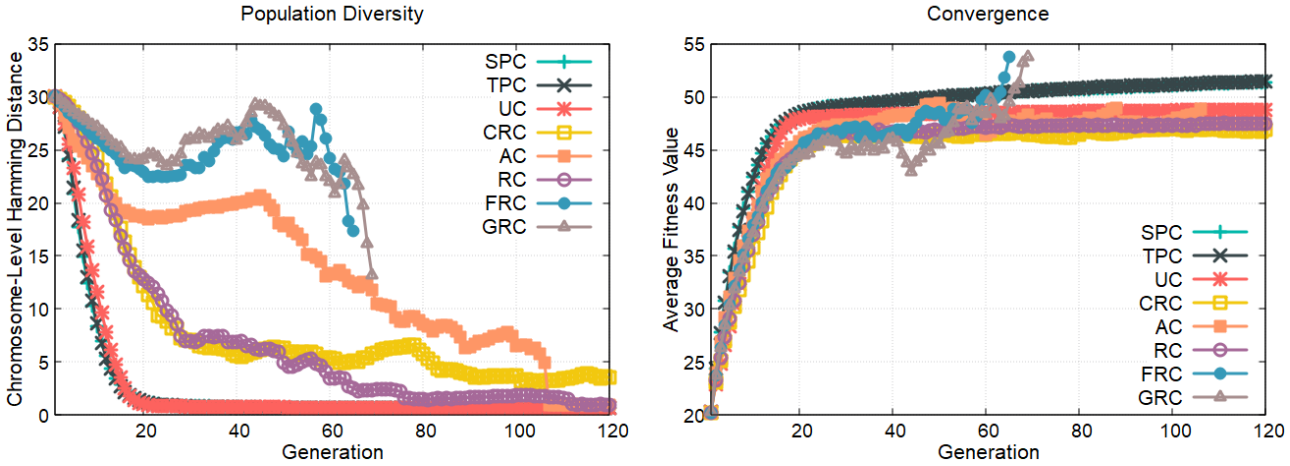
(c) Trap 3 (size=120): population diversity (left) and convergence (right)

FIGURE 17. **Population diversity and convergence for the trap problem (k = 3).**

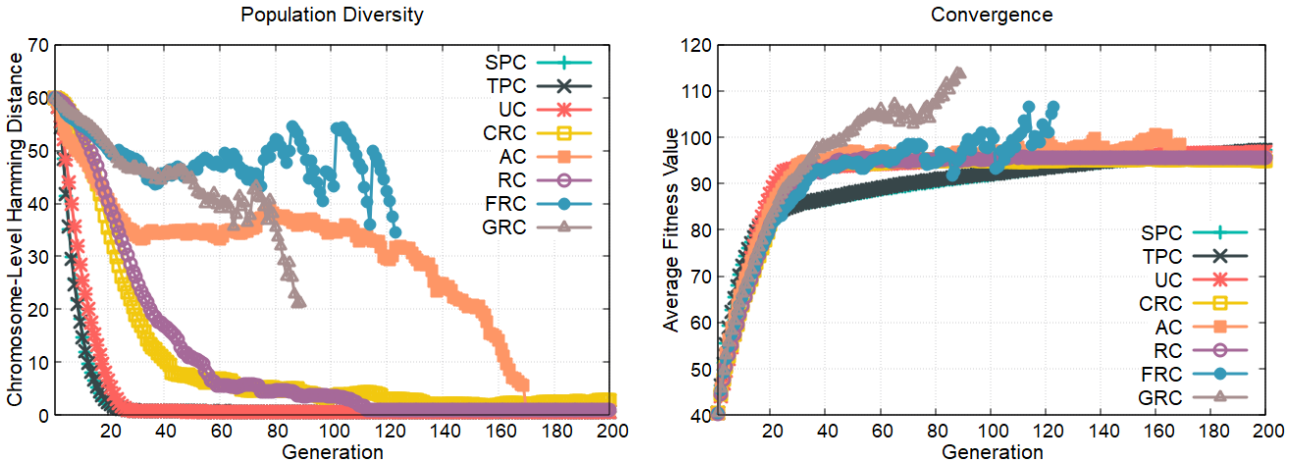
Although FRC sounds less flexible and diversified than other crossover methods, it preserves building blocks, as discussed in [12], and can efficiently solve problems. However, there are certain cases in which FRC cannot avoid trapping. For instance, given the two parents' chromosomes (000 111 000 and 000 111 000), FRC cannot generate offspring



(a) Trap 5 (size=30): population diversity (left) and convergence (right)



(b) Trap 5 (size=60): population diversity (left) and convergence (right)



(c) Trap 5 (size=120): population diversity (left) and convergence (right)

FIGURE 18. **Population diversity and convergence for the trap problem (k = 5).**

chromosomes with the long building block of 111 111, but CRC and AC can generate offspring bit patterns such as 111 111 000 and 000 111 111, which are comparatively closer to the optimal solution.

In short, GRC, which is designed as an umbrella of ring-based crossover methods, embodies the advantages of ring operations. GRC able to maintain population diversity and can preserve intrinsic building blocks.

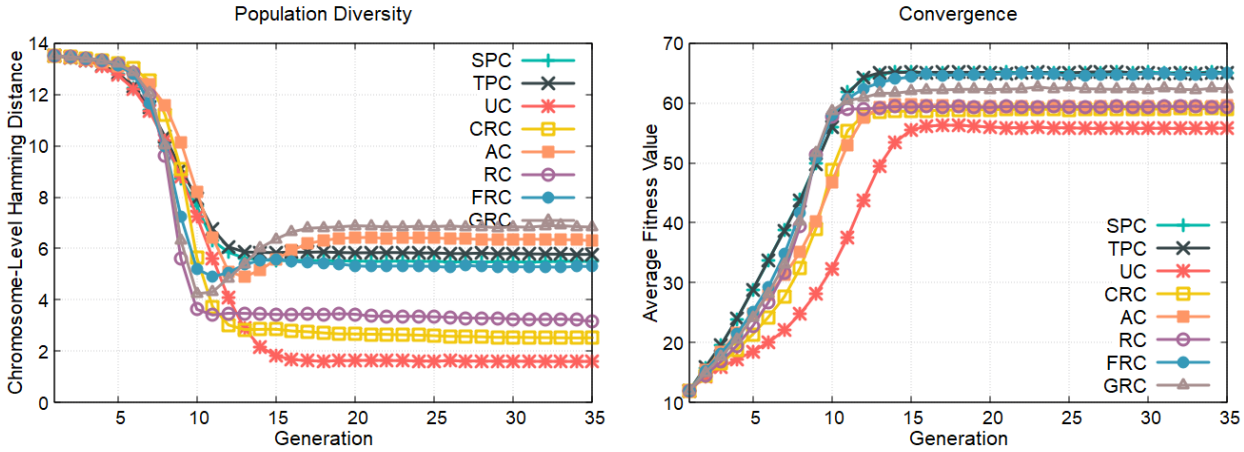


FIGURE 19. Population diversity and convergence for the H-Trap problem.

C. LIMITATIONS AND FUTURE RESEARCH

In this paper, we evaluated crossover techniques based on one-max, zero-max, royal road, hierarchical if-and-only-if, trap, and hierarchical trap problems. We selected these problems because their solutions are binary coded (i.e., solvable with the ring-based crossover methods considered in this paper, which are designed for operating binary-representation chromosomes). Furthermore, the corresponding fitness functions are unitation functions, which facilitate specification and analysis. These problems are often selected to comparatively analyze algorithm behaviors considering the building block composition. There are various nonring-based crossover techniques (e.g., sphere crossover [34], average crossover [35], flat crossover [36], edge recombination crossover [37] and partially mapped crossover [38]) that can be used to solve various problems (such as numerical optimization, traveling salesman, and vehicle routing problems) with chromosomes that are integer- or real-value encoded. Such problems are not within the scope of this paper.

However, to further assess the limitations of ring-based crossover, we conducted additional experiments that involved solving other kinds of problems, namely, numerical minimization problems, based on 30 functions from the CEC 2014 benchmark dataset [39]. To solve such problems, we changed the chromosome encoding from bit-string encoding to real-value encoding. For mutation, instead of bitwise mutation, we employed polynomial mutation [40], as it provides more promising preliminary results than does Gaussian mutation [40]. The crossover and mutation rates are the same as those in previous experiments, as shown in Table 4. The search space is $[-100, 100]^D$, where D is set to 10. According to CEC 2014, the number of runs per problem is 51, and the number of fitness evaluations per run is 100,000. The experimental results are shown in Table 10.

Overall, all crossover techniques yield similar results. GRC slightly outperforms the other methods, as its solutions are slightly closer to the optimal solution for most of the problems. As expected, in all test problems, GRC yields the best solutions superior than the other ring-based crossovers, as it incorporates the beneficial features of other crossover methods, thus enlarging search space and enhancing the probability of obtaining superior solutions.

However, for problem number 2 ($f_2(x) = x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2$) and problem number 3 ($f_3(x) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$)), all the ring-based crossover techniques yield results farther from the optimal results than those of the traditional techniques. We note that both problems have initial x_1 terms that are not summations, as are the subsequent terms. As a result, x_1 contributes to the fitness value differently from the other terms, and the chromosome positions are not of equal importance. Let e_i and E_i be the difference between the value of x_i and its optimal value (x_i^*), and the difference between the value of f_i and its optimal value (f_i^*), respectively. e_1 contributes to E_2 less than the other e_i ($i \neq 1$) do by a factor of 10^6. However, e_1 causes E_3 to be larger than the other e_i ($i \neq 1$) do by a factor of 10^6. During the evolution process, swapping x_1 for the elite parent with x_i from the other parent might change the elite status between parents. All ring-based crossover techniques exchange genes at different positions; therefore, they perform worse than traditional methods for these two problems. The experimental results suggest that although ring-based crossover techniques are good at promoting diversity, they are not promising for problems in which the elite status of the parents can be preserved only if their genes are inherited without position changes.

As shown in Table 10, GRC outperforms all the other crossover methods (both the traditional and ring-based methods) for 22 problems, performs equally well for 4 problems, and is outperformed for 4 problems. For problem numbers 4 and 7, GRC yields results slightly further from the optimal values (the GRC results are 401.2 and 701.3, while the results of traditional methods are 400.1 and 700.1, respectively). We note that for unimodal functions (problem numbers 1–3) and simple multimodal functions (problem numbers 4–16), the traditional methods and GRC yield similar performance (except for problem numbers 2 and 3, as previously explained). For the considered hybrid functions (problem numbers 17–22) and composition functions

TABLE 10. The performance of crossover techniques based on the CEC 2014 numerical optimization problems.

Problem No.	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
1	100	Best	5.33E+05	1.24E+05	1.06E+05	7.77E+06	9.33E+04	4.61E+06	1.29E+06	**7.89E+04**
		Avg. best	1.81E+07	1.55E+07	9.23E+06	3.15E+07	5.47E+06	4.70E+07	2.64E+07	6.54E+06
		Avg. run time	0.1409	0.1186	0.1186	0.1452	0.1244	0.1250	0.1351	0.1213
2	200	Best	**3185.0**	7589.3	7045.3	5.69E+08	3.99E+06	9.29E+08	4.50E+08	3.12E+06
		Avg. best	4.32E+04	8.09E+04	3.87E+04	1.22E+09	5.35E+07	2.99E+09	4.18E+07	4.84E+07
		Avg. run time	0.1069	0.0901	0.0913	0.1201	0.0977	0.0962	0.0971	0.0993
3	300	Best	**303.9**	599.4	329.4	3093.8	797.419	3280.38	1239.0	701.0
		Avg. best	9896.6	9597.2	9960.1	1.11E+04	9675.0	1.36e+04	9409.5	9214.8
		Avg. run time	0.1048	0.0888	0.0876	0.1241	0.0934	0.0990	0.1063	0.0996
4	400	Best	**400.1**	**400.1**	**400.1**	512.5	401.5	527.3	448.2	401.2
		Avg. best	430.6	431.1	428.3	745.0	444.4	712.1	516.3	442.1
		Avg. run time	0.1109	0.0919	0.0925	0.1213	0.0999	0.1002	0.1103	0.1011
5	500	Best	**520.0**	**520.0**	**520.0**	520.1	**520.0**	520.1	**520.0**	**520.0**
		Avg. best	520.1	520.1	520.0	520.3	520.1	520.3	520.2	520.3
		Avg. run time	0.1339	0.1194	0.1225	0.1618	0.1354	0.1388	0.1400	0.3842
6	600	Best	603.2	602.6	602.5	606.5	602.9	606.6	602.9	**602.4**
		Avg. best	605.4	605.0	605.1	608.2	604.7	608.7	605.6	605.7
		Avg. run time	6.2350	6.0260	5.9807	5.9804	5.9678	5.9571	5.9154	5.9758
7	700	Best	**700.1**	**700.1**	**700.1**	718.9	701.7	724.5	702.3	701.3
		Avg. best	700.4	700.4	700.3	741.2	704.3	751.7	710.2	703.7
		Avg. run time	0.2126	0.1927	0.1918	0.2285	0.2031	0.2062	0.2111	0.2004
8	800	Best	805.0	805.0	805.0	826.6	804.8	845.3	813.4	**803.9**
		Avg. best	814.8	814.2	811.3	847.0	813.4	859.4	823.6	812.8
		Avg. run time	0.1002	0.0754	0.0744	0.1125	0.0836	0.0864	0.0932	0.0821
9	900	Best	909.0	907.0	906.0	919.1	904.6	922.2	910.9	**903.3**
		Avg. best	923.8	924.4	917.3	950.5	918.7	951.7	930.1	916.0
		Avg. run time	0.1474	0.1051	0.1121	0.1474	0.1183	0.1207	0.1259	0.1134
10	1000	Best	1010.4	1007.1	1007.1	1129.9	1015.8	1198.8	1101.2	**1006.6**
		Avg. best	1185.8	1188.6	1090.7	1541.1	1100.5	1840.0	1315.1	1132.9
		Avg. run time	0.1630	0.1437	0.1400	0.1792	0.1471	0.1489	0.1569	0.1489
11	1100	Best	1226.0	1231.5	1376.7	1887.8	1198.6	2012.1	1413.8	**1168.7**
		Avg. best	2057.9	2084.5	1797.8	2391.4	1747.4	2391.7	1953.9	1681.5
		Avg. run time	0.1896	0.1700	0.1750	0.2105	0.1823	0.1771	0.1881	0.1817
12	1200	Best	**1200.1**	**1200.1**	1200.2	1200.4	**1200.1**	1200.2	1200.2	**1200.1**
		Avg. best	1200.4	1200.4	1200.3	1200.9	1200.4	1201.0	1200.5	1200.4
		Avg. run time	1.9068	1.7592	1.8349	1.6792	1.6020	1.6468	1.7742	1.6226
13	1300	Best	**1300.2**	**1300.2**	**1300.2**	1301.0	**1300.2**	1300.7	1300.3	**1300.2**
		Avg. best	1300.6	1300.5	1300.4	1301.8	1300.5	1302.2	1300.8	1300.5
		Avg. run time	0.1158	0.0971	0.0990	0.1336	0.0971	0.1039	0.1198	0.0974
14	1400	Best	1400.2	**1400.1**	1400.2	1404.3	**1400.1**	1404.9	1401.8	**1400.1**
		Avg. best	1400.5	1400.6	1400.6	1410.6	1400.4	1412.5	1406.1	1400.5
		Avg. run time	0.1253	0.1014	0.1030	0.1465	0.1066	0.1112	0.1272	0.1047
15	1500	Best	1501.4	1501.8	1501.7	1514.8	1502.5	1517.5	1503.9	**1501.1**
		Avg. best	1505.9	1505.6	1504.1	1551.8	1505.6	1699.0	1508.3	1504.8
		Avg. run time	0.1308	0.1204	0.1201	0.1670	0.1253	0.1388	0.1425	0.1290
16	1600	Best	1602.7	1602.6	1602.8	1602.7	1601.7	1602.5	1602.2	**1601.5**
		Avg. best	1603.6	1603.5	1603.3	1603.3	1602.7	1603.3	1603.1	1602.7
		Avg. run time	0.2356	0.2065	0.2068	0.3395	0.2865	0.3070	0.3082	0.2898
17	1700	Best	1.81E+04	1.82E+04	4.76E+04	8283.2	1.23E+05	1.01E+05	5460.2	**4254.7**
		Avg. best	1.27E+06	1.11E+06	9.76E+05	4.76E+05	6.06E+05	4.47E+05	4.43E+05	4.79E+05
		Avg. run time	0.1691	0.1449	0.1446	0.1814	0.1483	0.1486	0.1553	0.1624
18	1800	Best	2383.9	1872.6	2023.6	7133.7	2166.9	7493.0	1945.3	**1864.4**
		Avg. best	1.53E+04	1.44E+04	1.26E+04	2.43E+04	1.54E+04	3.53E+04	1.83E+04	1.62E+04
		Avg. run time	0.1446	0.1201	0.1213	0.1627	0.1278	0.1314	0.1391	0.1369
19	1900	Best	1901.2	1901.0	1901.1	1905.1	1901.3	1905.0	1904.2	**1900.9**
		Avg. best	1903.1	1903.3	1902.9	1909.8	1902.9	1911.8	1905.8	1902.9
		Avg. run time	1.3422	1.3015	1.3070	1.3716	1.3434	1.3401	1.3370	1.3407
20	2000	Best	2176.2	2034.4	2031.5	2432.7	2045.3	2729.0	2120.3	**2029.8**
		Avg. best	1.13E+04	9434.9	9930.8	1.45E+04	1.28e+04	1.64E+04	1.24E+04	1.10E+04
		Avg. run time	0.1563	0.1351	0.1339	0.1771	0.1391	0.1550	0.1565	0.1425
21	2100	Best	2380.7	4673.8	4518.3	6858.5	1.06E+04	3013.8	2323.0	**2246.6**
		Avg. best	3.32E+05	5.18E+05	4.27E+05	1.49E+05	3.19E+05	2.55E+05	1.81E+05	1.75E+05
		Avg. run time	0.1660	0.1480	0.1492	0.1838	0.1587	0.1580	0.1645	0.1654

TABLE 10. *(Continued.)* **The performance of crossover techniques for the CEC 2014 numerical optimization problems.**

Problem No.	Opt. Sol.	Stat	SPC	TPC	UC	CRC	AC	RC	FRC	GRC
22	2200	Best	2221.1	2221.4	2221.0	2232.1	2221.5	2236.3	2223.3	**2213.6**
		Avg. best	2332.7	2346.5	2324.7	2319.1	2282.1	2330.8	2271.7	2268.3
		Avg. run time	0.3281	0.3011	0.2944	0.3379	0.3057	0.3119	0.3147	0.3088
23	2300	Best	2629.5	2629.5	2629.5	2500.0	2629.9	2500.1	2633.4	**2500.0**
		Avg. best	2629.8	2630.3	2630.2	2644.3	2631.3	2656.0	2643.4	2627.0
		Avg. run time	0.5634	0.5156	0.5009	0.5371	0.5101	0.5092	0.5181	0.8229
24	2400	Best	2519.1	2516.4	2516.3	2567.5	2515.8	2579.5	2526.3	**2515.2**
		Avg. best	2545.4	2539.3	2534.9	2594.3	2532.4	2596.1	2552.2	2534.0
		Avg. run time	0.4197	0.3867	0.3860	0.4026	0.4001	0.3781	0.4063	0.3983
25	2500	Best	2632.9	2649.0	2645.3	2683.0	2625.9	2688.6	2669.0	**2620.5**
		Avg. best	2697.7	2697.2	2693.4	2698.7	2688.7	2699.6	2696.6	2687.2
		Avg. run time	0.4803	0.4473	0.4436	0.4697	0.4516	0.4562	0.4571	0.4544
26	2600	Best	2700.2	2700.2	2700.2	2700.6	2700.2	2700.7	2700.4	**2700.1**
		Avg. best	2700.6	2700.5	2700.4	2701.8	2700.4	2701.6	2700.8	2700.4
		Avg. run time	6.5940	6.6048	6.6498	6.6134	6.6020	6.6192	6.6425	6.7160
27	2700	Best	2705.8	2706.7	2708.0	2861.2	2707.4	2846.9	2743.9	**2704.8**
		Avg. best	3020.4	3027.9	3033.9	3088.7	2982.6	3088.5	3002.8	2924.4
		Avg. run time	6.4917	6.4786	6.5031	6.5668	6.4963	6.6357	6.5322	6.5371
28	2800	Best	3209.5	3187.4	3181.5	3251.8	3175.7	3261.9	3180.9	**3169.9**
		Avg. best	3364.7	3346.4	3288.5	3372.3	3232.4	3496.9	3249.2	3238.1
		Avg. run time	0.7711	0.7368	0.7531	0.9360	0.8900	0.8961	0.9056	0.8964
29	2900	Best	3191.4	3199.5	3190.6	3522.7	3183.0	3779.9	3332.22	**3158.45**
		Avg. best	4.53E+04	3462.2	3476.9	7.94E+04	3562.8	1.20E+05	3982.7	3507.3
		Avg. run time	1.6149	1.5916	1.5861	1.7010	1.6771	1.6710	1.6752	1.6673
30	3000	Best	3674.0	3799.1	3668.8	4966.8	3647.0	5670.2	4355.3	**3618.9**
		Avg. best	4659.4	4722.3	4509.3	1.14E+04	4622.5	1.23E+04	5907.8	4490.8
		Avg. run time	0.6458	0.6241	0.6118	0.6832	0.6501	0.6554	0.6624	0.6547

(problem numbers 23–30), GRC outperforms all the crossover techniques, confirming that GRC is better at exploring and discovering solutions in wide and complex search spaces.

We are also motivated by the previous ring-based method proposed in [12], which demonstrated FRC's outstanding performance in solving the trap problem. We therefore further evaluated the performance of FRC based on the other selected problems and proposed the GRC scheme. In the hierarchical trap problem, which is the most difficult to solve among the selected problems, the proposed GRC method outperforms other techniques. It, however, yields only 88% success at best. Thus, one limitation of ring-based crossover techniques is that they cannot ensure the discovery of the global optimum when problems have both deceptive and hierarchical properties. More complex methods (e.g., HBOA) could solve this kind of problem at the expense of requiring additional computational resources in terms of the number of fitness evaluations. For instance, HBOA computes the conditional probabilities of the bits at all possible chromosome positions, thus requiring 90% more fitness evaluations than GRC. Consequently, ring-based crossover methods are more appropriate for applications involving time-sensitive computations.

In the future, we will enhance the GRC approach to solve hierarchical problems, and overlapping semirings should be further explored. Other future works will include evaluating enhanced GRC methods based on other benchmark and real-world problems in various domains and experimentally and mathematically analyzing the behavior and performance of improved GRC methods.

VI. CONCLUSION

This paper proposed a new crossover technique called generalized ring-based crossover (GRC). This method is a generalization of the existing ring-based crossover techniques, and it incorporates their characteristics. In particular, GRC allows the shifting of bits among chromosomes, semirings reversion, semiring swapping (starting at different positions), and the exchange of front and rear semirings. We evaluated the proposed method based on well-known benchmark problems: one-max, zero-max, royal road, hierarchical if-and-only-if, trap, and hierarchical-trap problems. The proposed method yields superior results (higher discovery rates or fewer fitness evaluations) for all benchmark problems compared with those of traditional crossover methods (SPC, TPC and UC) and other ring-based crossover methods (CRC, AC, RC and FRC). In the small-size one-max and zero-max problems (size $= 30$), the discovery rate of all crossover methods was 100%. In the one-max (size > 30), zero-max (size > 30), royal road, hierarchical if-and-only-if, and trap ($k = 3$) problems, GRC outperformed the traditional crossover methods and yielded the same performance as the ring-based methods in terms of the discovery rate. In the trap ($k = 5$) problem, only GRC and FRC could discover the optimal solution in 100% of cases. For the hierarchical-trap problem, which is considered one of the most difficult problems for GAs to solve, GRC outperforms all the other methods in terms of the discovery rate. In addition, GRC requires the fewest fitness evaluations to solve all problems. Overall, according to the results of all the experiments, GRC requires, on average, 93% and 57% fewer fitness evaluations than traditional methods and other ring-based methods, respectively.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[2] T. Jansen and I. Wegener, "The analysis of evolutionary algorithms—A proof that crossover really can help," *Algorithmica*, vol. 34, no. 1, pp. 47–66, Sep. 2002.

[3] B. Doerr, E. Happ, and C. Klein, "Crossover can provably be useful in evolutionary computation," *Theor. Comput. Sci.*, vol. 425, pp. 17–33, Mar. 2012.

[4] D. Sudholt, "How crossover speeds up building block assembly in genetic algorithms," *Evol. Comput.*, vol. 25, no. 2, pp. 237–274, Jun. 2017.

[5] E. C. Pinto and C. Doerr, "A simple proof for the usefulness of crossover in black-box optimization," in *Proc. PPSN*, vol. 11102, 2018, pp. 29–41.

[6] G. Pavai and T. V. Geetha, "A survey on crossover operators," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–43, Feb. 2017.

[7] T. D. Gwiazda, *Genetic Algorithms Reference: Crossover for Single-Objective Numerical Optimization Problems*, vol. 1. Poland: Tomasz Gwiazda, 2006.

[8] L. Manzoni, L. Mariot, and E. Tuba, "Balanced crossover operators in genetic algorithms," *Swarm Evol. Comput.*, vol. 54, May 2020, Art. no. 100646.

[9] K. B. Ali, A. J. Telmoudi, and S. Gattoufi, "Improved genetic algorithm approach based on new virtual crossover operators for dynamic job shop scheduling," *IEEE Access*, vol. 8, pp. 213318–213329, 2020, doi: 10.1109/ACCESS.2020.3040345.

[10] B. Koohestani, "A crossover operator for improving the efficiency of permutation-based genetic algorithms," *Expert Syst. Appl.*, vol. 151, Aug. 2020, Art. no. 113381.

[11] L. Pan, W. Xu, L. Li, C. He, and R. Cheng, "Adaptive simulated binary crossover for rotated multi-objective optimization," *Swarm Evol. Comput.*, vol. 60, Feb. 2021, Art. no. 100759.

[12] D. Pumsuwan, S. Rimcharoen, and N. Leelathakul, "Front-rear crossover: A new crossover technique for solving a trap problem," in *Proc. 14th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Jul. 2017, pp. 1–6, doi: 10.1109/JCSSE.2017.8025922.

[13] Z. Liang-Jie, M. Zhi-Hong, and L. Yan-Da, "Mathematical analysis of crossover operator in genetic algorithms and its improved strategy," in *Proc. IEEE Int. Conf. Evol. Comput.*, Nov./Dec. 1995, pp. 412–417, doi: 10.1109/ICEC.1995.489183.

[14] B. Pavez-Lazo and J. Soto-Cartes, "A deterministic annular crossover genetic algorithm optimisation for the unit commitment problem," *Expert Syst. Appl.*, vol. 38, no. 6, pp. 6523–6529, Jun. 2011.

[15] Y. Kaya, M. Uyar, and R. Tekin, "A novel crossover operator for genetic algorithms: Ring crossover," *Proc. Inf. Comput. Sci.*, vol. 1, pp. 1286–1292, 2012. [Online]. Available: http://archives.un-pub.eu/index.php/P-ITCS/article/view/868/1093

[16] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "BOA: The Bayesian optimization algorithm," in *Proc. 1st Ann. Conf. Genet. Evol. Comput. (GECCO)*. San Francisco, CA, USA: Morgan Kaufmann, 1999, pp. 525–532.

[17] G. E. Liepins and M. D. Vose, "Characterizing crossover in genetic algorithms," *Ann. Math. Artif. Intell.*, vol. 5, no. 1, pp. 27–34, Mar. 1992.

[18] L. Booker, "Improving search in genetic algorithms," in *Proc. Genet. Algorithm Simulation Annealing*, L. Davis, Ed. San Mateo, CA, USA: Morgan Kaufmann, 1987, pp. 61–73.

[19] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proc. 3rd Int. Conf. Genet. Algorithm*. San Mateo, CA, USA: Morgan Kaufman, 1989, pp. 2–9.

[20] J. C. Culberson, "Mutation-crossover isomorphisms and the construction of discriminating functions," *Evol. Comput.*, vol. 2, no. 3, pp. 279–311, Sep. 1994.

[21] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Proc. 1st Eur. Conf. Artif. Life*. Cambridge, MA, USA: MIT Press, 1992, pp. 245–254.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.

[23] R. A. Watson, G. S. Hornby, and J. B. Pollack, "Modeling building-block interdependency," in *Parallel Problem Solving From Nature*. Berlin, Germany: Springer, 1998, pp. 97–106.

[24] K. Deb and D. E. Goldberg, "Analyzing deception in trap functions," *Found. Genet. Algorithms*, vol. 2, pp. 93–108, Jan. 1993.

[25] S. Nijssen and T. Bäck, "An analysis of the behavior of simplified evolutionary algorithms on trap functions," *IEEE Trans. Evol. Comput.*, vol. 7, no. 1, pp. 11–22, Feb. 2003.

[26] K. Deb and D. E. Goldberg, "Sufficient conditions for deceptive and easy binary functions," *Ann. Math. Artif. Intell.*, vol. 10, no. 4, pp. 385–408, Dec. 1994.

[27] D. E. Goldberg, K. Deb, and J. Horn, "Massive multimodality, deception, and genetic algorithms," in *Parallel Problem Solving From Nature*. Amsterdam, The Netherlands: Elsevier, 1992, pp. 37–45.

[28] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in *Proc. 6th Int. Conf. Genet. Algorithms*, 1995, pp. 184–192.

[29] D. E. Goldberg, "Simple genetic algorithms and the minimal deceptive problem," in *Genetic Algorithms and Simulated Annealing*. San Mateo, CA, USA: Morgan Kaufmann, 1987.

[30] M. Pelikan and D. E. Goldberg, "Escaping hierarchical traps with competent genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2001, pp. 511–518.

[31] P. A. Diaz-Gomez and D. F. Hougen, "Initial population for genetic algorithms: A metric approach," in *Proc. Int. Conf. Genet. Evol. Methods*, 2007, pp. 43–49.

[32] J. Holland, "Building blocks, cohort genetic algorithms, and hyperplane-defined functions," *Evol. Comput.*, vol. 8, no. 4, pp. 373–391, Dec. 2000.

[33] H. M. Pandey, A. Chaudhary, and D. Mehrotra, "A comparative review of approaches to prevent premature convergence in GA," *Appl. Soft Comput.*, vol. 24, pp. 1047–1077, Nov. 2014.

[34] T. Grueninger and D. Wallace, "Multi-modal optimization using genetic algorithms," CAD Lab., Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. 96.02, 1996.

[35] R. A. Rahman and R. Ramli, "Average concept of crossover operator in real coded genetic algorithm," in *Proc. Econ. Develop. Res.*, 2013, pp. 73–77.

[36] R. R. Sharapov, "Genetic algorithms: Basic ideas, variants and analysis," in *Vision Systems: Segmentation and Pattern Recognition*, G. Obinata and A. Dutta, Eds. Rijeka, Croatia: InTech, 2007.

[37] D. E. Goldberg and R. Lingle, Jr., "Alleles, loci, and the traveling salesman problem," in *Proc. 1st Int. Conf. Genet. Algorithms Their Appl.*, 1985, pp. 154–159.

[38] D. Whitley, T. Starkweather, and D. Shaner, "Travelling salesman and sequence scheduling: Quality solutions using genetic edge recombination," in *Handbook of Genetic Algorithms*, L. Davis, Ed. New York, NY, USA: Van Nostrand Reinhold, 1991.

[39] J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization," Nanyang Technol. Univ., Singapore, Tech. Rep. 201311, 2013.

[40] K. Deb and D. Deb, "Analyzing mutation schemes for real-parameter genetic algorithms," *J. Artif. Intell. Soft Comput.*, vol. 4, no. 1, pp. 1–28, 2014.

SUNISA RIMCHAROEN received the B.Sc. degree in computer science from Burapha University, Chon Buri, Thailand, in 2003, and the M.Sc. and Ph.D. degrees in computer science from Chulalongkorn University, in 2005 and 2009, respectively. She has been a Faculty Member of the Faculty of Informatics, Burapha University, since 2009. Her current research interests include evolutionary computation, machine learning, prediction, and algorithmic trading.

NUTTHANON LEELATHAKUL received the B.Eng. degree (Hons.) in electrical engineering from Kasetsart University, Bangkok, Thailand, in 1999, the M.Eng. degree in electrical and computer engineering from Cornell University, in 2004, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, in 2010. He has been a Faculty Member of the Faculty of Informatics, Burapha University, Chon Buri, Thailand, since 2010. His current research interests include computer networking, security in the IoT, deep learning, and data analytics.