International Journal of Electrical and Computer Engineering (IJECE)
Vol. 16, No. 2, April 2026, pp. 883~894
ISSN: 2088-8708, DOI: 10.11591/ijece.v16i2.pp883-894

Elitist genetic algorithm improved with parenting fitness parameter

Ouiss Mustapha, Ettaoufik Abdelaziz, Marzak Abdelaziz
Faculty of Science of Ben Msik, Hassan II University Casablanca, Casablanca, Morocco

Article Info

Article history:

Received Mar 22, 2025
Revised Dec 14, 2025
Accepted Jan 16, 2026

Keywords:

Elitist genetic algorithm
Machine learning
Parenting fitness
Premature convergence
Vehicle routing problem

ABSTRACT

In genetic algorithms, the selection of individuals that will be part of future generations is a critical process of the algorithm. Various strategies exist to select these individuals: the general approach and the elitist approach. The general approach involves replacing the whole current population with the offspring generated so far. The elitist approach introduces a competitive element in which both parents and offspring compete for survival, and only fit individuals will be part of the next generation. While selecting fit individuals helps the algorithm to produce better results, the elitism has a major drawback: the premature convergence, which can limit the algorithm's overall performance. In this article, we compared a typical elitist genetic algorithm and an elitist algorithm improved with the parenting fitness parameter in resolving the vehicle routing problem with drones (VRPD). The parenting fitness parameter helps preserving diversity by retaining parents with high offspring potential despite of their personal fitness. The findings from the study demonstrates that integrating the parenting fitness parameter lead to better results in comparison with a typical elitist genetic algorithm, with relative improvement varying from 1.06% to 10.34% according to the dataset's size.

Corresponding Author:

Ouiss Mustapha
Department of Math and Informatic, Faculty of Science of Ben Msik, Hassan II University Casablanca
Commandant Driss Al Harti Road, Casablanca 20670, Morocco
Email: mustapha.ouiss-etu@etu.univh2c.ma

1. INTRODUCTION

Genetic algorithms (GAs) belong to population-based metaheuristics, and were introduced for the first time in 1975 by Holland [1]. Their nature makes them a specialized form of random heuristic search (RHS) [2]. Genetic algorithms are used to resolve complex and time-consuming problems. Since genetic algorithms use a set of individuals, the algorithm continues evolving these solutions using the selection mechanisms and genetic operations such as crossover and mutation. The canonical version of genetic algorithms, often referred to as the simple genetic algorithm (SGA) was detailed in the work of [2] and [3]. Genetic algorithms have the interesting capacity to find good solutions to complex problems. Genetic algorithms are widely used in the literature, and they are constructed from various components such as chromosome encoding, fitness functions, parent selection mechanisms, genetic operators (crossover and mutation), and the evolution strategies, these components are detailed as follows:

- Population of solutions: Genetic algorithms are population-based metaheuristics, that is, they manipulate a set of chromosomes, also called solutions. The representation of the chromosome depends on the considered problem. Chromosomes can be binary encoded, real encoded, or even a mix of these encoding types.

- Fitness function: The fitness function is the most critical part of the algorithm, also known as the objective function. The value of this function defines the quality of the solution that the genetic algorithms try to optimize. Its implementation varies from one problem to another.
- Parent selection: This process is responsible of selecting solutions that will produce new solutions using genetic operators. The process selects two parents from the population.
- Crossover: The crossover operation mixes the genetic material of the selected parents in order to create offspring.
- Mutation: The mutation operation allows the algorithm to enhance the quality of an offspring, by changing its genes' value, from 0 to 1, for example, in a binary encoded chromosome, or by flipping genes in a string encoded chromosome.
- Evolution scheme: The evolution scheme is the phase in which the genetic algorithm selects solutions that will be part of the next population. This phase is an important part of the algorithm. Existing approaches fall into two strategies: i) general, and ii) elitist. The general approach is the straightforward approach in which the whole population is replaced by the offspring generated so far. In the elitist strategy, n individuals are selected from the current population ($n < N$), where N is the size of the population. These individuals have the best fitness value. The remaining individuals ($N - n$) are being selected from the parents and offspring, since they compete for survival. We may find in the literature variants of these strategies; however, they belong either to the general approach or to the elitist one

Unfortunately, genetic algorithms, as other metaheuristics, can prematurely converge and lose alleles in the run. Premature convergence is a major problem that every metaheuristic suffers from, no exception for genetic algorithms. With this drawback of the genetic algorithm, researchers try to avoid premature convergence using several approaches.

The premature convergence challenge has been addressed through diverse approaches in the literature. Varnamkhasti and Lee [4] tackle the premature convergence problem by dividing the population into a group of male chromosomes and another group of female chromosomes. In the mating phase, a female is selected by a tournament selection, while a male is selected based of hamming distance (HD). In the same goal, another technique, which consist on using chaos theory instead of randomness is used in the work of [5]. Each time a random number is needed by the genetic algorithm, a logistic map, or tent map are used to generate chaotic variables. In the research of [6], the genetic algorithm dynamically changes the genetic operations during the execution to decrease the probability of premature convergence. The algorithm changes the crossover rate and mutation rate during the execution. In the paper of [7], authors present a rank-based selection operator, that aim to avoid premature convergence by applying a balance between diversity and selection pressure. Individuals are prioritized according to their fitness status using weights, and classified into three fit categories; lowest, average and best ones. The selection weights are further employed within each class ensuring population diversity, and higher weights are offered to the most fit individuals to maintain selection pressure. Several works [8]–[10] used a distribution mechanism of genetic algorithm model. The idea is to split the population into independent sub-populations. These subpopulations may have different genetic configurations. Exchanging individuals is possible between sub-populations using the migration mechanism. With this approach, sub-populations evolve in different path, which makes the genetic algorithm explore wide areas. Hybridization of genetic algorithm is used in the work of [11] and [12]. These works use other algorithms to handle the mutation operation. Intervention on both crossover and mutation is applied in the work of [13]–[15], on mutation only in the work of [16], and on crossover only in [17]. Hybridization can also be performed with other metaheuristics [18], [19]. Nicoară [20] introduced two different approaches to alleviate the premature convergence by applying different genetic operators dynamically, and by re-initializing a portion of the population in the run. In the work of [21], authors use a second algorithm called extended-Nelder-Mead (ENM) to enhance the best found solutions produced by the genetic algorithm. Another approach is used in the article of [22], which consists of inserting random genes in the newly created offspring. In the article of [23], a genetic algorithm was proposed that used chromosomes implemented in four layers, each one responsible for encoding certain requirements; hence, the mutation and crossover operations are performed on each layer separately, leading to a diversity in the offspring individuals while avoiding undesirable genes. The algorithm combines the randomly selected dominant individuals using a roulette wheel selection process. These individuals will be kept in the future generation. A summary of these works is provided in Table 1.

While existing research focuses on selection techniques, operator modifications, or hybridization, this article investigates the impact of integrating a parenting fitness parameter [24] into an elitist genetic algorithm to mitigate premature convergence without external algorithms, in optimizing the vehicle routing problem (VRP). The major strength of the parenting fitness is that it can be integrated in existing genetic algorithms with minimum efforts, and without changing the algorithm's core. The implications of this work

extend to real-world applications such as network design and resource allocation among others. Unlike hybridization methods (*e.g.*, [11]), integrating the parenting fitness parameter does not need to use external algorithms to tackle premature convergence.

The remainder of this paper is structured as follows: section 2 details the proposed algorithm. Section 3 describes the experiments. Section 4 presents and discusses the results, and section 5 concludes with future research directions.

Table 1. Summary of literature on premature convergence avoidance in gas

Work	Premature convergence avoidance mechanism
[4]	Population division by sex
[5]	Chaos theory for randomness replacement
[6]	Dynamic adjustment of crossover/mutation rates
[7]	Rank-based selection with weighted fitness classes
[8]–[10]	Parallel subpopulations with migration
[11], [12], [18], [19], [21]	Hybridization with other algorithms
[13]–[15]	Modified crossover and mutation operations
[16], [22]	Mutation adaptations
[17]	Crossover adaptations
[20]	Dynamic genetic operator application
[23]	Custom chromosome encoding

2. ELITIST GENETIC ALGORITHM WITH PARENTING FITNESS

2.1. Parenting fitness: concept and mechanism

The genetic algorithm improved with parenting fitness parameter (eGAwPF) [24] is an elitist algorithm that implements the parenting fitness parameter to tackle premature convergence, in order to obtain better final results. The value of the parenting fitness quantifies the capacity of a parent to produce, from mating, fit offspring. Since the genetic algorithm deals with chromosomes composed by genes considered as building blocks [3] can, by combining them, produce better offspring.

The integration of the parenting fitness parameter should be done with moderate effort, since it requires the following:

- Adding the updating function in the loop;
- Adding three new genes in the chromosome. One can use external arrays for the same goal;
- Adding the selection mechanism of the best parents in the evolution phase.

The Figure 1 illustrates an example of a crossover operation on average fit parents, that will produce offspring with a combination of the parent's building blocks. The genes B1, B2, B3, and B4 are considered in this example a portion of the best final solution. With this philosophy, the best solutions are being constructed, by combining building blocks which are small portions with high potential.

In a genetic algorithm improved with the parenting fitness parameter, individuals with high parenting fitness are being kept through generations, even if their fitness (personal fitness) may be weak. Since the algorithm uses elitism to choose individuals that will survive through generations, it is considered an elitist genetic algorithm.

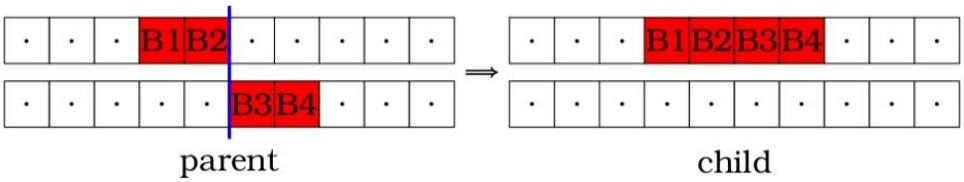

Figure 1. An example of a crossover operation on average fit parents

2.2. Algorithm representation

The genetic algorithm improved with parenting fitness parameter (eGAwPF) [24] illustrated by the Algorithm 1 extends the simple genetic algorithm [2], [3] by integrating the parenting fitness parameter and the parenting fitness calculation function. The Figure 2 illustrates the flow of the algorithm. The integration of the parenting fitness parameter does not perturb the genetic algorithm' core, hence it can be easily implemented in existing algorithms.

Algorithm 1. pseudo-code of a genetic algorithm improved with parenting fitness parameter

```
define crossover rate: χ;
define mutation rate: μ;
Generate initial population P (t = 0);
Initialize pf(i)0 ← −1 ∀i ∈ P
while t # maximal generation number loop do
      Select parents P1(t) and P2(t) from population;
      Generate a random number rndχ in ]0,1[;
      if χ > rndχ then
          Execute crossover operation χ on the selected parents;
          for all child ∈ offspring do
               for all gene ∈ child's genes do
                    Generate a random number rndμ in ]0,1[;
                    if μ > rndμ then
                         Execute mutation operation μ on gene;
                    end if
               end for
          end for
      end if
      for all individual o in the offspring population do
          update parent's parenting fitness of o
      end for
      Prepare the next population;
      end while
```

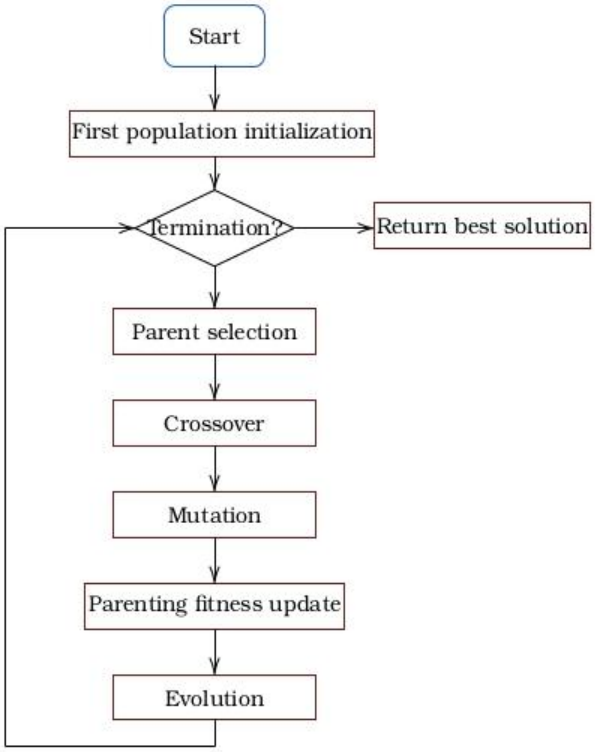

Figure 2. Flow of a genetic algorithm improved with the parenting fitness parameter

2.3. Evolution phase and fitness update

2.3.1. Evolution phase

The Algorithm 2 illustrates how the next population is being processed. A percentage of the best parents is being set in the beginning of the algorithm, denoted as *r*. Next, in the evolution phase, parents and offspring are merged into one population denoted $P_{\mu\lambda}$. The population $P_{\mu\lambda}$ is sorted according to the fitness function of its individuals. Another sorting is executed on the parents μ, this time in a descending way according to the parenting fitness parameter *pf*. A portion of the best parents μ_c equal to $r \times \mu$ is selected to be part of the next population. The remaining places of the next population are being composed with the best individuals of the population $P_{\mu\lambda}$, which is composed of individuals in the set $P_{\mu\lambda}$ - μ_c.

Algorithm 2. Preparation of the next population.

```
r ← percentage of best parents;
P_μλ ← μ+λ;
sorting P_μλ by fitness function f
sorting μ by parenting fitness pf
μ_c ← r×μ
P_n ← []
P_n ← μ_c first parents from μ
P_n ← P_n+(P_μλ-μ_c)
P ← P_n
```

2.3.2. Parenting fitness update

The parenting fitness of parents is updated after the creation of the whole offspring in the current loop. For each offspring, the parenting fitness parameter of its both parents in generation t is updated according to (1):

$$pf(i)_t = pf(i)_{t-1} + d(t)_{im} \tag{1}$$

while the value of $d(t)_{im}$ is calculated by (2).

$$d(t)_{im} = \overline{f} - f(i)_t \tag{2}$$

where $\overline{f}$ defines the mean of the whole offspring's fitness in the current loop, and $f(t)_i$ defines the fitness function of the individual i (the offspring of the current selected parent to update its parenting fitness) in the current loop t. The mean value is calculated according to (3).

$$\overline{f} = \frac{\sum f(i)_t}{N \times 2} \tag{3}$$

where N represents the population size. The parenting fitness of the parents in the current population is updated as follows:

Step 1: The parenting fitness is initialized after the generation of the first population with the default value -1.

Step 2: The genetic algorithm calculates the fitness function of the produced offspring from the mating process of the two selected parents, using the crossover operator. Each offspring has two added parameters: the indexes of its parents, which will be used to update the individuals with these indexes.

Step 3: The parenting fitness is updated at the end of the operations of mating, crossover, and mutation, and before the evolution operation.

Step 4: The fitness function of each offspring is compared to the mean $\overline{f}$ of the newly created offspring. The difference between $\overline{f}$ and the offspring's fitness is denoted $d(t)_{im}$. In this formulation, a value $\overline{f}$ is considered, which is equal to the sum of the fitness values of the offspring population divided by N×2. Since the offspring population is 2 times bigger than the size of the population. For each offspring, the parenting fitness of its parents is updated using (1).

2.4. Chromosome representation

The Figure 3 illustrates a representation of the chromosomes' structure used in the genetic algorithm improved with parenting fitness parameter. The *f* gene defines the fitness function of the chromosome, where the *pf* defines the parenting fitness. The value idx_1 and idx_2 are used to store the indexes of the chromosome's parents, in order to update their parenting fitness value in the corresponding phase.

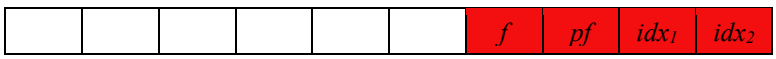

Figure 3. Chromosome integrating the parenting fitness parameter

The Figure 4 illustrates an example of the representation of parents at the end of a loop. For each parent's parenting fitness parameter, a new value is assigned to it. Figure 5 represents an example of an individual that has a good fitness function, but a weak parenting fitness value. On the contrary, the Figure 6 represents a good parent (according to its parenting fitness), but with a weaker fitness function. Since we are dealing with a minimizing problem, the best individuals are those with a low fitness value. The Algorithm 3 shows the parenting fitness updating phase.

6	2	17	8	...	18	9	1	13	187	24	-1	-1
6	14	8	20	...	10	18	4	1	203	24	-1	-1
9	14	15	12	...	18	19	20	13	208	-79	-1	-1
12	17	19	10	...	14	9	3	18	192	-60	-1	-1

Figure 4. Example of representation of parents in the end of a loop

12	17	19	10	...	14	9	3	18	192	-60	-1	-1

Figure 5. Example of representation of an individual with good fitness and weak parenting fitness

6	14	8	20	...	10	18	4	1	203	24	-1	-1

Figure 6. Example of representation of an individual with weak fitness and good parenting fitness

Algorithm 3. Parenting fitness updating phase

```
Initialize pf (i)_0 ← −1, ∀i ∈ P
    for each offspring o ∈ O do
        Calculate d(t)_im via Eq. 2
        Update pf (i)_t ← pf (i)_t−1 + d(t)_im
    end for
Preparation of the next population;
```

3. EXPERIMENTS

In order to comprehensively evaluate the efficiency of our algorithm improved with the parenting fitness, we have selected a challenging optimization problem to serve as our testing ground. Specifically, we have opted to work on the VRP, with a particular focus on the vehicle routing problem with drones (VRPD) variant. VRPD is an extension of the well-known capacitated vehicle routing problem (CVRP), which was introduced for the first time in the work of [25]. One particular aspect that sets VRP apart is the fact that allele loss is not a concern, as the problem formulation ensures that all nodes are included in the chromosome encoding. In this type of optimization problem, the primary challenge lies in the tendency to get stuck in local optima. The population composed of N individuals is randomly generated at the beginning of the algorithm. Each individual is composed of *l* genes, where *l* represents the number of nodes in the dataset. An individual (or solution) represents the total set of nodes clustered to the departure node N_0, that a drone should visit.

The Table 2 indicates the selected genetic parameters for both eGAwPF and eGA. Both algorithms share common parameters, such as the number of generations and the population size. The genetic algorithms (eGA and eGAwPF) were implemented using the Python language, and executed on an Intel ® i3 machine, with a dual-core 2.20 GHz, and 4 Go RAM. Also, both algorithms were executed under the same stochastic conditions, that is, no random seed was used in the experiments.

Table 2. Selected genetic parameters for both eGAwPF and eGA

Parameter	Value
Population size	N × 2 (N is the size of the dataset)
Number of loops	N × 2
Parent selection	2-Tournament
Crossover operation	Order crossover operator
Crossover rate	0.75
Mutation operation	Swap mutation
Mutation rate	0.02

3.1. Mathematical formulation of the VRPD

In the VRPD, the objective function is to minimize the maximum distance of all routes performed by the drone. The VRPD mathematical formulation is adapted from the well-known flying sidekick traveling salesman problem (FSTSP) [26]. We consider the following notations and constraints [24]:

- The set of nodes N ($N = 1, ..., n$);
- The departure and final arrival node N_0;
- The set of nodes N_+ to visit;
- Distance $L_{i,j}$ between i and j;
- Set R defining all routes performed by a drone;
- Maximum flight range mf_d of the drone d.

The objective function is formulated as:

$$min \sum_{i=1}^{N} R_i \quad (4)$$

The constraint 5 ensure that the route R_i do not exceed the maximum flight distance of the drone d:

$$R_i \leq d \quad (5)$$

In these experiments, no constraint is considered for the maximum drone load. We consider the main depot as a take-off node and the landing node. Constraint 6 ensures that the drone takes off from the central main depot exactly once, while the constraint 7 requires the drone to return to the main depot exactly once.

$$\sum_{j \in N} x_{0j} = 1 \quad (6)$$

$$\sum_{j \in N} x_{j0} = 1 \quad (7)$$

The constraint 8 ensures that a node is visited exactly once.

$$\sum_{i,j \in N} x_{ij} + \sum_{i,j \in N} y_{ij} = 1 \; i \neq j \quad (8)$$

3.2. Fitness function

To compare the effectiveness of our approach against a typical elitist genetic algorithm (eGA), we implemented the exact same fitness function, for the reason that a fitness function implementation is a critical part of the genetic algorithm, and a wrong implementation may lead to wrong calculations either in a positive way or a negative one. The fitness function calculation for the VRPD is defined in the Algorithm 4, which corresponds to the decomposition of the big tour defined by the chromosome, into feasible routes.

Algorithm 4. VRPD Sub-tours construction

```
Initialization of the Maximum Route Length mrl ← 0;
   mf_d ← Maximum flight distance of the drone d;
   mrl ← distance(depot, chromosome(0));
for all gene i, gene i+1 ∈ chromosome do
      md_{i,i+1} ← distance(i,i+1);
if mrl + md_{i,i+1} > mf_d then create new route (return to the depot, and from depot to the
position defined in the gene I + 1;
   else
   mrl ← mrl + md_{i,i+1} ;
   end if
   end for
   mrl ← mrl + distance(chromosome(L), depot); // L: chromosome length.
```

3.3. Dataset description

To test the algorithm, we used multiple datasets downloaded from the instances presented in the work of [27]. Each instance file is labelled N.M.T, where N is the number of customers, M is the grid's dimension, and T is the scenario's generic name. The first line in each instance file indicates the number of customers. Each line from the second line contains coordinates X, Y and demand of the customer. Another file containing information about the coordinates of the depot, and information about the used drone, is used.

3.4. Route construction function

In the vehicle routing problem optimization, the main goal is to construct routes that respect defined constraints and objectives. The fitness of each chromosome in the loop t is calculated with (9).

$$f_i(t) = \sum_{i=1}^{k} R_i \quad (9)$$

In the VRP, chromosomes define the sequence of nodes visited by the vehicle. The algorithm 4 represents the pseudo-code of the decomposition of the big tour into sub-tours. The distance function is a function that returns the distance between the node *i* and the node *i* +1. The distance is calculated by (10).

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad (10)$$

A gap value between the best of eGAwPF for each percentage and the best of eGA is defined by (11):

$$Gap_1 = (B_{ega} - B_{pf}) / B_{pf} \quad (11)$$

while the gap value between the overall best of eGAwPF and the best of eGA is defined by (12):

$$Gap_2 = (B_{ega} - B^*_{pf})/B^*_{pf} \quad (12)$$

where B_{ega} is the best-known solution (BKS) of the eGA and B_{pf} is the best-known solution of the elitist genetic algorithm with parenting fitness for a particular parenting fitness percentage, and B^*_{pf} is the overall best of eGAwPF.

3.5. Crossover

The crossover controls the diversification mechanism. We consider the order crossover operator (OX) [28], [29]. Algorithm 5 presents the Order crossover operator steps. The order crossover operates as follows:

- Two parents $P_1(t)$ and $P_2(t)$ are selected from the population according to a defined strategy.
- Crossover points crp_1 and crp_2 are chosen randomly in the interval [2, L - 1], where L is the length of the chromosome, and $crp_1 > crp_2$.
- Genes of the parents are swapped to create new offspring $O_1(t)$ and $O_2(t)$ on the crossover points.
- The redundant genes of the offspring are removed, and the gaps are filled with the remaining values.

Algorithm 5. Order crossover (OX) operator pseudocode

```
Select parents P1(t) and P2(t) from population;
Select crossover points crp1 and crp2;
Swap Genes of P1(t) and P2(t) between crp1 and crp2;
Remove redundant genes from the offspring O1(t) and O2(t);
for all offspring O do
   Fill empty genes with remaining values;
   end for
```

3.6. Mutation

The natural mutation operator for the vehicle routing problem, as well as similar combinatorial optimization problems, is the swap mutation (SM) operator. In the context of swap mutation, the genetic algorithm selects two positions rnd_1 and rnd_2 (where rnd_1 # rnd_2) at random on the chromosome, and interchanges their values. The algorithm 6 represents the swap mutation operation.

Algorithm 6. Swap mutation (SM) operator pseudocode

```
Generate a random numbers rnd1 in [1, n];
Generate a random numbers rnd2 in [rnd1, n];
tmp ← individual_i[rnd1];
individual_i[rnd1] ← individual_i[rnd2];
individual_i[rnd2] ← tmp;
```

4. RESULTS AND DISCUSSION

4.1. Sensitivity analysis

In order to study the variation impact of the parenting fitness initialization between loops, the genetic algorithm improved with parenting fitness parameter was executed in two versions: i) with reinitialization of the parenting fitness of parents, and ii) without reinitialization. In the first set, we reinitialize the parenting fitness of the parents at each iteration, while in the second set, the value of the parenting fitness remains after the loop. Also, the parenting fitness rate is set to 2%, 10%, and 50% of the best parents. The remaining individuals are the best of the combined offspring and individuals of the current population. Executing the algorithm with different parenting fitness percentages offers a perspective on the impact of different percentages on the algorithm's performance and its final result.

4.2. Results

For the experiments on the eGAwPF without reinitialization of parenting fitness, Tables 3, 4, and 5 represent the corresponding results for small, medium, and large instances, respectively. The results obtained from our research are presented, for the experiments on the eGAwPF with reinitialization of parenting fitness in Table 6, 7, and 8 for small, medium, and large instances, respectively.

Table 3. Computational results on small instances, with EGAwPF without reinitialization

	eGA	EGAwPF (50%)		EGAwPF (10%)		EGAwPF (2%)		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
20.10.3	62.0	55.0	0.127	53.0	0.170	59.0	0.051	53.0	0.170
20.10.4	51.0	65.0	-0.215	65.0	-0.215	67.0	-0.239	65.0	-0.215
20.5.2	30.0	33.0	-0.091	28.0	0.071	27.0	0.111	27.0	0.111
20.5.3	29.0	28.0	0.036	26.0	0.115	26.0	0.115	26.0	0.115
20.5.4	31.0	31.0	0.000	26.0	0.192	26.0	0.192	26.0	0.192

Table 4. Computational results on medium instances, with EGAwPF without reinitialization

	eGA	EGAwPF (50%)		EGAwPF (10%)		EGAwPF (2%)		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
50.20.4	653.0	690.0	-0.054	**639.0**	0.022	691.0	-0.055	**639.0**	0.022
50.30.1	989.0	**983.0**	0.006	1052.0	-0.060	1016.0	-0.027	**983.0**	0.006
50.30.2	906.0	**885.0**	0.024	**889.0**	0.019	**901.0**	0.006	**885.0**	0.024
50.30.3	1060.0	**1017.0**	0.042	**1053.0**	0.007	**1017.0**	0.042	**1017.0**	0.042
50.30.4	**934.0**	959.0	-0.026	956.0	-0.023	967.0	-0.034	959.0	-0.026

Table 5. Computational results on large instances, with EGAwPF without reinitialization

	eGA	EGAwPF (50%)		EGAwPF (10%)		EGAwPF (2%)		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
200.30.2	4456.0	**4325.0**	0.030	**4153.0**	0.073	**4298.0**	0.037	**4153.0**	0.073
200.30.3	**4447.0**	4548.0	-0.022	4582.0	-0.029	4545.0	-0.022	4545.0	-0.022
200.30.4	4286.0	**4094.0**	0.047	4304.0	-0.004	**4260.0**	0.006	**4094.0**	0.047
200.40.1	5786.0	**5695.0**	0.016	5799.0	-0.002	5806.0	-0.003	**5695.0**	0.016
200.40.2	5970.0	6020.0	-0.008	**5955.0**	0.003	6169.0	-0.032	**5955.0**	0.003

Table 6. Computational results on small instances, with EGAwPF with reinitialization

	eGA	EGAwPF 50%		EGAwPF 10%		EGAwPF 2%		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
20.10.3	62.0	**57.0**	0.088	**47.0**	0.319	**60.0**	0.033	**47.0**	0.319
20.10.4	**51.0**	60.0	-0.150	61.0	-0.164	63.0	-0.190	60.0	-0.150
20.5.2	30.0	33.0	-0.091	31.0	-0.032	**29.0**	0.034	**29.0**	0.034
20.5.3	29.0	**24.0**	0.208	**26.0**	0.115	30.0	-0.033	**24.0**	0.208
20.5.4	31.0	**22.0**	0.409	**30.0**	0.033	**27.0**	0.148	**22.0**	0.409

Table 7. Computational results on medium instances, with EGAwPF with reinitialization

	eGA	EGAwPF 50%		EGAwPF 10%		EGAwPF 2%		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
50.20.4	653.0	662.0	-0.014	663.0	-0.015	**638.0**	0.024	**638.0**	0.024
50.30.1	989.0	1067.0	-0.073	**983.0**	0.006	1005.0	-0.016	**983.0**	0.006
50.30.2	906.0	925.0	-0.021	**869.0**	0.043	**826.0**	0.097	**826.0**	0.097
50.30.3	**1060.0**	1113.0	-0.048	1085.0	-0.023	1061.0	-0.001	1061.0	-0.001
50.30.4	**934.0**	1050.0	-0.110	974.0	-0.041	1044.0	-0.105	974.0	-0.041

Table 8. Computational results on large instances, with EGAwPF with reinitialization

	eGA	EGAwPF 50%		EGAwPF 10%		EGAwPF 2%		B^*_{pf}	GAP2
Instance	Best	Best	GAP1	Best	GAP1	Best	GAP1		
200.30.2	4456.0	**4379.0**	0.018	**4419.0**	0.008	**4301.0**	0.036	**4301.0**	0.036
200.30.3	**4447.0**	4541.0	-0.021	4521.0	-0.016	4593.0	-0.032	4521.0	-0.016
200.30.4	**4286.0**	4310.0	-0.006	4302.0	-0.004	4302.0	-0.004	4302.0	-0.004
200.40.1	5786.0	5867.0	-0.014	**5676.0**	0.019	**5762.0**	0.004	**5676.0**	0.019
200.40.2	5970.0	5972.0	0.000	6006.0	-0.006	**5881.0**	0.015	**5881.0**	0.015

4.3. Execution analysis

The genetic algorithm uses four functions: parents' selection, crossover, and mutation, each of these functions have a complexity of O(n). However, the genetic algorithm runs n iterations; hence, the overall complexity is O(n^2). The complexity of the parenting fitness function is O(n), since it uses n iterations while updating the parents' parenting fitness.

4.4. Relative improvement

Let N, E be the mean value of results found by eGA and the best results of eGAwPF for small, medium, and large instances. The used equation to calculate the relative improvement is defined by (14), while Tables 9 and 10 represent the results of the calculation.

$$\Delta = \frac{\bar{N} - \bar{E}}{\bar{N}} \times 100 \quad (14)$$

Table 9. Relative improvement with reinitialization of parenting fitness

		eGA	EGAwPF
Small instances	Mean	40.6	36.4
	Median	31	29
	Relative improvement (%)		**10.34**
Medium instances	Mean	908.4	896.4
	Median	934	974
	Relative improvement (%)		**1.32**
Large instances	Mean	4989	4936.2
	Median	4456	4521
	Relative improvement (%)		**1.06**

Table 10. Relative improvement without reinitialization of parenting fitness

		eGA	EGAwPF
Small instances	Mean	40.6	39.4
	Median	31	27
	Relative improvement (%)		**2.96**
Medium instances	Mean	908,4	896.6
	Median	934	959
	Relative improvement (%)		**1.3**
Large instances	Mean	4989	4888.4
	Median	4456	4545
	Relative improvement (%)		**2.02**

4.5. Discussion

The eGAwPF shows significant performance in comparison with a standard eGA. The eGAwPF was executed in two version: one without reinitialization of the parenting fitness value of parents, and another with reinitialization. For the first execution, the overall gap in performance, for the used datasets, ranges from 0.111 to 0.192, 0.006 to 0.042, 0.003 to 0.073, for small, medium and large instances, respectively. The latter shows an overall performance gaps, that varied from 0.034 to 0.409, 0.006 to 0.097, and 0.015 to 0.036 for small, medium and large instances, respectively. However, in some cases, the eGA outperforms the eGAwPF, however, even in these cases (except for the dataset 20.10.4 in the first set of experiments), the gap does not exceed the performance ranges observed when eGAwPF outperforms eGA. Also, the non-reinitialization of the parenting fitness can produce more fit individuals. The experiments conducted demonstrates that the parenting fitness parameter enhances the performance of the genetic algorithm. Unlike other techniques as hybridization, that may need significant updates in the genetic algorithm's core, the parenting fitness parameter can be easily integrated into existing GA frameworks in a non-disturbing way with minimum effort and reduced changes, since it only requires adding the updating function and adapting the evolution phase.

5. CONCLUSION AND FUTURE DIRECTIONS

In this article, we investigated the impact of integrating the parenting fitness parameter into an elitist genetic algorithm to mitigate premature convergence in the context of optimizing the vehicle routing problem. Our experiments revealed a relative improvement of 1.06 to 10.34% depending on the dataset's size and the parenting fitness reinitialization approach. These experiments validate that preserving high-quality

parents enhances the final results of the algorithm. Instead of focusing only on individuals' fitness, the use of the parenting fitness allows the algorithm to take advantage of their potential. Being one of the most known metaheuristics, the genetic algorithm, despite of existing for more than five decades, presents opportunities to be improved to produce better results. In future work, we will investigate a mechanism that use the correlation between the fitness function of the individual with its parenting fitness value. This correlation will be used in the selection phase to ensure the balance between being the best solution and the best parent. Also, upgrading the algorithm to handle multi-objective problems should be explored.

FUNDING INFORMATION

The authors state no funding is involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Ouiss Mustapha	✓	✓	✓	✓	✓	✓		✓	✓					
Ettaoufik Abdelaziz				✓		✓				✓	✓	✓		
Marzak Abdelaziz		✓				✓						✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors state no conflict of interest.

DATA AVAILABILITY

The datasets used during the current study are available in: *https://zenodo.org/record/1403150/*

REFERENCES

[1] J. Holland, *Adaptation in natural and artificial systems*, 2nd ed. The MIT Press, 1992.
[2] M. D. Vose, "The simple genetic algorithm: foundations and theory," *The Simple Genetic Algorithm*, vol. Complex ad, pp. 21–38, 2022, doi: 10.7551/mitpress/6229.003.0007.
[3] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, vol. 27, no. 02. Addison-Wesley Publishing Company, inc, 1989.
[4] M. Jalali Varnamkhasti and L. S. Lee, "A fuzzy genetic algorithm based on binary encoding for solving multidimensional knapsack problems," *Journal of Applied Mathematics*, vol. 2012, 2012, doi: 10.1155/2012/703601.
[5] M. Javidi and R. Hosseinpourfard, "Chaos genetic algorithm instead genetic algorithm," *International Arab Journal of Information Technology*, vol. 12, no. 2, pp. 163–168, 2015.
[6] H. S. Mojaveri, "Utilizing model knowledge for design developed genetic algorithm to solving FJS|rj|FMas problem," *Int. Journal of Professional Business Review*, vol. 3, no. 2, pp. 172–186, 2018, doi: 10.26668/businessreview/2018.v3i2.49.
[7] A. Hussain and S. A. Cheema, "A new selection operator for genetic algorithms that balances between premature convergence and population diversity," *Croatian Operational Research Review*, vol. 19, pp. 107–119, 2020, doi: 10.17535/crorr.2020.0009.
[8] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 43–62, 2000, doi: 10.1109/4235.843494.
[9] K. Hao, J. Zhao, K. Yu, C. Li, and C. Wang, "Path planning of mobile robots based on a multi-population migration genetic algorithm," *Sensors (Switzerland)*, vol. 20, no. 20, pp. 1–23, 2020, doi: 10.3390/s20205873.
[10] G. Takasao, T. Wada, H. Chikuma, P. Chammingkwan, M. Terano, and T. Taniike, "Preventing premature convergence in evolutionary structure determination of complex molecular systems: demonstration in few-nanometer-sized TiCl4-Capped MgCl2Nanoplates," *Journal of Physical Chemistry A*, vol. 126, no. 31, pp. 5215–5221, 2022, doi: 10.1021/acs.jpca.2c02112.
[11] M. H. Abed and M. N. Mohmad Kahar, "Hybridizing genetic algorithm and single-based metaheuristics to solve unrelated parallel machine scheduling problem with scarce resources," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 12, no. 1, 2023, doi: 10.11591/ijai.v12.i1.pp315-327.
[12] P. G. Luan and N. T. Thinh, "Hybrid genetic algorithm based smooth global-path planning for a mobile robot," *Mechanics Based Design of Structures and Machines*, vol. 51, no. 3, 2023, doi: 10.1080/15397734.2021.1876569.
[13] J. Zhang, J. Zhuang, H. Du, and S. Wang, "Self-organizing genetic algorithm based tuning of PID controllers," *Information Science.*, vol. 179, no. 7, 2009, doi: 10.1016/j.ins.2008.11.038.

[14] T. N. Fatyanosa and M. Aritsugi, "An automatic convolutional neural network optimization using a diversity-guided genetic algorithm," *IEEE Access*, vol. 9, pp. 91410–91426, 2021, doi: 10.1109/ACCESS.2021.3091729.
[15] S. Z. Ramadan, "Reducing premature convergence problem in genetic algorithm: application on travel salesman problem," *Computer and Information Science*, vol. 6, no. 1, 2012, doi: 10.5539/cis.v6n1p47.
[16] D. W. Gong, M. Q. Zhu, X. J. Guo, and M. Li, "Genetic algorithm based on chaotic mutation to deal with premature convergence," *Kongzhi yu Juece/Control & Decision*, vol. 18, no. 6, 2003.
[17] E. G. Boudissa, F. Habbi, N. E. H. Gabour, and M. Bounekhla, "A new dynamic genetic selection algorithm: Application to induction machine identification," *Revue Roumaine des Sciences Techniques Serie Electrotechnique et Energetique*, vol. 66, no. 3, pp. 145–151, 2021.
[18] G. L. Du and N. Xue, "The research on mutil-objective location routing problem based on genetic simulated annealing algorithm," *Applied Mechanics and Materials*, vol. 543--547, 2014, doi: 10.4028/www.scientific.net/AMM.543-547.2842.
[19] S. Li, Y. Liu, and Y. Feng, "Identification of the damping coefficients in dynamic system using hybrid genetic algorithm," *Jisuan Lixue Xuebao/Chinese J. Comput. Mech.*, vol. 21, no. 5, 2004.
[20] E. Nicoară, "Mechanisms to avoid the premature convergence of genetic algorithms," *Petroleum-Gas University of Ploiesti Bulletin, Mathematics-Informatics-Physics Series*, vol. 61, no. 1, 2009.
[21] W. Musa, K. R. Ku-Mahamud, S. Salim, and A. Sediyono, "Hybrid real-value-genetic-algorithm and extended-Nelder- mead algorithm for short term energy demand prediction," *Journal of Information and Communication Technology*, vol. 23, no. 1, 2024, doi: 10.32890/jict2024.23.1.2.
[22] A. T. Busayo, Z. Mohamad, N. A. Mahiddin, and W. N. S. W. Nik, "Workflow scheduler optimization using an enhanced hybrid genetic algorithm," *International Journal of Advanced Technology and Engineering Exploration*, vol. 11, no. 111, 2024, doi: 10.19101/IJATEE.2023.10102108.
[23] S. Jia and others, "The green flexible job-shop scheduling problem considering cost, carbon emissions, and customer satisfaction under time-of-use electricity pricing," *Sustainability*, vol. 16, no. 6, 2024, doi: 10.3390/su16062443.
[24] M. Ouiss, A. Ettaoufik, A. Marzak, and A. Tragha, "Genetic algorithm parenting fitness," *Mathematical Modeling and Computing*, vol. 10, no. 2, pp. 566–574, 2023, doi: 10.23939/mmc2023.02.566.
[25] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, Oct. 1959, doi: 10.1287/mnsc.6.1.80.
[26] C. C. Murray and A. G. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 86–109, 2015, doi: 10.1016/j.trc.2015.03.005.
[27] D. Sacramento, D. Pisinger, and S. Ropke, "An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones," *Transportation Research Part C: Emerging Technologies*, vol. 102, pp. 289–315, 2019, doi: 10.1016/j.trc.2019.02.018.
[28] P. Moscato, "On genetic crossover operators for relative order preservation," Technical report, 1989.
[29] İ. İlhan, "An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem," *Swarm and Evolutionary Computation*, vol. 64, 2021, doi: 10.1016/j.swevo.2021.100911.

BIOGRAPHIES OF AUTHORS

Ouiss Mustapha received the master degree in computer science from Mohammed V University-Rabat, Faculty of Sciences, Rabat. His research field of interest includes evolutionary algorithms, high performance computing. He can be contacted at email: ouissm@gmail.com.

Ettaoufik Abdelaziz is Professor in the Department of Mathematics and Informatics at Faculty of Sciences Ben M'Sik, Hassan II University, Morocco. His research field of interest includes data warehouse, cloud computing and optimization. He can be contacted at email: aettaoufik@gmail.com.

Marzak Abdelaziz received his Ph.D. in computer science from Mohammed V University-Rabat, National School of Computer Science and Systems Analysis, Rabat, Morocco, 2000. Currently, he is a Professor of Computer Science and director of the Information Technology and Modeling Laboratory at the Faculty of Science Ben M'sik, Casablanca, Morocco. His research interests include agile software development, metamodeling, the applications of artificial intelligence, software engineering, data mining, educational data mining, machine learning, and the internet of things. He can be contacted at email: marzak@hotmail.com.